The Australian National University

# Web Based Operator Interface for an Underwater Robot

By

***Sunil Rao***

***(3073691)***

A thesis submitted in partial fulfilment of the requirements for the degree of
**Bachelor of Engineering**
June 2000

## Supervisors

**Mr. Samer Abdallah (FEIT)**
**Dr. David Wettergreen (RSISE)**
**Prof. Alexander Zelinsky (RSISE)**

RSISE – Research School of Information Sciences and Engineering, Australian National University
FEIT – Faculty of Engineering and Information Technology, Australian National University

# ACKNOWLEDGEMENTS

The author wishes to thank Alex Zelinsky, Samer Abdallah and David Wettergreen for giving me the opportunity to work on this project as well as for their continuous assistance and guidance throughout the project.

Thanks to Chris Gaskett and Chanop Silpa-Anan for their help, ideas and tips on various aspects of the project. Big thanks also to Peter Brown for his help in establishing the connection to Kambara and communicating with it. I do not believe that I would have been able to make much progress without their assistance.

I also wish to thank Seungjun Oh, Matthew Bryant, Harley Truong, Jordan Cvetanovski, Colin Thomsen and Orson Sutherland for their excellent company, their ideas, motivation and help when things were not going smoothly. Thanks to Colin and Matthew again for the lively games of chess.

Thanks also to Arvind Venkataraman and Chaitanya Kuber for proof reading my thesis and providing me with valuable comments. Thanks to my sister Sneha as well for her help throughout these last four months. And finally to my parents – thanks for your encouragement and understanding through these last four months. Could not have done it without you.

# ABSTRACT

This thesis examines the continued development of the operator interface for Kambara, an Autonomous Underwater Vehicle being developed at the ANU. The interface receives state information from the robot and presents it to the operator in intuitive graphical representations. It is also supposed to provide means for the operator to send commands to the robot.

The project goal was to develop a web-based interface that would provide adequate guidance for the robot to perform useful tasks. To this end, the project has developed a system that can be executed on a web browser such as Netscape.

The system developed also aids direct teleoperation of the robot through the use of a software joystick for manipulating the five degrees of freedom of the AUV. The redesign of the existing GUI has aided the process by having the camera controls in constant view of the operator. An investigation has also been conducted regarding the use of VRML 3D models for better visualisation of the rotation and translation state of the robot.

The system uses RMI communication methods in Java 2 to develop cross-network capabilities. The current network design conforms to the long-term goal defined for the interface. The interface receives telemetry router data in real time from the robot. The update of the GUI is at the rate of 1.24 Hz if the server is located on the system but 6.58 Hz if the server is on a separate machine. In an ideal Ethernet environment with no traffic the values are 3.5 Hz with the server on the system and 9.57 Hz when the server is on a separate machine.

Camera commands have also been developed for the client and can be sent to the camera on-board Kambara. These were tested using a Sony D-230 camera, which was located on-board as well as off-board Kambara during various stages of implementation and testing. Enhancement of the existing control system has also been achieved with progress made in the development of teleoperation and individual control.

# GLOSSARY

| | |
|---|---|
| **3D** | Three Dimensional |
| **ANU** | The Australian National University |
| **API** | Application Programming Interface |
| **Applet** | A program that is designed to be run using a Web browser |
| **Application** | A program that is run from a command line |
| **AUV** | Autonomous Underwater Vehicle |
| **Double** | A "primitive" Java datatype that represents a 64-bit floating-point value (IEEE 754- 1985) |
| **GUI** | Graphical User Interface |
| **JDK** | Java Developers Kit |
| **JVM** | Java Virtual Machine |
| **Long** | Another "primitive" datatype representing a 32-bit floating-point value (IEEE 754-1985) |
| **NCSA** | National Centre for Supercomputing Applications, University of Illinois, USA |
| **Packet** | Bundle of information that is sent across a network connection. It is the simplest form of a data unit that carries the address of the destination and the data that is to be sent to that destination |
| **RMI** | Remote Method Invocation |
| **RSISE** | Research School of Information Sciences and Engineering, ANU |
| **TCP / IP** | Transmission Control Protocol / Internet Protocol. A connection-oriented approach of communication between entities |
| **Thread** | Objects that are used to have more than one set of operations executing in the same program at one time. |
| **UDP** | User Datagram Protocol. A connectionless approach of transmitting data between entities |
| **VRML** | Virtual Resource Modelling Language |

# LIST OF FIGURES

# TABLE OF CONTENTS

# CHAPTER 1: INTRODUCTION

Oceans cover over 70% of the earth's surface. Yet, not much is known about the underwater environment with 94% of the ocean floor never having been explored or charted in a detailed manner. As the world's population increases, the world-wide consumption of natural resources will also increase leading to an inevitable exploration of the oceans. For people to better understand and manage this environment, it is vitally important that accurate and detailed information can be obtained.

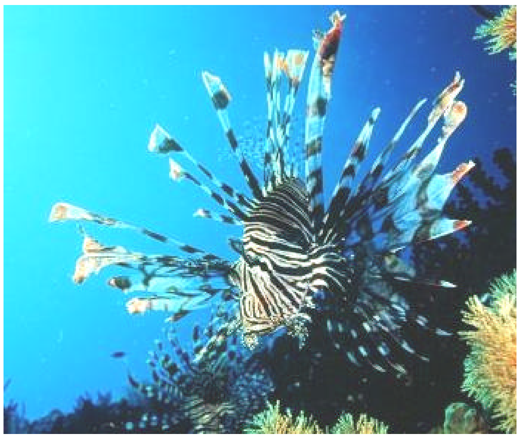


**Figure 1: Life on the Great Barrier Reef**

This is particularly important for Australia with its vast areas of coastal waters containing vast biological and mineralogical resources, which are largely unexplored and unknown. They must be investigated to allow wise development and proper protection. The task of preserving these resources brings with it the problem of development of vehicles that would assist in fulfilling such objectives.

At the ANU[1], an Autonomous Underwater Vehicle (AUV) is under development for tasks in exploration and inspection. It is named Kambara and would help provide convenient, cost effective solutions for developing and preserving areas such as the Great Barrier Reef. The main objectives of this effort are to enable underwater robots to autonomously maintain their position, follow along fixed natural and artificial features, search in regular patterns, and ultimately, to swim after dynamic targets. These capabilities are essential to tasks like cataloguing reefs, exploring geologic features, and studying marine creatures, as well as inspecting pipes and cables, and assisting divers.

Kambara's mechanical structure, shown in Figure 2, is an open-frame rigidly supporting five thrusters and two watertight enclosures. It is a simple, low-cost vehicle suitable as a test-bed for shallow water research in underwater robot autonomy [2]. Kambara's thrusters enable roll, pitch, yaw, heave, and surge manoeuvres. It is underactuated and not able to perform direct sway (lateral) motion; it is non-holonomic.

[1] More specifically, the Department of Systems Engineering in the Research School of Information Sciences and Engineering (RSISE)

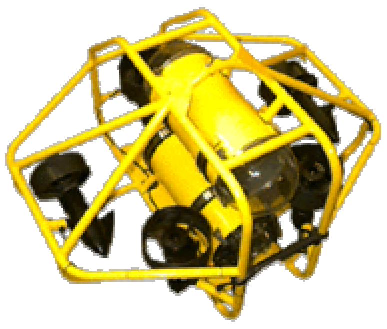


**Figure 2 Kambara**

Kambara is also equipped with power, electronics on-board sensing and computing resources. The system on-board Kambara can learn to distinguish relevant sensor information and to co-ordinate the action of its thrusters to move purposefully [3].

This project involves the design and development of advanced web-based operator interfaces for underwater robots. The aim of this project is to develop a specific instance for Kambara. The interface will receive state information from the robots and present it to the operator in intuitive graphical representations.

The interfaces will also have mechanisms for the operator to define and send commands to the robot. This would be done via mouse, keyboard or a combination, to indicate the desired actions. The variations in the state of Kambara are then viewed via the state information displayed on the interface.

The Operator Interface development was started in 1999. The foundations for the basic interface and network architecture have already been laid [1]. The key software goals achieved included:

- Presenting robot telemetry information to the operator in intuitive graphical representations.
- Development of a 3D model and the use of Java3D in displaying the model to the operator.
- Basic client-server network architecture development cross-network capabilities built into the interface, thus allowing more than one user to connect to the robot.

## *1.1 Description and Contribution*

### 1.1.1 Kambara Software Architecture

Kambara's software architecture is designed to allow autonomy at various levels:

- o at the signal level for adaptive thruster control,
- o at the tactical level for competent performance of primitive behaviours and,
- o at strategic level for complete mission autonomy [4].

The software architecture details the overall structure and collective behaviour of the software system. The software modules are designed as independent computational processes that communicate as shown in Figure 3. As can be seen in the figure, the operator interface is only a small part of the architecture.

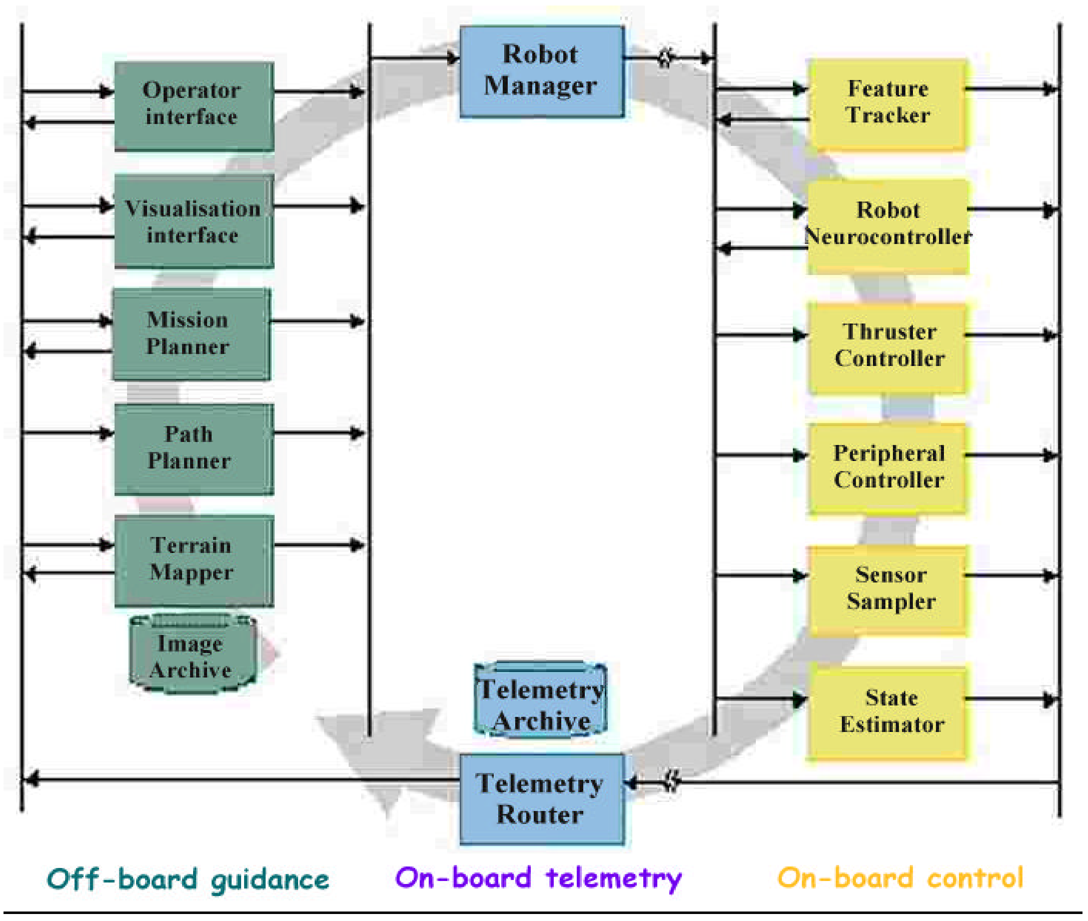


**Figure 3: Kambara Software Architecture**

The Robot Manager is the downstream communication module, directing commands to the modules running on-board. The Peripheral Controller drives all other devices on the robot, for example the cameras or scientific instruments. The Telemetry Router moves Robot State and acquired image and science data upstream to any number of public consumers and telemetry archives.

The Visualisation Interface transforms robot telemetry into a description of Robot State that can be rendered in a three-dimensional view. The Operator Interface interprets telemetry and presents a numerical expression of the Robot State. It provides methods for generating commands to the Robot Manager for direct teleoperation of robot motion and for supervisory control of the on-board modules. The Operator Interface can operate in conjunction with the Visualisation Interface to preview robot actions.

### 1.1.2 Project Goals

- The Kambara project has been designed so that the robot can be manipulated using a web browser. The existing interface is constructed as an application and therefore needs to be converted to an applet. While in-house development will necessitate the use of the application, the essential long-term objective for the operator interface is to be able to use it over the Internet.

- A feature of the operator interface is the use of a Wavefront 3D model for representing the current vehicle state (from the state information received). However, the model is simplistic, as it represents just the frame of the robot. Investigations need to be conducted with regard to using VRML models that will better represent the state of the robot (with thrusters and cameras).
- A feature of AUVs and an aim of the operator interface is direct teleoperation of robot motion. To this end, the development of a software joystick needs to be considered in order to facilitate the operator to manipulate the robot in any or a combination of its five degrees of freedom.
- One of the major shortcomings of the GUI layout at the end of 1999 was that it was not as flexible as desired. Automatic resizing was not a feature with most of the components requiring fixed dimensions to obtain the desired layout. Therefore, making improvements to the GUI involves a redesign of the layout. This also aids the goals stated above, especially the development of a software joystick for aiding direct teleoperation.
- Currently, the interface receives state information from a robot simulator. However, the overall objective of the Kambara project is for the robot to send state information to the interface while navigating areas such as the Great Barrier Reef. The robot must also accept commands to enable the operator to map a route for it. To those ends, the existing network design needs to be changed to allow:
  - Kambara's on-board system can receive commands from the operator interface and act on it.
  - The operator interface can receive the updated telemetry information from Kambara and display it.
- Enhancement of control modes is considered to ensure only one client can control the robot at a particular time.

## *1.2 Thesis Outline*

The goals of the project necessitate examining of the GUI followed by the software joystick and the network architecture.

Chapter 2 provides a background about the existing GUI layout and follows on to describe the redesign of the GUI layout. The conversion of the existing application to an applet follows the redesign with the investigation of the use of VRML 3D models concluding the chapter.

Chapter 3 details the process of the software development of a software joystick to be used by operators to manipulate the direction of the robot. The software joystick is to be developed as part of the improvements designated for the GUI layout.

In Chapter 4, the network architecture of the interface is discussed with particular emphasis on the changes required to conform to the long-term goal of the Operator Interface aspect of the Kambara project.

Chapters 2,3 and 4 discuss the analysis and design of the applet, the GUI layout, the software joystick and the network architecture. Chapter 5 discusses the implementation of the above, followed by Chapter 6, which describes the testing procedures developed and used during the project.

Conclusions from this project are drawn in Chapter 7 including a summary of this thesis, my contribution in terms of the work done and a discussion of the work that could continue using this project as a basis.

# CHAPTER 2: GUI ANALYSIS AND DESIGN

The existing operator interface is constructed as a Java application. Since the aim of the Kambara project is to enable the clients to view state information and send commands to the robot using a web browser, the current application should be converted to a Java applet that can subsequently be viewed on a web browser.

The 3D model is an important aspect of the interface. The current application makes use of a simplistic Wavefront model, which just depicts the frame of Kambara. While this is adequate for viewing the state of Kambara, it can be improved upon by using more complicated models.

As mentioned in Section 1.1.2, the GUI needs to be redesigned to better represent the state information of the robot. Improvements also need to be made with regard to navigation or teleoperation of the robot.

This chapter provides a background of the original GUI software architecture. It follows up with a description of the improvements made and the subsequent redesign of the GUI. A discussion is then presented on issues relating to the conversion of the existing application to an applet. The chapter concludes by outlining the investigation of the usage of a VRML 3Dmodel in place of the current model.

The chapter is organised as follows:

- Section 2.1 provides a background of the existing software architecture of the GUI. This is important because provides a basis for the remaining sections of the chapter.
- Section 2.2 describes the improvements made to the GUI layout including changes to the software architecture.
- Section 2.3 describes the reasons for converting the application to an applet and the design including the software architecture.
- Section 2.4 outlines the investigation conducted to replace the Wavefront 3D Kambara model with a VRML model.
- Section 2.5 provides a summary of the chapter detailing the major results of this chapter.

## *2.1 Background*

The Java programming language used for the software development of the operator interface [1]. Java was chosen because:

- It is portable and has cross-platform capabilities.
- It has convenient APIs for networking and 3D programming.
- It is easy to use for GUI development.

The interface has predominantly been developed using Java2 and Java3D API1.1. Java2 contains Swing libraries, which are used for GUI development in this project.

The software architecture of the original GUI is shown in Figure 4. *MainKambaraClient* is the main module and just an object in which three subcomponents; *KambaraClient*, *KambaraMainFrame* and *OutputKambaraData* are created. *KambaraData* represents the state information being received from the robot.

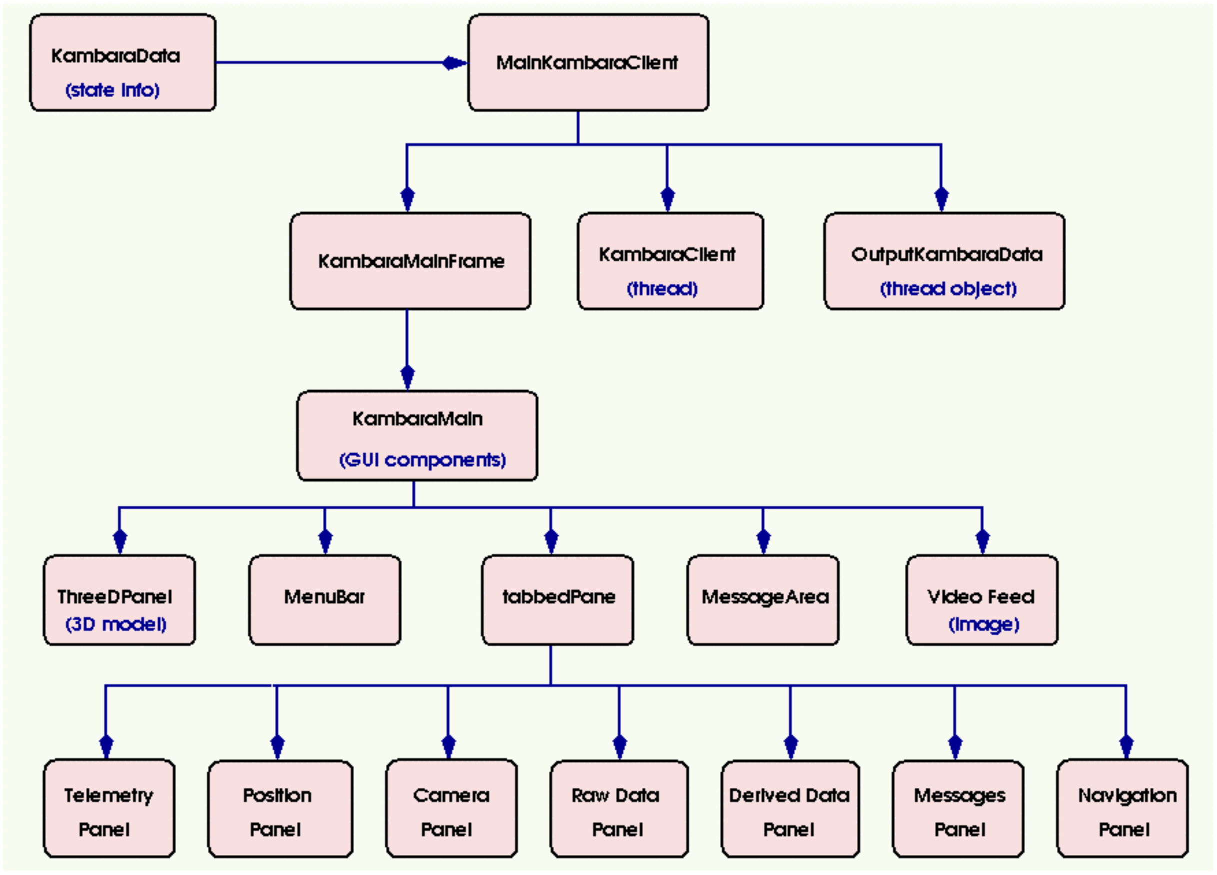


**Figure 4: Original GUI Software Architecture**

KambaraClient deals primarily with the network details of the client. It is responsible for establishing a connection between the client and a Java web server via Remote Method Invocation (RMI)[2] [1], and for setting up communication between the client and the server. It also has a thread object that continually receives state information from the server.

OutputKambaraData is a thread object that runs when a network connection is established. This is responsible for continually updating the GUI components with any new state information received. It is started when KambaraClient receives state information packets from the server.

KambaraMainFrame holds the panel containing all the GUI components; *KambaraMain*. KambaraMain is a panel that has been divided into a tabbedPane, a 3D model, a video feed placeholder, a menubar and a message area. Within the tabbedPane, information was divided into separate panels, based on the type of information being displayed. These include a

[2] RMI and the networking aspects of the Operator Interface are discussed further in Chapter 4

TelemetryPanel, CameraPanel, PositionPanel, RawData Panel, DerivedData Panel, MessagesPanel and NavigationPanel.

The TelemetryPanel provides the user with the general information about the vehicle. The PositionPanel provides a two-dimensional view of the position of Kambara and the relative position of the target.

The CameraPanel shows the current pan, tilt and zoom values of the camera on-board Kambara. It is also supposed to be used in case the operator wants to change the pan, tilt or zoom of the camera. This aspect is further discussed in Chapter 4 (Section 4.2.1).

The RawData and DerivedData panels show the sensor data and data computed on-board Kambara, respectively. These panels along with the TelemetryPanel provide all the numerical data to the client.

The MessagesPanel lists all the warning and error messages that are transmitted to the client while the NavigationPanel is used for manipulating the robot in any or a combination of its five degrees of freedom.

The 3D model and the video feed placeholder are in constant view of the client. The video feed had not been implemented due to the Kambara software development not having reached the desired stage. The 3D model is a Wavefront model and depicts the current rotational and translation state of the AUV.

After providing the background for the existing software architecture of the GUI, it is now worthwhile to discuss the improvements that were made, resulting in the redesign of the GUI.

## *2.2 GUI Layout*

The interface depicts most of the state information received from the robot. However, improvements need to be made for viewing the 3D model, the use of the CameraPanel that can manipulate the pan, tilt and zoom of the camera on-board Kambara and also in the NavigationPanel, which is used for manipulating the robot itself.

### 2.2.1 Design Principles

The interface had been designed based on the design principles of space, proximity, alignment and contrast [8]. These are related to the way in which users perceive information. The same principles were used while redesigning the layout of the interface.

- **Space**

  Space is the most effective element that can be used to provide support for your audience in their cognitive processing of visual displays. Spatial relationships are perceived precognitively - that is, without conscious effort. They do not have to be decoded and interpreted, as do colour cues, typographical cues, and so on.

- **Proximity**

  Elements that are close together in a visual display will be assumed to be related. Elements that are far apart will not be seen as related to each other. When elements are not clearly differentiated by proximity, the user has to group them consciously by focusing on them, taking in their meaning, and deciding which ones go together.

- **Alignment**

  Humans perceive items that are aligned vertically and/or horizontally to be more organised than those that are not, and people process, learn and remember organised information better than unorganised information.

- **Contrast**

  Use contrast to make elements more or less dominant in the display, influencing the order in which they are processed and their perceived importance or urgency. The primary forms of contrast are size, colour and shape.

### 2.2.2 Software Architecture

The architecture was changed to implement and incorporate the improvements designated for the GUI layout. The changes are highlighted in Figure 5, which illustrates the current software architecture of the GUI.

A new subcomponent, *ThreeDFrame* is created in MainKambaraClient. This component is a frame that holds the 3D Wavefront model (loaded into the ThreeDPanel) and is separate to the main interface. In the original implementation, the 3D model was viewed in KambaraMainFrame. While the panel incorporates the model, ThreeDPanel is still created in KambaraMain; it is just not shown in KambaraMainFrame.

Also, the *CameraPanel* component is no longer a part of the tabbedPane. It now occupies the position of the ThreeDPanel and the video feed placeholder. It is created as a subcomponent of KambaraMain and not that of the tabbedPane.

The DataPanel is now a new subcomponent of the tabbedPane. It incorporates the information contained in the *RawData* panel and *DerivedData* panel into one panel. While the Raw and Derived Data panels have not been deleted, they have just been made into subcomponents of the DataPanel. Another improvement was to completely redesign the *Navigation* panel to facilitate an easier and more intuitive manipulation of the movements of the robot.

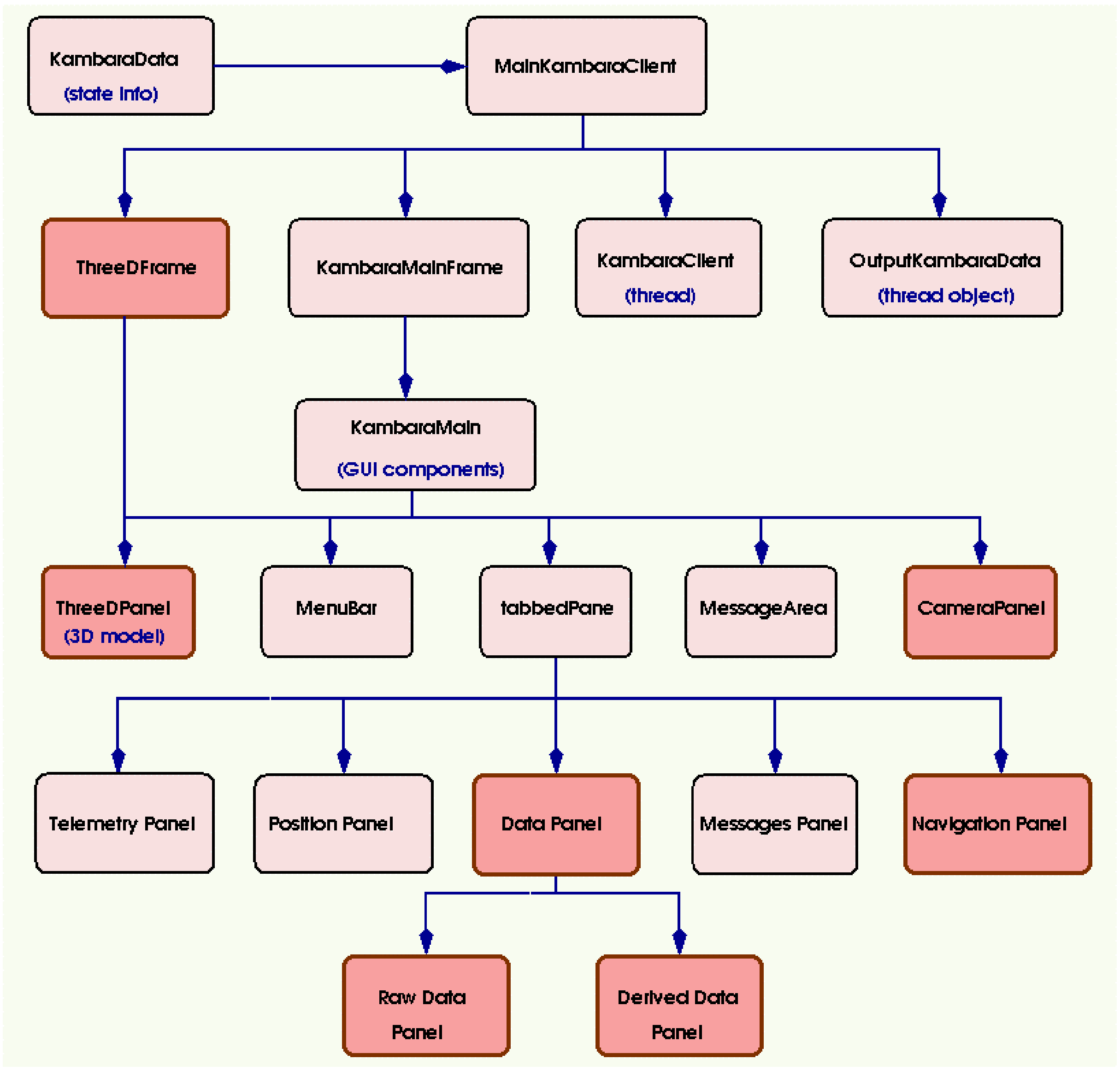


**Figure 5: Current GUI Software Architecture**

The rest of the architecture is inherited from the original software design. While the above was a brief overview of the improvements made, it is now best to understand all the issues associated with the decisions that led to these changes.

- **Kambara 3D Model**

  In the original architecture, the 3D model is viewed in a panel in KambaraMainFrame. This panel is small and hard to navigate. This inadequacy caused it to be rendered virtually useless as the user could not obtain or gather sufficient information from it. That is, while the panel could show the state changes when the interface received information from the robot (or the simulation), it was not very user-friendly in terms of manipulating the model. This aspect becomes more evident when the model traverses the boundaries of the view and the panel needs to be navigated.

An option to rectify this problem was to incorporate the ThreeDPanel module into the tabbedPane. But one of the essential requirements of the panel was for it to be in view of the user at all times.

Therefore, it was decided that the model should have its own separate frame, ThreeDFrame, which is shown in Figure 6. The panel is still one of the subcomponents of KambaraMain. This makes it easier to update the 3D model when the interface receives state information.

The creation of ThreeDFrame also presented us with more options, especially when developing the software joystick. Apart from viewing the vehicle state obtained from the robot, it can be used to simulate the orientation of the robot for a move command. These details as well as other aspects of the development of the software joystick are further discussed in Chapter 3.

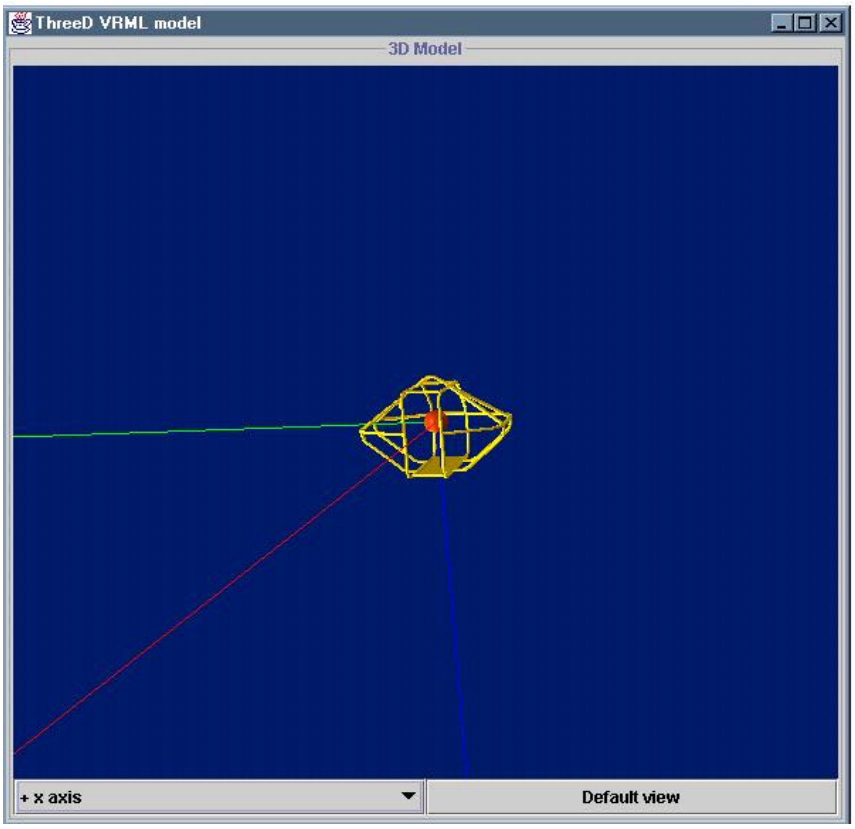


**Figure 6: ThreeDFrame**

- **Camera Panel**

  The Kambara model and the video feed placeholder were supposed to be constantly visible to the user. It was planned that the digitiser would be used to continually refresh an image, which would be presented in the GUI. Unfortunately, the AUV development has not reached the stage where a video stream could be obtained and this feature could be tested.

It is also useful to have the camera in constant view as this gives the operator an opportunity to know the state of the camera at all times and therefore adequately know which direction the camera is pointing to.

The panel also has utilities for sending camera commands to the robot that would help map a route for Kambara. This will help the operator change the pan, tilt or zoom anytime, based on the state of the robot. This also provides an opportunity to plan ahead, avoid obstacles and map a route for the robot.

Therefore, the video feed placeholder was removed and replaced with a visualisation of the pan, tilt and zoom values of the camera (see Figure 7).

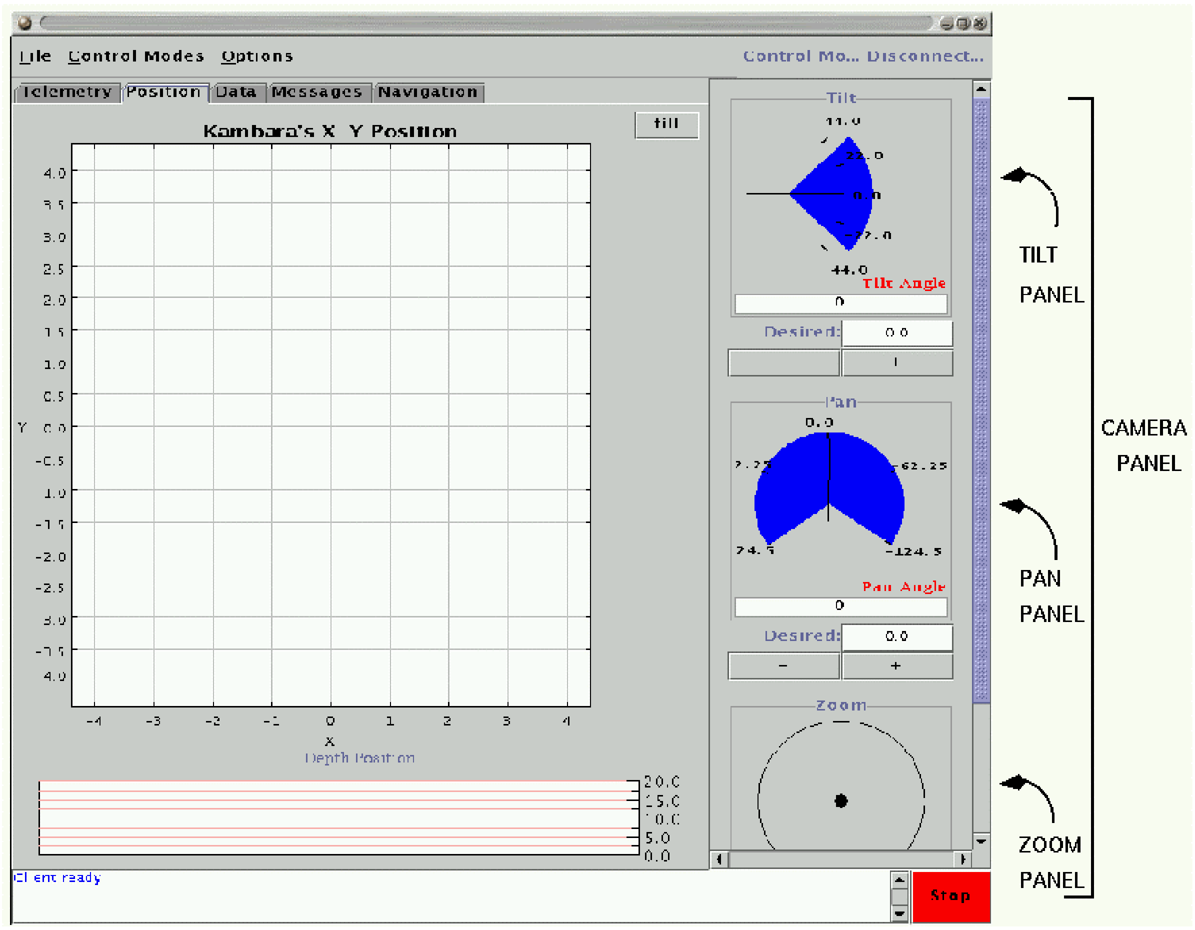


**Figure 7: Camera Panel**

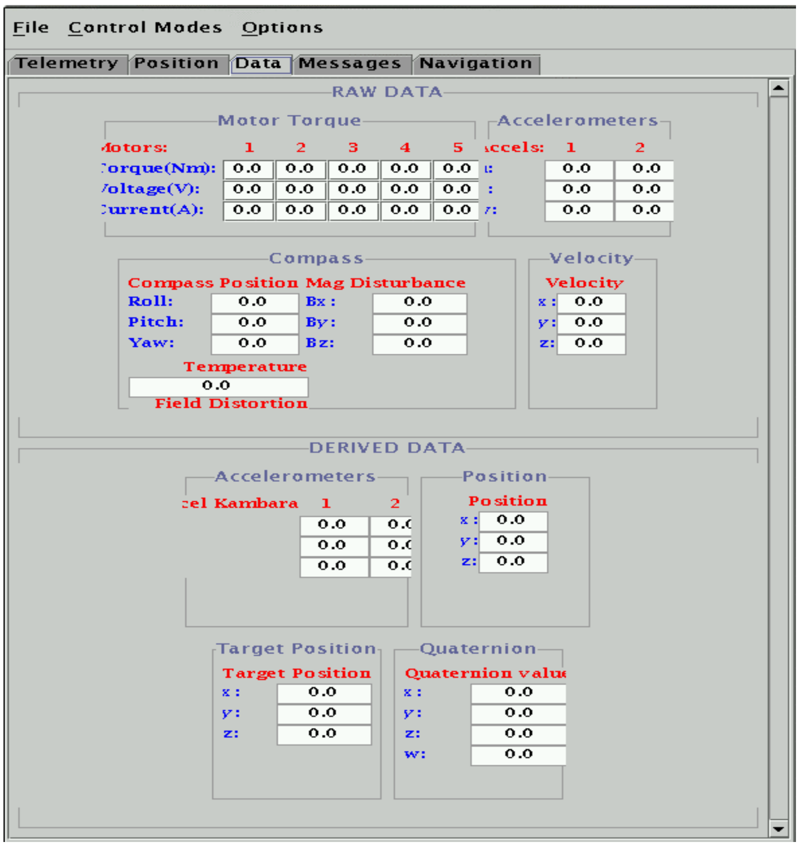
- **Data Panel**

  As mentioned before, the RawData panel contains data that is obtained directly from the sensors. On the other hand, the DerivedData panel contains information that is computed on-board Kambara. The Data panel incorporates both these forms of data onto one panel as shown in Figure 8. This gives users an opportunity to compare similar sets of values and thereby ascertain the effect of factors such as gravity.

  The raw data depicted in the panel includes:

  1. Motor torque, current and voltage values for each of the five thrusters.

2. Compass values including the roll, pitch and yaw, magnetic disturbances, temperature and field distortion.
3. Velocity and accelerometer values.



| Motors: | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Torque(Nm): | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Voltage(V): | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Current(A): | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

| Accels: | 1 | 2 |
|---|---|---|
| : | 0.0 | 0.0 |
| : | 0.0 | 0.0 |
| : | 0.0 | 0.0 |

| Compass Position | | Mag Disturbance | |
|---|---|---|---|
| Roll: | 0.0 | Bx : | 0.0 |
| Pitch: | 0.0 | By : | 0.0 |
| Yaw: | 0.0 | Bz: | 0.0 |

Temperature
0.0
Field Distortion

| Velocity | |
|---|---|
| x : | 0.0 |
| y : | 0.0 |
| z: | 0.0 |

| cel Kambara | 1 | 2 |
|---|---|---|
| | 0.0 | 0.0 |
| | 0.0 | 0.0 |
| | 0.0 | 0.0 |

| Position | |
|---|---|
| x : | 0.0 |
| y : | 0.0 |
| z: | 0.0 |

| Target Position | |
|---|---|
| x : | 0.0 |
| y : | 0.0 |
| z: | 0.0 |

| Quaternion value | |
|---|---|
| x : | 0.0 |
| y : | 0.0 |
| z: | 0.0 |
| w: | 0.0 |

**Figure 8: Data Panel (combines Raw and Derived Data)**

The derived data is the data computed on-board Kambara. The data depicted includes:

1. Accelerometer values.
2. Current vehicle position and estimated target position.
3. Current quaternion value of the vehicle computed using the compass roll, pitch and yaw values (raw data).

This along with the Telemetry panel provides a general picture of the information being received from Kambara and makes it much easier to analyse.

- **Navigation Panel**

  The implementation for navigating the robot in its five degrees of freedom was minimalist and laborious for the operator. All those components were deleted and the need for improvements led to the investigation, design and development of a *software joystick*. The joystick allows the operator to use mouse clicks and / or keyboard events to change the direction of the robot.

With a picture of the current GUI software architecture in mind (for an application), the details of the conversion of the existing application to an applet can be explained.

## 2.3 Applet

To allow multiple clients to connect to the robot and view the state information, a web-based operator interface is required. Of these clients only one will be able to control Kambara.

Hence, the application is converted to an applet while allowing the interface to also be viewed as an application. The main reason for this was the ease of use associated with applications for in-house development, especially for testing purposes.

### 2.3.1 Software Architecture

The main factor associated with the design was to be able to start the interface from the applet. That is, the applet would serve as a starting point or as a link to the interface. The software architecture was developed and modified accordingly.

The software architecture of the applet is given in Figure 9. *KambaraMainApplet* is the main module that holds the applet and has four subcomponents; *KambaraMainFrame*, *KambaraClient*, *OutputKambaraData* and *ThreeDFrame.* Essentially, KambaraMainApplet takes the role of MainKambaraClient in the application.

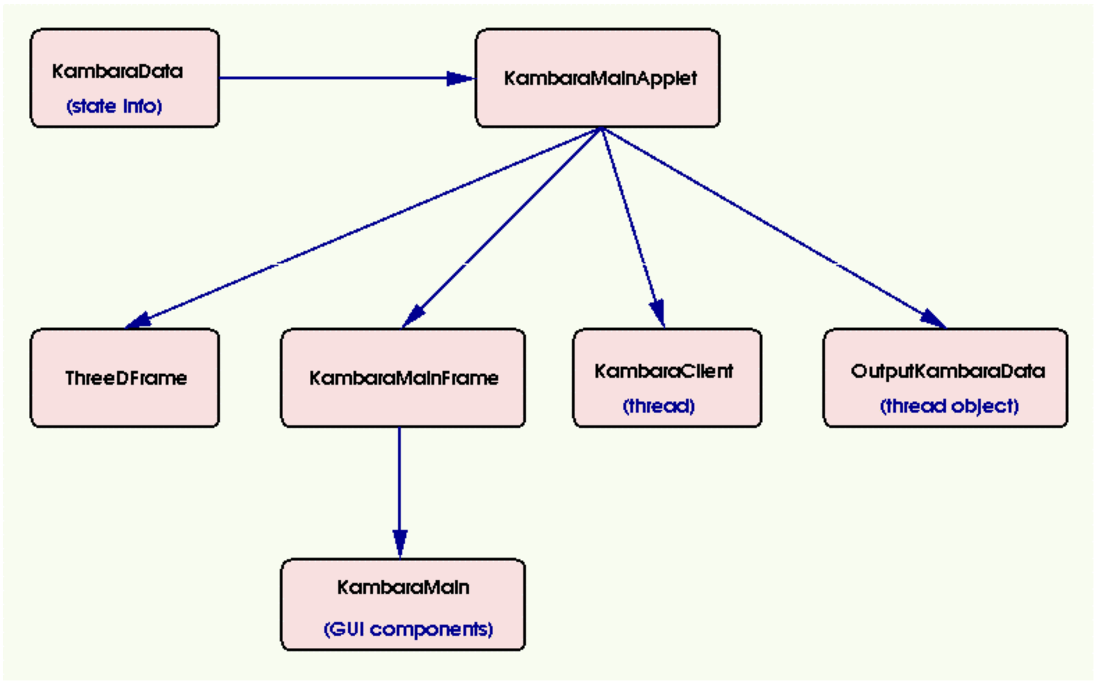


**Figure 9: Kambara Applet Software Architecture**

KambaraClient handles all the network details of the client while OutputKambaraData updates the state information on the interface. *KambaraMain* is the panel that holds all the GUI components and is a subcomponent of KambaraMainFrame[3].

[3] Section 2.1 provides more details about KambaraClient, KambaraMainFrame, KambaraMain and OutputKambaraData

The only difference between the architectures of the application (shown in Figure 5) and the applet (Figure 9) is that KambaraMainApplet assumes the role of MainKambaraClient. This facilitates the development of the interface as an application as well as an applet.

The next stage in the development of the GUI is the investigation of a VRML 3D model for the interface.

## *2.4 VRML Model of Kambara*

The Wavefront model used in the interface was developed from a VRML model. Due to conversion problems, the model is very simplistic and consisted of just the frame of the robot [see Figure 10].

The model is visualised using the Java3D API [10][14]. Pictures rendered with Java3D are called scenes (or "virtual universe") that are in turn broken into subcomponents, namely behaviour, model, object characteristics, 3D co-ordinate, math. These create a complex world of 3D objects and are a major cause in the decrease in the speed and performance of the interface.

It was felt that having a more complicated model including thrusters and cameras would greatly enhance the operator's understanding of the robots' behaviour. This would be best achieved by directly loading a VRML model and not converting it into any other format.

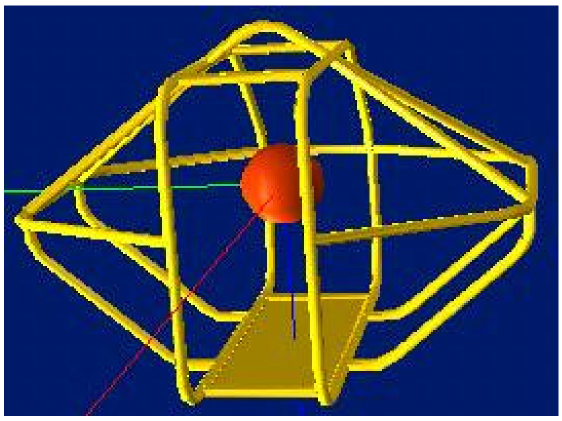


**Figure 10: Kambara Wavefront Model**

A VRML model depicting the cameras and the thrusters along with the frame of Kambara [see Figure 11] had been developed at the Systems Engineering Department in RSISE. But, one of the major considerations to be taken into account was again the performance and the speed with which the client would update its GUI. That is, use of the VRML model should not come at the expense of further reduction in performance.

It was found that the Java3D API in it current form, does not support the loading of VRML models into a scene. To this end, the Internet was scanned for any software packages (preferably developed in Java) that would help us achieve the objective. This led us to the NCSA Portfolio [12, Appendix A]. This utility is used in conjunction with the Java3D API to load models of different formats including Wavefront and VRML.

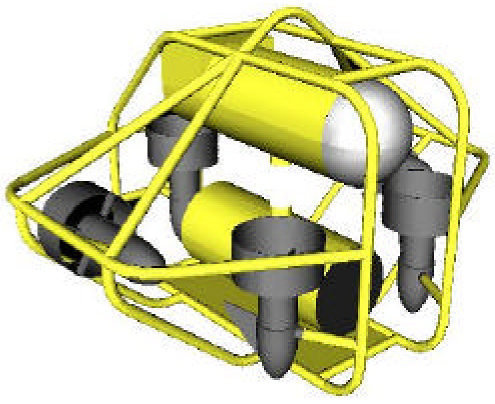


**Figure 11: VRML Model**

But, NCSA Portfolio could not be used to load VRML models due to problems encountered with the VRML model and the utility itself. These are discussed in the following section.

### 2.4.1 Loading Problems

The problems encountered with the VRML model and *NCSA Portfolio* were serious enough for me to stop the investigation and continue using the Wavefront model. They were:

- NCSA Portfolio supports the loading of only VRML97 models.
- The loader in NCSA Portfolio, used to load the models does not possess any parameters to properly position the model in the ThreeDFrame. We could also not use the Java3D API methods, as they do not support VRML models.
- The simplest VRML model available [Figure 11] was 1 Megabytes in size. Loading a file of such size led to a drastic reduction in the performance, which was unacceptable. As mentioned in the previous section, use of the model could not come at the expense of speed or performance.

Therefore, it was decided that the existing Wavefront model would continue to be used till technologies improved and facilitated easier and faster use of the VRML model.

## *2.5 Chapter Summary*

The redesign of the GUI included creation of a separate frame for just the 3D model, incorporation of the raw sensor data and the derived data into one panel and the separation of the CameraPanel from the tabbedPane that was made a part of the main interface. The applet architecture is very similar to the application thereby facilitating the execution of the interface as an application as well as an applet. Currently, he VRML 3D model cannot be used in place of the Wavefront model due to constraints imposed by Java3D and NCSA Portfolio.

It was mentioned earlier in the chapter that the NavigationPanel was completely redesigned with all the existing components deleted and a software joystick developed in place. The next chapter discusses the software joystick in detail including the analysis and the design.

# CHAPTER 3: SOFTWARE JOYSTICK

Chapter 2 was concerned with the redesign of the operator interface GUI. This included the redesign of the NavigationPanel. This chapter details the reasons for the changes in the NavigationPanel and the development of a software joystick to be incorporated as part of the navigation mechanism of the interface. The software joystick is to be used by the operator to be able to manipulate the robot in any or a combination of its five degrees of freedom.

However, as the mechanisms for actually moving the robot have not yet been developed, this chapter will concentrate on the requirements and the associated design involving the joystick. It will first provide a background about why the interface needs a software joystick followed by detailed explanations about what functions the joystick needs to perform and the design used to implement those functions.

This chapter is organised as follows:

- Section 3.1 describes the control modes that were set in place for the operator interface in 1999 [1]. These help determine why the software joystick needs to be developed.
- Section 3.2 outlines the requirements of the joystick including the constraints.
- Section 3.3 discusses the design including a detailed explanation of the software architecture of the joystick.
- Section 3.4 explains the major outcomes of this chapter as a summary.

## *3.1 Control Modes*

Four control modes for the operator interface have been defined. These would facilitate multiple users to view the state information being sent from the robot, even if only one of them is in control of it. The four control modes are Observer, Individual control, Teleoperation and Supervisory control.

### 3.1.1 Observer Mode

Users that do not control the robot are in Observer mode. It enables them to watch the activities of the operator and view all state information sent by Kambara. This is the state in which users should be in, if they have not successfully gained control of the robot.

### 3.1.2 Individual Control Mode

This mode is used when it is necessary to fire individual thrusters. This mode would come into use when say; Kambara is in an awkward position next to a rock. It would also be useful if the operator wants to manipulate the camera (pan, tilt and zoom).

### 3.1.3 Teleoperation Mode

This mode is used to enable the operator to move Kambara in any or a combination of its five degrees of freedom. All of the thrusters will be co-ordinated to carry the specified motion out.

### 3.1.4 Supervisory Control Mode

This control mode is similar to that used on the rover that landed on Mars in 1998 [13]. It essentially allows the operator to plan a route.

The joystick is meant to be a stepping-stone in implementing the Teleoperation control mode and as a major improvement to the previous navigation utilities provided in the interface.

## *3.2 Requirements and Analysis*

The implementation for the teleoperation mode in the existing application was basic and non-intuitive. The user had to laboriously select one of ten directions and separately manipulating the roll, pitch and yaw as shown in Figure 12.

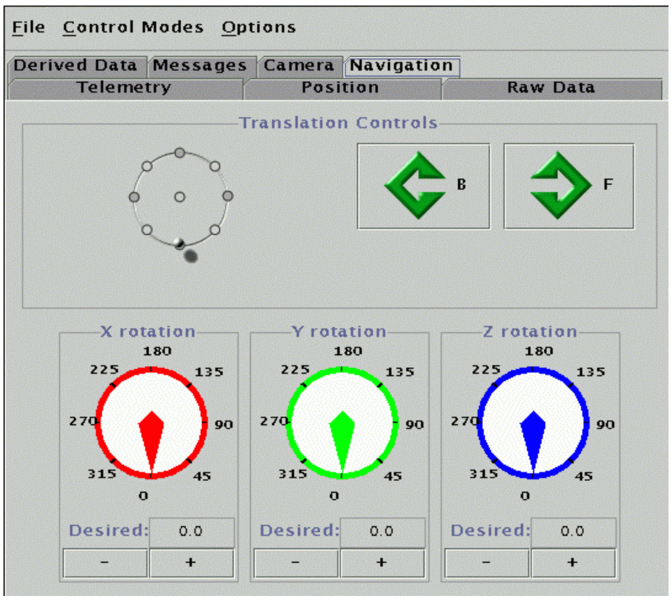


**Figure 12: Robot Navigation Controls (1999)**

To make navigating the robot easier, the concept of a software joystick was investigated. The basic idea is to allow:

- Mouse clicks and / or keyboard events to determine the surge, heave, roll, pitch and yaw in a non-laborious manner.
- Not slow down the interface any more than its current state.
- Be able to type specific values of the above-specified degrees of freedom and change them.
- Be able to abort the prospective change if the operator is unsatisfied with the changes.

- Have a view of the changes in the orientation of Kambara. This would be done using the existing 3D model.

## 3.3 Design

### 3.3.1 Considerations

It is human nature to get frustrated when updates of information are slow and the interface is not easy to use. The 3D model of Kambara uses the Java3D API that results in a substantial decrease of speed and decrease in the performance. It was necessary that the joystick did not slow the interface any further. Having another 3D view would necessitate a similar use of Java3D, which would make the interface unusable due to its poor performance.

As mentioned before, the ease of use for the operator is another major consideration. The joystick will not be useful if the operator is unable to grasp its intricacies in a short span of time. It was for these above-mentioned reasons that two two-dimensional views were used to manipulate the robot. They help keep the different components of the joystick, simple and easy to use. Having taken into account the factors used in determining the design, the software architecture of the joystick can now be described.

### 3.3.2 Software Architecture

The main module that contains the joystick is *NavigationPanel*, which in turn is a subcomponent of the tabbedPane[4]. It is just a panel that holds 3 subcomponents; *TopPanel*, *SidePanel* and *MovementsPanel*. The architecture is shown in Figure 13.

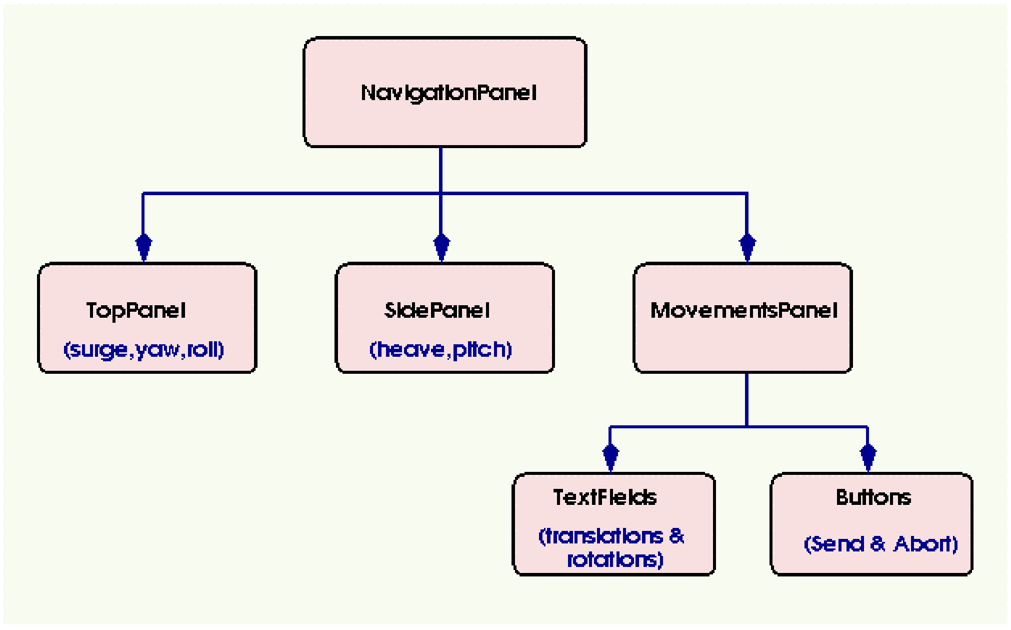


**Figure 13: NavigationPanel Software Architecture**

[4] See Chapter 2, Section 2.2 for more details about the current software architecture of the GUI

- **TopPanel**

The TopPanel lets the operator manipulate three of the five possible degrees of freedom. This is accomplished through the use of mouse clicks. The panel provides the operator with a perspective of the top view of the robot, which is shown in Figure 14. The changes in some selective movements can be generated based on this view.

The view allows the operator to set the surge, yaw and roll of the robot. That is, the translation in the X direction (surge) and the rotations in the X and Z directions (roll and yaw respectively) are set using this panel. These can be set intuitively through mouse clicks and / or a combination of the keyboard and mouse.

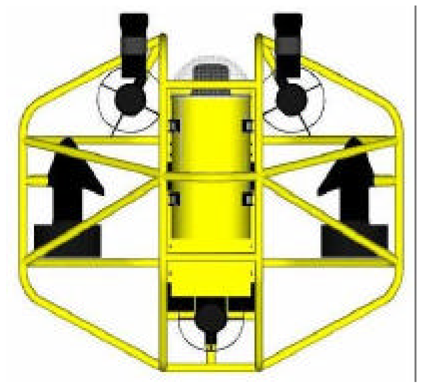
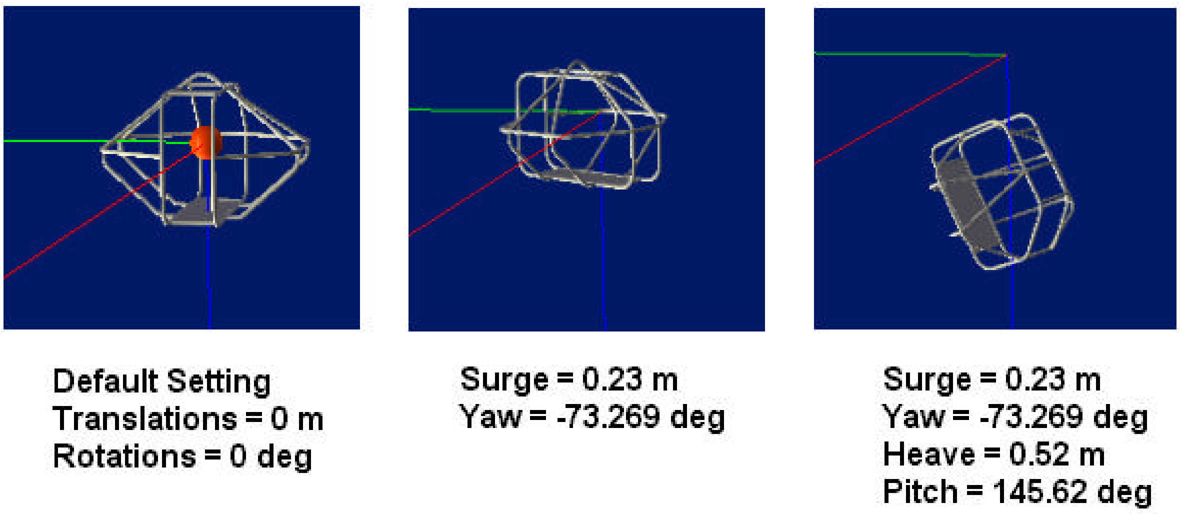


**Figure 14: Top View of Kambara**

The 3D model is used concurrently with the panel to provide a simulation of the change in the orientation of the robot. This change will occur on the completion of the desired surge, yaw or roll. This gives the operator a better idea of the robot's movements. It also acts as a feedback so that the operators can correct the values and be able to map a route for the robot. An example of the use of the 3D model to view the change in the orientation through the use of the TopPanel is given below in Figure 15.



Default Setting
Translations = 0 m
Rotations = 0 deg

Surge = 0.23 m
Yaw = -73.269 deg

Surge = 0.23 m
Yaw = -73.269 deg
Heave = 0.52 m
Pitch = 145.62 deg

**Figure 15: Simulation of the movements of the joystick (in the 3D model)**

- **SidePanel**

  The Side view of the robot is depicted in this panel. This is illustrated in Figure 16. The pitch and heave movements[5] are set using this component of the joystick. These are changed or set through the use of mouse clicks.

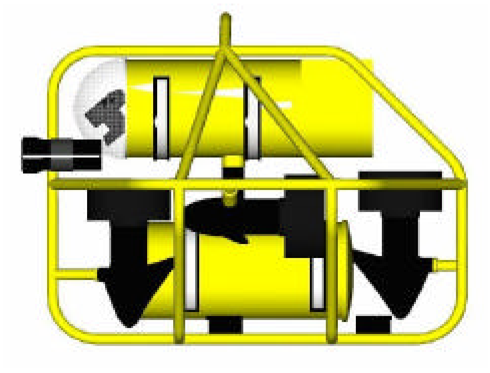


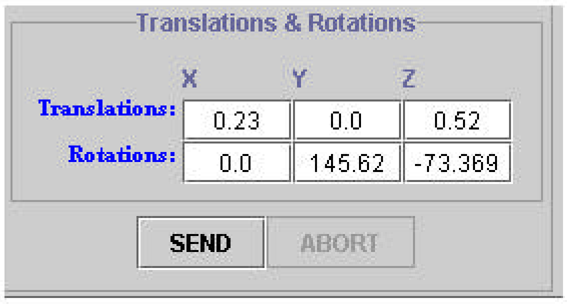
**Figure 16: Side View of Kambara**

  The usage of the 3D model is the same as in the TopPanel with the types (directions) of movement being different.

- **MovementsPanel**

  Another method of changing any degree of freedom is through the MovementsPanel. There are textfields that accept key entries and implement the specified changes in the orientation. The changes are the same, as one would expect from the TopPanel or the SidePanel.

  But the textfields are also used to show the latest changes implemented through the use of the TopPanel or the SidePanel [see Figure 17].



Translations & Rotations

| | X | Y | Z |
|---|---|---|---|
| Translations: | 0.23 | 0.0 | 0.52 |
| Rotations: | 0.0 | 145.62 | -73.369 |

**Figure 17: Current Translations and Rotations**

  The Send the Abort buttons shown in Figure 17 are present to facilitate the sending and aborting of move commands to the robot, taking the required changes as parameters.

[5] Pitch is the rotation in the Y direction while heave is the translation in the Z direction

## *3.4 Chapter Summary*

A software joystick has been analysed and designed for facilitating direct teleoperation of the robot. It consists of two two-dimensional views (top view and side view) and manipulates the designated five degrees of freedom through keyboard and / or mouse inputs. There is a panel for manual numerical changes as well (MovementsPanel).

The 3D model is used to simulate the prospective changes in the translations and the rotations. Due to the overall Kambara software development not having reached a stage where the robot accepts move commands, the joystick does not send or abort any commands.

In the event that the joystick is used to send the commands, we need to further our understanding of the networking aspects of the operator interface. The next chapter discusses these aspects and provides explanations about how the clients can send commands to the robot.

# CHAPTER 4: NETWORK ANALYSIS AND DESIGN

The previous chapter discussed one of the first stages of teleoperation, that is, the development of a software joystick. The next stage essentially involves the transmission of commands to it and for Kambara to act on those commands. This involves a detailed discussion of the network architecture of the operator interface.

This chapter starts off by describing the architecture that was in place at the end of 1999. The chapter goes on to discuss the requirements and analysis for sending commands to the robot and receiving Telemetry data from it. It then concludes with a discussion of the long-term goal for the interface network design and the conformance to the long-term goal.

The rest of the chapter is organised as follows:

- Section 4.1 gives a background of the Network Architecture including the architecture that was already in place at the beginning of the project and the functions of the server and the client side.
- Section 4.2 outlines the requirements and analyses the transmission of camera commands to the robot and reception of telemetry router data from the robot.
- Section 4.3 describes the long-term goal for the interface network design and the changes used to make the current architecture conform to it.
- Section 4.4 gives a summary of the main outcomes of the network analysis and design.

## *4.1 Background*

The system depicted in Figure 18 was the implementation at the beginning of the project. There are multiple clients (applications) connecting to a Web server that is written in Java.

### 4.1.1 Server

The essential role of the server is to receive state information from the robot and distribute it to the clients and issue commands to the robot from 'one' of its clients. Clients register with the server via an RMI connection and thereby request state information. All the clients registered with the server then receive UDP[6] packets containing the latest state information.

RMI allows an application to call methods and access variables inside another application, which may be running in a different Java environment or a different system altogether, and to pass objects back and forth over a network connection. The primary goal of RMI was to make interacting with a remote object as easy as interacting with a local one.

In addition, however, RMI includes more sophisticated mechanisms for calling methods on remote objects to pass objects or parts of objects either by reference or by value, as well as

[6] See Appendix B for more details about User Datagram Protocol (UDP)

additional exceptions for handling network errors that may occur while a remote operation is occurring.

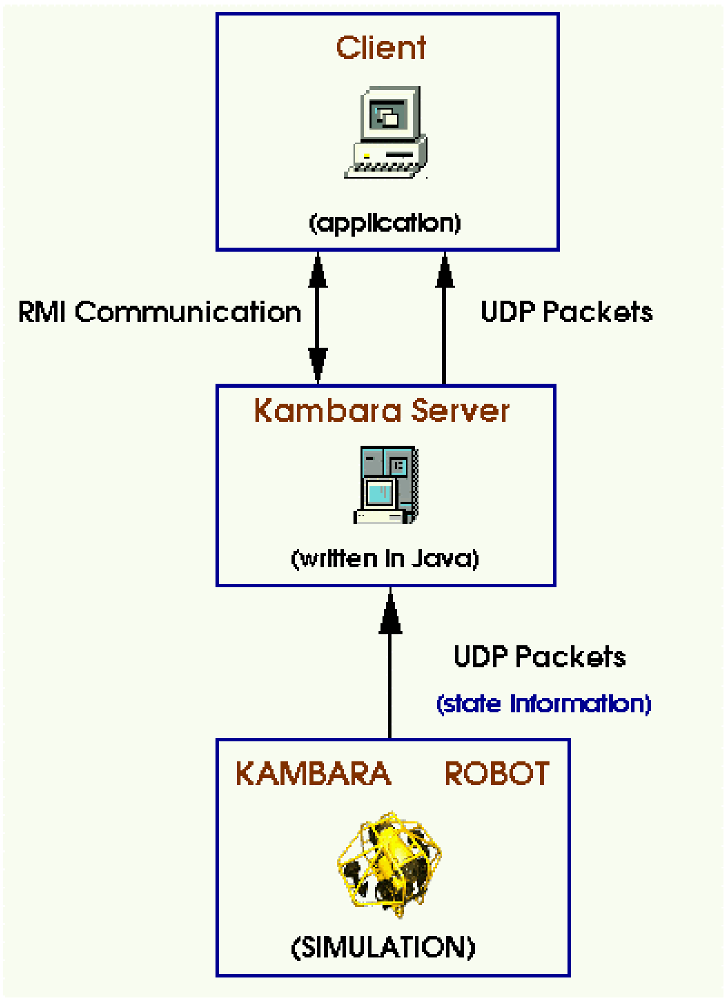


**Figure 18: Original Network Design**

The server is also responsible for ensuring that only one client at a time has the ability to control the vehicle, via the *ControlState* object stored on the server. Therefore the client can issue commands at any time even while the server is receiving and distributing the state information packets.

### 4.1.2 Client

The basic role of the client is to register with the server and receive the latest state information of the robot, from it. The client is currently an application that can be executed on any machine supporting the Java runtime environment. In the event that the client is also in control of the robot the ability to issue commands such as changing the pan, tilt or zoom of the camera on-board Kambara and moving the robot in any or a combination of its five degrees of freedom, is provided.

To receive state information from the server, the client uses the RMI connection to access the server's method *addStateInfoListener*, which stores the client's address and port number. The information received by the client is in the form of UDP packets, which are unreliable and can get lost. But this disadvantage is compensated by the fact that the Kambara system is continually sending information. Also, UDP is very fast and does not require a confirmation of each packet.

As mentioned in the previous subsection, the methods for issuing commands are defined in the server's object *ControlState.* But, while this ability is provided to the client, the functionality for sending commands has not been provided. This needs to be addressed because ultimately, the manipulation of the robot is going to be achieved only via the operator interface.

Another issue to be addressed is that the client receives state information from a simulator and not the actual robot. For the operator interface to be operational, this aspect needs to be corrected as the aim of the interface is to present data that is either produced on-board Kambara or from its components.

Having described the roles of the server and the client, it is now useful to analyse the changes that need to be made in order for the operator interface to properly interact with the robot.

## *4.2 Requirements and Analysis*

### 4.2.1 Camera Commands

While the joystick commands can be simulated using the 3D model as discussed in Chapter 3 (Section 3.3), the overall aim is to be able to communicate with the system on-board the robot and send the joystick commands to it.

The definition and sending of camera commands is the first part in that process. Based on the software development on-board Kambara, a vector of camera command data was defined. This is listed in Table 1 where the pan, tilt and zoom values are absolute values.

| Range (Bytes) | Data Type | Data Represented |
|---|---|---|
| 0-7 | Long | Initialise (boolean; always = 1) |
| 8-15 | Double | Pan Value |
| 16-23 | Long | Absolute Pan (boolean) |
| 24-31 | Double | Tilt Value |
| 32-39 | Long | Absolute Tilt (boolean) |
| 40-47 | Double | Zoom Value |
| 48-55 | Long | Absolute Zoom (boolean) |

**Table 1: Vector of Camera Command Data**

The telemetry data received by the client from the robot (via the server) contains the updated pan, tilt and zoom values of the camera. Therefore, the requirement is to send the commands quickly through UDP packets, with the client using the updated values to determine if the command was received properly or lost. Considering that the client receives simulator data, a means to get data from the Telemetry Router on-board the AUV needs to be analysed.

### 4.2.2 Telemetry Router

If the clients are to receive proper telemetry data and actually determine the status of Kambara, the server needs to communicate with the Telemetry Router in real time instead of a simulator [Appendix C]. A vector of the data received from the Telemetry Router is given in Table 2.

Some of the variables such as computer status and target velocity are not represented in GUI. These have no bearing currently, with their values constantly set to zero. It was determined that corresponding changes to the GUI would be made when the Kambara software development reached a stage where it defined such variables.

Also, considering that Kambara will continuously send telemetry data, speed will be a bigger consideration than reliability. Therefore, it was decided to continue using the UDP protocol (in a similar manner to receiving simulator data)

| Range of data in doubles | Data Represented |
|---|---|
| 0-2 | Robot position x, y, z values |
| 3-6 | Robot position – Quaternion |
| 7-9 | Kambara velocity u, v, w values |
| 10-12 | Kambara angular velocity p, q, r values |
| 14-16 | Target position x, y, z values |
| 17-20 | Target position – Quaternion |
| 21-23 | Target velocity u, v, w values |
| 24-26 | Target angular velocity p, q, r values |
| 28-32 | Motor torque (5 thrusters) |
| 34-38 | Motor voltages |
| 39-43 | Motor currents |
| 45 | Camera Pan value |
| 46 | Camera Tilt value |
| 47 | Camera Zoom value |
| 48 | Computer Status (64 bits of unsigned chars) |

**Table 2: Vector of Telemetry Router Data received from Kambara**

The essential goal of implementation of the camera commands and telemetry router data is for the Network Design to conform to the long-term goal that has already been defined. The design issues relating to the implementation of sending camera commands and receiving telemetry data are discussed next.

## 4.3 Design

### 4.3.1 Long Term Goal for Network Design

The long-term goal for the Interface Network Design [1] is depicted in Figure 19. As is evident from the figure, the design shown in Figure 18 bears a few differences to it. The essential idea was to interface the client with the server and have that in turn communicate with the AUV.

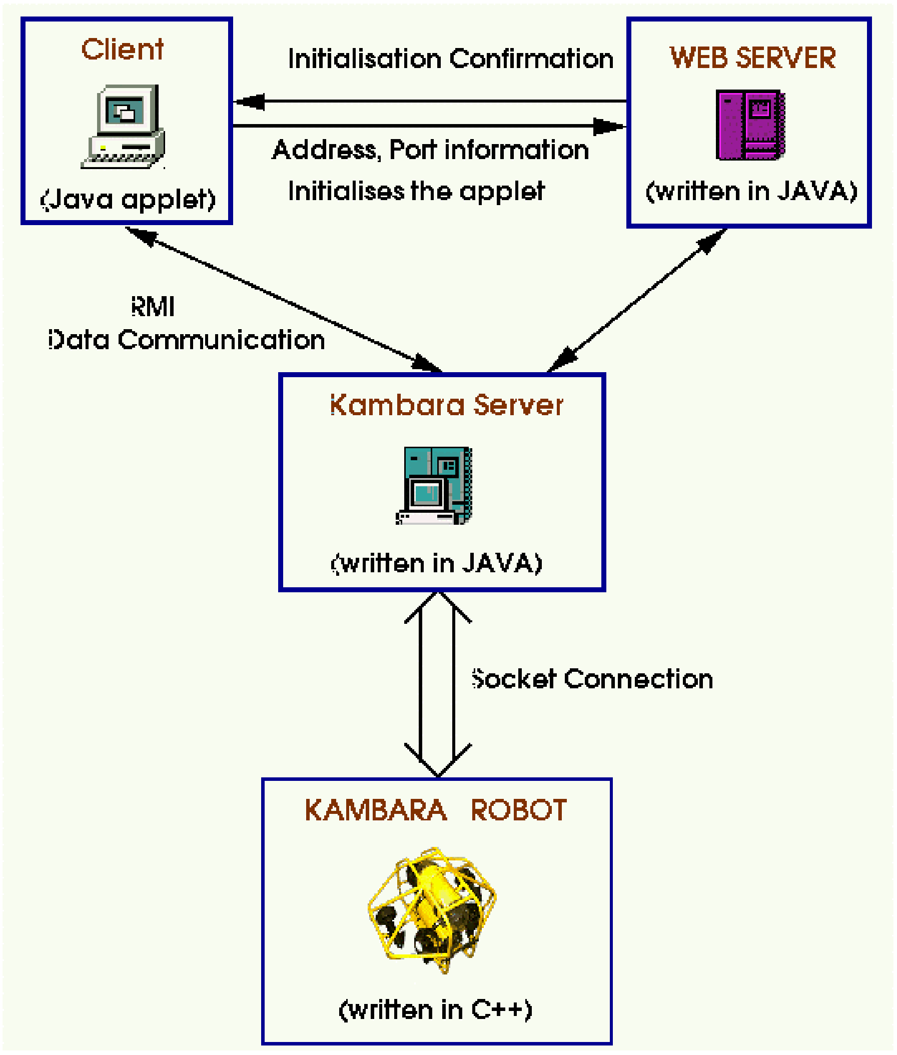


**Figure 19: Long-Term Goal for Network Design**

The Web server in the figure is used for initialisation of the applet. From that point on though, all client-server communication will be between the client applet and the Java server via RMI. A simple socket connection will be used between the Java server and the computer on-board the AUV, since RMI is only used between two Java objects.

### 4.3.2 Conformance to Lo ng-term Goal

The major differences between the original design and the long-term goal (Figures 18 and 19) are given below. The long-term view for the interface network design included the execution of the client as an applet and proper communication with the robot. This would be in the form of commands from the client to the robot and state data from the robot to the client. The conformance criteria are

1. The clients are constructed as applets as opposed to applications.
2. The server gets state information from the Telemetry Router on-board Kambara as opposed to a simulator.
3. Mechanisms need to be constructed for a client to send commands to the robot. While there are methods defined for sending commands, these have not been implemented.

As mentioned in Section 2.3, the client application has been converted to be used as an applet as well. This achieves the first of the three goals stated above. The network architecture had to be changed so as to allow proper communication between the client and the robot. The current network design is shown in Figure 20. Currently, the software used to run the camera on-board Kambara does not accept socket connections. It only listens for UDP packets.

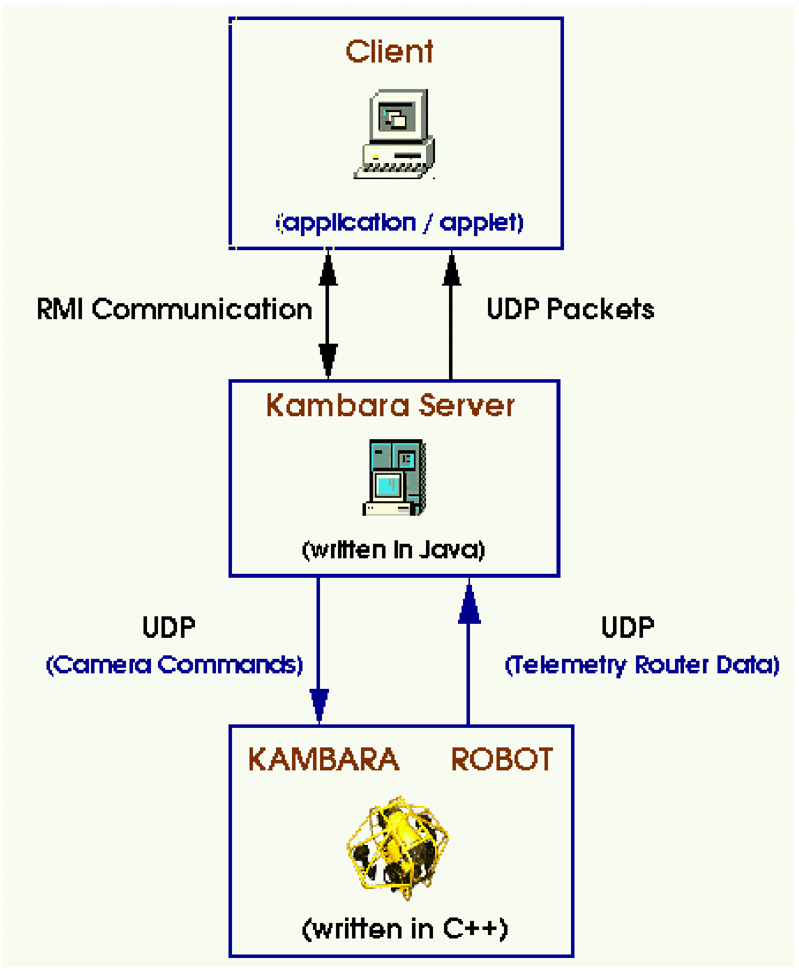


**Figure 20: Current Interface Network Design**

The *KambaraServer* takes the robot machine address and the camera port number as parameters and uses them to create UDP socket connections to the robot. The client uses the ControlState object on the server to send camera commands and the server listens for telemetry router data, which is then passed on to all the clients. The camera values are contained in the telemetry data (Table 2) and therefore it can be determined if the camera command was transmitted and implemented.

A typical flow of commands throughout this system is as follows.

1. A client sends a registration command through a RMI connection, indicating that they wish to receive state information
2. The server registers the client taking the address and port number and puts that information onto a vector of clients.
3. Each telemetry router data packet received by the server is sent to all the clients. The packets are continuously received by the server and are sent to the clients without any decoding.
4. The client receives the telemetry data, decodes the UDP packets and updates the GUI based on the data.
5. The client then requests control of the vehicle.
6. The server sends an error message back to the client if another client is already in control of the AUV. In the event that no one is controlling the vehicle, the client is given control and GUI is updated to reflect the adequate mode.
7. On obtaining control of the vehicle, the client can send commands to manipulate the camera on-board Kambara. The state information packets reflect the changes, which are updated on the GUI.
8. The client can relinquish control of the vehicle and go back to the *Observer* mode at any time.

## 4.4 Chapter Summary

The original network design had shortcomings with respect to sending proper commands to the robot and receiving actual telemetry data from the robot. The long-term goal for the interface network design was defined in 1999 [1]. The Network design was changed to improve the deficiencies in the original design and conform to the long-term goal.

The previous chapters (Chapters 2 and 3) along with this chapter, discussed the analysis and design aspects of the project. We should now discuss the implementation of the stated designs. This is discussed further in the next chapter titled *"Implementation"*.

# CHAPTER 5: IMPLEMENTATION

This chapter discusses one of the most important aspects of the project. The software goals outlined in Chapter 1 have been analysed and adequately designed as described in Chapters 2, 3 and 4. But these need to be implemented for the successful completion of the project. The software objectives achieved and discussed in this chapter include,

1 Conversion of the application to an applet.
2 Improvements and redesign of the GUI layout.
3 Development of a software joystick to aid in the manipulation of the surge, roll, pitch, yaw and heave movements of the robot.
4 Sending commands to the robot, in this case, *camera commands.*
5 Receiving telemetry router data continuously from Kambara.

The rest of the chapter is organised as follows,

- Section 5.1 outlines the improvements made to the GUI layout and describes the new structure.
- Section 5.2 discusses the implementation of the applet and the problems associated with it.
- Section 5.3 details the issues associated with the implementation of the software joystick.
- Section 5.4 describes the implementation of the network improvements that will be a stepping stone to conforming to the long-term goal of the Interface Network Design.
- Section 5.5 summarises the chapter concentrating on the outcomes of the implementation stage.

## *5.1 GUI Layout*

As discussed in Section 2.3.2, the software architecture of the GUI was improved upon. This resulted in the state information being displayed as follows:

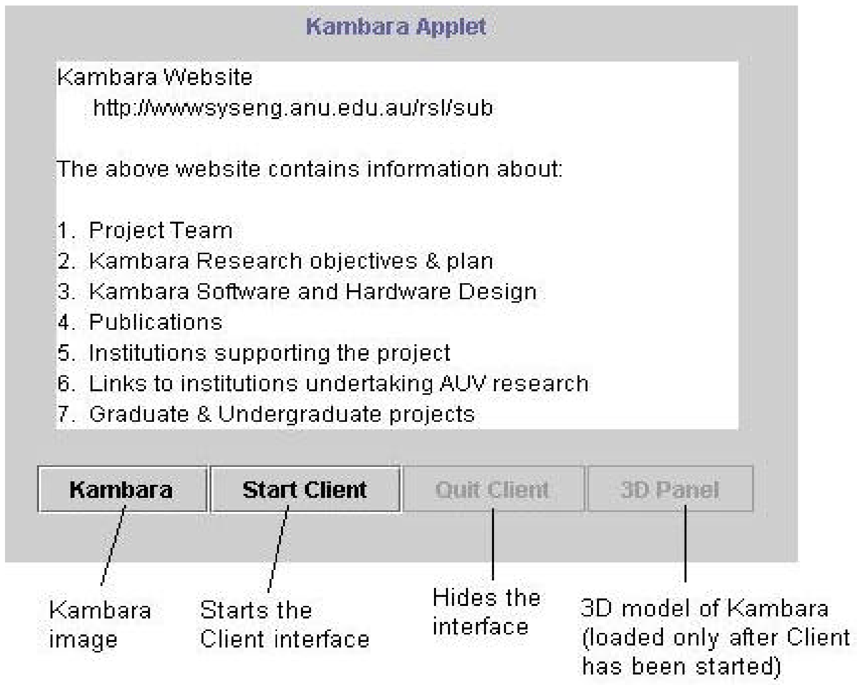
1) TelemetryPanel: relating to the general information about the vehicle.
2) PositionPanel: a two-dimensional indication of the position of the vehicle and the comparative position of the target.
3) DataPanel: depicts two sets of values; the numerical values coming directly (*raw*) from the sensors on the vehicle and the numerical values that have been *derived* on-board the vehicle.
4) MessagesPanel: any error, status or warning messages that are transmitted to the client.
5) NavigationPanel: controls used to navigate the robot when in teleoperation mode. Currently, this is implemented using a software joystick.
6) ThreeDFrame: A separate frame holding the 3D model of Kambara and depicting its current rotation and translation state.
7) CameraPanel: shows and manipulates the current pan, tilt and zoom values of the camera.

The first five panels were constructed using a tabbedPane and implemented using GUI components developed using source code found on the Internet or developed originally. As mentioned in Section 2.3.2, the CameraPanel is in constant view in the current interface. The snapshots of the panels are provided in Appendix E.

The only problem encountered with the GUI was the instability of the ThreeDFrame, especially on Solaris machines. This was predominantly due to Java3D, which is still in its infancy in terms of development. It is believed that future versions of Java3D would be much more stable and should solve the problem. But in case this problem still persists, there should be a reassessment of the use of the 3D model.

## *5.2 Applet*

The Java applet used to load the operator interface is given in Figure 21. The applet contains a brief description of the Kambara project and has four buttons that have varying purposes.



Kambara Applet

Kambara Website
http://wwwsyseng.anu.edu.au/rsl/sub

The above website contains information about:

1. Project Team
2. Kambara Research objectives & plan
3. Kambara Software and Hardware Design
4. Publications
5. Institutions supporting the project
6. Links to institutions undertaking AUV research
7. Graduate & Undergraduate projects

Kambara
Start Client
Quit Client
3D Panel
Kambara image
Starts the Client interface
Hides the interface
3D model of Kambara (loaded only after Client has been started)

**Figure 21: Kambara Applet**

1) *Start Client* is used to start an instance of the interface.
2) *3D Panel* starts the ThreeDFrame holding the Wavefront model (Figure 10). It is activated when the user starts the client.
3) *Quit Client* will be used when the client wants to end the session. It hides the interface and the client can then quit the applet.
4) *Kambara* shows the user an image of the robot (Figure 2).

But there were problems associated with loading the applet into a web browser such as Netscape. The problems are discussed in the following Sections 5.1.1 and 5.1.2.

### 5.1.1 Java Plug-In for Web Browsers

Current web browsers do not support Swing libraries used in Java 2 [9], the version of Java used in the development of the interface. Java Plug-Ins [11] need to be installed and run with the present versions of Internet browsers, notably Netscape. This is in order for them to not only display Java3D objects, but also support Swing libraries.

### 5.1.2 HTML Conversion

Another problem was that the HTML document loading the applet needs to have tags that will specify the browser to use the Java Plug-In and not its own Java Virtual Machine (JVM). By default, browsers such as Netscape use their own JVM and need to be specifically informed to use the Java Plug-In. Therefore, a Java Plug-In HTML Converter [11] needs to be used to convert standard HTML pages to a form that specifies the usage of the Java Plug-In.

The above problems added to the slow performance of the applet, especially in loading the interface. This aspect is further discussed in the next chapter that describes the testing procedures used during the project.

## *5.3 Software Joystick*

The joystick implementation was quite time consuming with issues needing to be addressed at every juncture. Therefore, the joystick had to be implemented in a number of steps based on the issues. These issues are detailed in the following subsections.

### 5.3.1 Co-ordinate System

Because the joystick was implemented as two 2D panels, TopPanel and SidePanel, issues corresponding to the Co-ordinate system needed to be addressed. Specifically, we needed to address the fact that in 2D, Java only recognises the X and Y-axes. Therefore, depending on the view being used, we needed to include the Z-axes in place of either the X or the Y-axes.

Another issue was that the view might not necessarily interpret the horizontal direction to be the X-axes with the vertical direction as the Y-axes. This is evident in Figure 22, which shows the orientation of Kambara in its three axes and the corresponding interpretation required for the Top and Side Views.

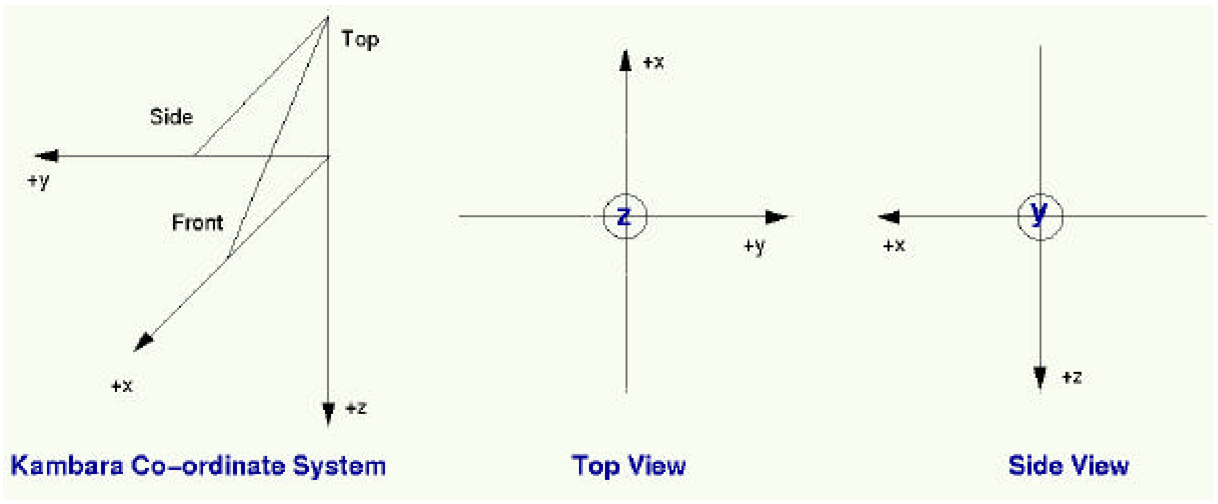


**Figure 22: Co-ordinate System (Kambara and Joystick Panels)**

### 5.3.2 Rotation Angles and Quaternions

The second problem faced during the implementation of the joystick was the computation of the angles. These could not be computed in four quadrants, as is normally the case. It had to vary depending on the view. For example, for either of the panels, the yaw, pitch or roll angles will have a range of ± 180°. But while the roll and yaw angles are computed with respect to the vertical axis in the TopPanel, the pitch is computed with respect to the horizontal axis in the SidePanel. This is depicted in Figure 23.

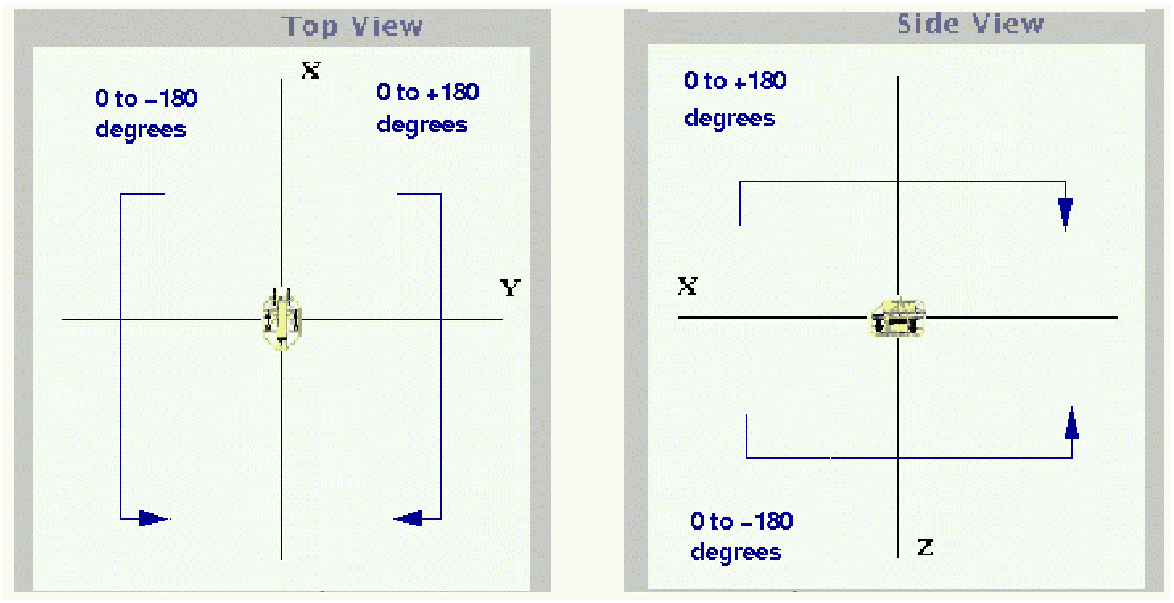


**Figure 23: Relative movements in the TopPanel and the SidePanel**

The other problem concerning the angles was the conversion to *Quaternions*. Quaternions are used for a singularity-free representation of the attitude of the robot. In the operator interface, the 3D model uses quaternions to represent the orientation of the robot. While Java3D has methods and constructors to create quaternions, there are none for creation using three angles (roll, pitch and yaw).

The solution to the problem was to create individual quaternions for the roll, pitch and yaw and then multiply them together to compute the overall quaternion for the 3D model. That is, methods were developed that would compute the quaternion given one angle at a time. Therefore for the roll, pitch and yaw, the individual quaternions computed were as follows:

The quaternion was computed using the *Quat4D* class available in the *javax.vecmath* library in Java3D API 1.1.

```
For Roll:  Quat4d{sine(Roll/2), 0, 0, cosine(Phi/2)}
For Pitch: Quat4d{0, sine(Pitch/2), 0, cosine(Theta/2)}
For Yaw:   Quat4d{0, 0, sine(Yaw/2), cosine(Yaw/2)}
```

These individual quaternions are then multiplied together to get the final quaternion. Java3D provides methods for this very purpose.

### 5.3.3 Manipulations using MovementsPanel

The last problem encountered was concerning the manipulations the operator would make using the MovementsPanel. The question was whether the MovementsPanel would act as a correction mechanism for the operations made using the TopPanel and the SidePanel or whether it would have a similar role to those panels, albeit a much more specific role.

It was decided that the MovementsPanel could perform both roles. For example, if the user wanted to correct a pitch of 45 degrees to 30 degrees, a manual change of –15 degrees could be implemented using the MovementsPanel. Alternatively, if the operator wanted to change the pitch by a further 50 degrees, which usually accomplished using the SidePanel the MovementsPanel could implement that as well. In the end, the ultimate interpretation lies with the operator.

## *5.4 Networking*

The implementation of the network design was relatively easy. Use of the UDP protocol ensured that the confirmation of packets or commands was not necessary. The only foreseeable problem lay in the Java Virtual Machine allowing socket connections to listen, connect and accept information. But the policy file (*java.policy*) that determines what code is permitted to perform each action already allowed the use of socket on the defined ports.

There were two major problems associated with the network communication with the robot. The first problem was concerning the interpretation of the Telemetry Router data while the second problem was about the conversion of the camera values into bytes.

### 5.4.1 Data Interpretation

The GUI components were developed using the simulator data in mind. The Telemetry Router data is similar to the simulator with a few minor differences. But currently, the only meaningful information that the operator receives in the telemetry router data is the vehicle state (position and orientation), vehicle velocity and the camera values. Therefore, it was determined that a reassessment of the GUI components will be done once the telemetry data contains more useful information.

The vehicle state in the telemetry data is determined from a compass installed as part of the Kambara hardware. It computes the position and the quaternion that is subsequently viewed on the ThreeDFrame.

### 5.4.2 Conversion to Bytes

Problems arose when the camera command needed to be created in the form of UDP packets. The problem was with the conversion of the double and long values into bytes. Because a packet is the simplest form of data, all the information contained in it is in the form of bytes. Doubles and longs are 64 and 32-bit floating point datatypes respectively.

The only solution to the problem lay in conversion of all the values into one standard 64-bit value and subsequently splitting the values into groups of 8 bits each. This would require bit shifting, which was difficult because only 32 bits can be shifted at a time in Java. Therefore, the shifting was accomplished in two stages. The following was the method employed for shifting the values.

```
data[j+4] = (byte) (( value & 0xff000000) >> 24);
data[j+5] = (byte) (( value & 0x00ff0000) >> 16);
data[j+6] = (byte) (( value & 0x0000ff00) >> 8);
data[j+7] = (byte) (( value & 0x000000ff));
value >>= 32;
data[j] =   (byte) (( value & 0xff000000) >> 24);
data[j+1] = (byte) (( value & 0x00ff0000) >> 16);
data[j+2] = (byte) (( value & 0x0000ff00) >> 8);
data[j+3] = (byte) (( value & 0x000000ff));
```

The two sets of 32 bits are shifted such that their values are stored at the Least Significant bit of each byte. The actual value itself needs to be shifted by 32 after the first set so as to access the remaining 32 bits.

## 5.5 Control Modes

The basic control modes had already been implemented in the system. On loading the operator interface the user is in the control mode, "*Disconnected*". The options under the *Control Mode* pulldown menu are disabled at this stage. Once the user establishes a connection to the server, the control mode switches to the "*Observer Mode*". At this particular stage, the user can only view state information that is received from the robot.

On selecting "*Individual Control*" mode in the pulldown menu, the user can manipulate the state of the camera and the motors on the robot. This is achieved by providing several components in the CameraPanel and the TelemetryPanel.

The "*Teleoperation Control*" mode provides navigation mechanisms to the robot. That is, it shows the NavigationPanel to the operator. "*Supervisory Control*" mode is supposed to allow the user to plan or map a route for the robot, or execute a series of commands in sequence.

The shortcomings at the beginning of the project were related to improper and simplistic implementation of the Teleoperation and Individual control modes. With the implementation of the camera commands, the Individual control mode has been greatly enhanced. And the implementation of the software joystick has contributed to enhancement of the Teleoperation control mode. Also, the implementation of camera commands has paved the way for subsequent implementation of navigation commands. This is discussed further in Chapter 7.

The supervisory control mode has not been implemented. But, with the development of the camera commands, a means to planning a route for the robot can be designed and implemented. This would require the use of live video, which incidentally is also part of the NavigationPanel. This is also discussed further in Chapter 7.

## 5.6 Chapter Summary

While the creation of the applet did not pose any problems, the loading of the applet onto a web browser was a different matter entirely. The problems were associated with Java Plug-In for Java 2 and HTML conversion relating to the Java Plug-In.

The GUI has been redesigned and along with the menubar and the message area, it now consists of five tabbedPanes, a ThreeDFrame for the 3D model and a CameraPanel that is in constant view of the user. The main problem concerning the GUI was the instability of the ThreeDFrame, especially on Solaris machines.

The joystick was successfully implemented with the majority of the problems confined to co-ordinate and angle conversions. The network design developed in Chapter 4 (Section 4.3) was also implemented with camera commands sent to the robot and feedback received through the telemetry router data.

The control modes were also properly implemented and enhanced. This was partially achieved through the development of the software joystick and the enhancement of the network design.

The next step in the process of successfully completing the project is to validate and verify the correctness of the implementation with respect to the analysis and the design. The testing procedures used for this purpose are detailed in the next chapter.

# CHAPTER 6: VALIDATION AND VERIFICATION

The implementation needs to be validated and verified so as to determine if it conforms to the analysis and the design. This chapter discusses the testing of the implementation, thereby determining whether it satisfies the objectives of the project as stated in Section 1.1.2. This chapter is organised as follows,

- Section 6.1 describes the testing procedures that were central to the implementation.
- Section 6.2 details the testing of changes implemented in the GUI
- Section 6.3 outlines the testing issues related to the implementation of the applet.
- Section 6.4 discusses the testing of the joystick and the validation issues related to it.
- Section 6.5 details the testing concerning communication with the robot.
- Section 6.6 explains the testing conducted for determining if control modes have been correctly implemented.
- Section 6.7 summarises the chapter concentrating on the outcomes of the testing process.

## 6.1 Testing Procedures

The operator interface was predominantly tested on a Solaris machine. This is because the Windows operating system poses constraints on the location of the client and the server. That is, on a Windows machine, both the client and the server needed to be run on the local host, and this leads to a poorer performance.

Another reason for the preference of Solaris was because the server had to be executed on *hughes.anu.edu.au*, which could not be achieved with a client running on Windows. The reason being that Kambara would only accept or listen for camera commands from that particular machine. While this scenario is not ideal, it had to suffice for testing purposes, especially concerning camera commands and the client receiving telemetry data.

Also, the camera was predominantly plugged onto *nozomi.anu.edu.au* and tested there. This was not ideal, as one could only observe the status at the beginning and the end and thereby determine the changes in the pan, tilt and zoom.

## 6.2 GUI

It is necessary to test changes implemented in the GUI to ensure that newly added components still get updated with state information. This is especially important for the DataPanel that incorporates the raw sensor data and the derived data that is computed on-board Kambara. These were tested by using simulator data (Appendix C) as the telemetry router data does not contain meaningful sensor values currently. The redesign also satisfied the design principles listed in Section 2.2.1 with space, proximity and alignment used to maximum effect.

## 6.3 Applet

Due to reasons mentioned in Chapter 5 (Section 5.1), Applet was predominantly tested using the *Java AppletViewer* [6], which is part of the *Java Developers Kit* (JDK1.2.1). The applet was loaded onto a browser that had the Java 2 Plug-In installed. The only problem was the poor performance associated with loading the applet. This would be due to the large number of objects that need to be loaded, HTML conversion and the use of Java 3D in the interface.

The applet was tested to connect to a web server and thereby, receive state information from the robot. Tests were also conducted with respect to sending camera commands to the robot. The above was achieved successfully. In the above cases, the web server was located locally as well as on a separate machine.

## 6.4 Software Joystick

Majority of the testing concerning the joystick was associated with computation of the roll, pitch and yaw angles, from the keyboard and mouse inputs (on the TopPanel and the SidePanel)

As the values were displayed on the MovementsPanel, it acted as a good feedback and testing mechanism for the translation and rotation manipulations. The 3D model also acted as a good testing mechanism as well through the MovementsPanel. By negating the values, the model should go back to its original state, which was achieved successfully.

## 6.5 Communication with Kambara

There were two stages of testing in regard to communication with Kambara. The first stage was concerned with sending commands to the camera on-board the robot. The second stage was related to obtaining Telemetry Router data from the robot.

### 6.5.1 Camera Commands

The camera commands were tested using the Sony D-230 Camera shown in Figure 24. As mentioned before, in the earlier stages of development, the camera was not on-board the robot.

**Figure 24: Sony D-230 Camera**

At the latter stages of development though, the camera was installed on-board Kambara and the commands were then successfully tested as well (see Figure 25). The camera values are also constantly transmitted as part of the telemetry router data. These are subsequently displayed on the CameraPanel thereby informing the operator whether the command was successfully transmitted and implemented.

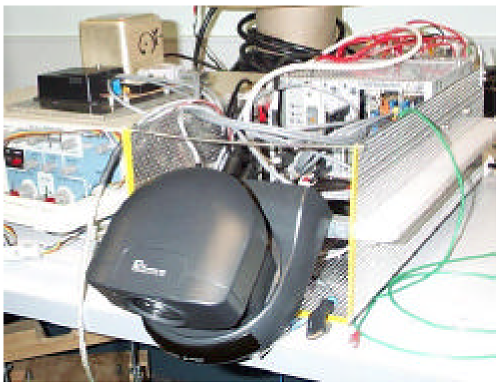


**Figure 25: Sony Camera on-board Kambara**

### 6.5.2 Telemetry Router Data

The Telemetry Router data contained meaningful values of just the vehicle state apart from the camera values. The vehicle state was determined from the compass on-board the robot. The data was sent as a continuous stream of packets.

Tests were conducted to determine the update frequency of the GUI under a number of conditions. These were conducted under two different environments

- o An ideal Ethernet environment with no network traffic associated with it.
- o A real world situation where performance will be affected due to network traffic.

The results are listed in Table 3.

| TEST | Number of packets sent by Kambara | Number of packets received by the client | Time taken (s) | Update frequency (Hz) |
|---|---|---|---|---|
| **With server present** | 1779 | 638 | 500 | 1.24 |
| **Without server present** | 5913 | 3951 | 570 | 6.58 |
| **Ideal Ethernet environment (server present)** | 4462 | 1586 | 450 | 3.52 |
| **Ideal Ethernet environment (Without server present)** | 6247 | 3973 | 415 | 9.57 |

**Table 3: Telemetry Data Update Rate**

The performance was not satisfactory when both the client and the server are located on the same machine. The performance as expected is much better in an ideal Ethernet environment with an increase of around 2.5 to 3 Hz. This is primarily due to the network traffic, which is not present in an ideal environment.

The server also affects the rate of GUI updates. This is due to the threads executed by the server for obtaining information from the robot and passing that information onto the clients.

## *6.6 Control Modes*

The observer, individual and teleoperation control modes were successfully implemented and tested. The first stage was examining the disconnected state. In this state, the user is unable to select any of the control modes on the pulldown menu.

Once the user is connected to the server, only the components relating to viewing the state information were made visible. On selecting the Individual control mode, the user was able to manipulate the motor torque values and also send camera commands to the robot through buttons that are added on the TelemetryPanel and the CameraPanel. In the Teleoperation mode, the NavigationPanel was made available to the operator as another tabbedPane.

Finally, the control and network aspects were combined and tested. Two clients were connected to the server with one of them given control of the robot. The other client made a request for control by selecting the Individual Control option in the pulldown menu. This client was refused control as expected. Also, control was relinquished when the controlling client returned to the observer mode and the second client was subsequently able to successfully obtain control of the robot.

## *6.7 Chapter Summary*

This chapter detailed the testing procedures used for validating and verifying the implementation of various aspects of the project. The applet was predominantly tested using Java AppletViewer but it was loaded into a Java 2 compliant browser. The joystick testing was concentrated on the computation of the rotations angles (roll, pitch, and yaw) with the MovementsPanel acting as a good feedback mechanism for determining the values.

A Sony D-230 camera was used for testing the camera commands and only during the latter stages of the implementation was the camera on-board the AUV. Telemetry router data was received from Kambara. Network traffic and local execution of the Kambara server seriously affect the GUI updates based on the telemetry data.

The next step is to draw conclusions including summarising the thesis and discussing my contribution to the Kambara project (more specifically, the development of the operator interface). This is detailed in the next chapter, which also includes discussion about further work that should be accomplished in order to successfully complete the operator interface.

# CHAPTER 7: CONCLUSION AND FURTHER WORK

## *7.1 Conclusion*

This project has continued the development of the operator interface for the Kambara project. Design and implementation of an applet has accomplished one of the main aims of manipulating Kambara over the Internet.

The redesign of the GUI along with increased use of Java 2 Swing libraries have increased the flexibility of the different components of the interface. The investigation of NCSA Portfolio for using a VRML model of Kambara in place of the Wavefront model was unsuccessful because of constraints imposed by the Java 3D API and NCSA Portfolio itself.

A software joystick has been developed to assist with the navigation of the robot. It will ultimately also help with direct teleoperation of the AUV. The manipulations of the joystick are simulated using the 3D model. Because the development on-board Kambara has not reached a stage where it accepts navigation commands, these have not been implemented.

The interface receives telemetry router data from the robot in real time as opposed to a simulator[7]. Experiments were conducted for determining the update rate of the GUI. It is found to be 6.58 Hz without server executing locally and 1.24 Hz with the server present on the system. Network traffic also affects the updates as similar tests on an ideal Ethernet environment give update rates of 3.5 Hz (with the server present) and 9.57 Hz (without the server on the system).

The network capabilities now also include the ability for the client to send camera commands to the robot. These have been successfully tested with the Sony D-230 camera on-board Kambara.

The different control modes have also been implemented and successfully tested. The individual and teleoperation control modes have been enhanced as a result of the development of the software joystick and the successful implementation of the camera commands.

## *7.2 Contributions*

One of the major contributions of my project is the development of the software joystick. Similar interfaces do not use such a mechanism for navigation. The joystick will be extremely useful in the areas of teleoperation and even supervisory control.

The implementation of obtaining state information data from Kambara is the second major contribution of my project. To date, the communication with Kambara had been simulated

[7] The original system obtained its input from a simulator RSISE. A vector of the simulator data is provided in Appendix C

with state information packets being received from an artificial source. Obtaining real time telemetry router data and updating the GUI from that data has been one of the motivations behind the Kambara project.

Development of a means of sending commands to the robot is another important aspect of the operator interface, a part of which has been accomplished through this project.

## *7.3 Further Work*

There is still much work to be done in order to get the interface properly operational. This project has focussed on the development of navigation mechanisms for moving the robot and also ensuring that there is proper communication between the client and the robot. These features, especially the camera commands have enhanced the control mode mechanisms available to the client. However, more work needs to be done especially in regard to obtaining live video from the robot.

### 7.2.1 Video Feed

The NavigationPanel[8] contains the joystick but it has also been designed to provide live video to the operator. With the camera commands having been implemented, images can be obtained depicting the current view of the camera. Live video is required to aid direct teleoperation and in order to map a route for the robot (supervisory control).

### 7.2.2 Supervisory Control

A plan needs to be made for supervisory control mode especially now that the robot accepts commands from the client. The use of the CameraPanel along with live video can facilitate planning of a route for the robot, or execution of a series of commands in sequence. This should be done in anticipation of future robot operations when an optic fibre tether is no longer present.

### 7.2.3 Network Architecture

The current design is similar to the long-term goal that has been described in Chapter 4 (Section 4.3), with one major difference. The protocol used for communication between the Kambara server and the robot is UDP. While this protocol suits our purpose of getting the commands easily and quickly to the robot and is extremely useful for testing, it is not adequate in the long term. This is because a much more reliable protocol is necessary in real time situations.

Once Kambara is operational, it is imperative that most of the commands are received by the robot and acted upon. Therefore, further improvements with regard to the existing design need to be implemented.

[8] A snapshot of the NavigationPanel is available in Appendix E5

### 7.2.4 Logging information

Presently, the logging system is in its primitive stages, with the menu option having been established, but no logging actually commencing. The method of logging needs to be examined. When the client is running as an applet, the Java security arrangements will not allow writing on a client's system without their permission. Therefore, it may be desirable to have a separate logging process running at the server end. File storage will also be a problem due to the frequency and the amount of data being transmitted.

# BIBLIOGRAPHY

**[1]** McPherson, C., 1999. *Network-based Operator Interface for an Underwater Robot*. Department of Engineering Honours Project Thesis, ANU.

**[2]** Gaskett, C., Wettergreen, D., Zelinsky, A., 1998. *Development of a Visually guided Autonomous Underwater Vehicle*. Oceans' 98

**[3]** Gaskett, C., Wettergreen, D., Zelinsky, A., 1999. *Reinforcement Learning for a Visually guided Autonomous Underwater Vehicle*. International Symposium on Unmanned Untethered Submersibles Technology. New Hampshire, USA

**[4]** Kambara Software Design:
http://wwwsyseng.anu.edu.au/rsl/sub/software/software_design.html

**[5]** Couch, J., 1999. *Java 2 Networking*. McGraw-Hill. New York

**[6]** Flanagan, D., 1996. *Java in a Nutshell*. O'Reilly and Associates. USA

**[7]** Lewis, J., Loftus, W., 1998. *Java Software Solutions*. Addison Wesley. USA

**[8]** Visual Design Resources
http://www.indiana.edu/%7Eiirg/ARTICLES/VIZRES/resource_page.html

**[9]** Java 1.2 API specification: http://java.sun.com/products/jdk/1.2/docs/api/index.html

**[10]** Java3D 1.1 API specification:
http://java.sun.com/products/java-media/3D/forDevelopers/j3dapi/index.html

**[11]** Java Plug-in for Browsers: http://java.sun.com/products/plugin/

**[12]** NCSA Portfolio: http://www.ncsa.uiuc.edu/~srp/Java3D/portfolio/

**[13]** Web Interface for TeleScience: http://wits.jpl.nasa.gov/

**[14]** Bouvier, DJ., 1999. *Getting started with the Java3D API*. Sun Microsystems.

**[15]** Tanenbaum, AS., 1996, *Computer Networks*. Prentice-Hall of India Private Limited

# APPENDIX A: NCSA PORTFOLIO

NCSA Portfolio is a collection of utility objects to use within your Java3D programs. Some of the newer features available with NCSA Portfolio include,

- Input devices for Flock of Birds, Cave Wand, Spacetec SpaceOrb 360, Microsoft Force Feedback Sidewinder, and mouse.
- Unified InputDevice Interface introduced
- New PickTool features for use with the Wand or SpaceOrb.
- New VTK ASCII loader
- New VTK binary loader (big endian only)
- New NFF loader
- New TextureCache object

Other NCSA Portfolio features include:

- **Universal model loader interface:** All supported 3D model types can be loaded through a single call to a ModelLoader object that acts as an interface to all 3D model readers. Supported loaders include
    - NCSA's 3D Studio (3DS) loader
    - NCSA's AutoCAD DXF loader (Version 12)
    - NCSA's Protein Data Bank (PDB) loader
    - NCSA's Digital Elevation Map (DEM) loader
    - NCSA's Imagine (IOB) loader
    - NCSA's TrueSpace (COB) loader (Version 2)
    - NCSA's VRML 97 (WRL) loader
    - NCSA's PLAY loader
    - NCSA's VTK ASCII loader
    - NCSA's VTK binary loader (big endian only)
    - NCSA's NFF loader
    - NCSA's Wavefront OBJ (including material files for colours) loader
    - Sun's Lightwave 3D Scene loader

  You can also add your own loader interfaces, and the ModelLoader will automatically recognise them.

- **Unified InputDevice Interface:** Starting in NCSA Portfolio release 1.3 beta 1, there is support for a unified input device interface. Using this interface, we can create a way to interchange input devices in programs without having to recompile the program itself. This is really useful if you are working on one device (say a desktop), and have to move the program to another type of device (such as a four-screened Portal). Future releases are expected to individually address how buttons are mapped on input devices.

- **Event management:** The object library provides a mechanism for managing events within Java 3D programs through the ToolManager and its Tools. Using ToolManager, application developers can easily switch between operating modes. In this release of NCSA Portfolio, we include a PickTool to manipulate 3D models in a scene and a ViewTool for changing the View within a scene. Application developers can also write their own Tools to use with the ToolManager.

- **Collaborative real-time updates:** Imagine two people starting Java 3D programs on two different machines, and have either of them control the program! NCSA Portfolio includes networking objects that allow you to easily create real-time collaborative applications.

- **Canvas snapshots:** Take a snapshot of the current Canvas3D view and save it as a JPEG file!

- **Record and replay of application events:** The NCSA Portfolio library can record the actions of Tools controlled by the ToolManager. Once the actions are recorded and saved, they can be loaded and replayed. You can use this to script live animations.

# APPENDIX B: USER DATAGRAM PROTOCOL

UDP (User Datagram Protocol) is a connectionless protocol that provides a way for applications to send encapsulated raw IP Datagrams and sends them without having to establish a connection. Many client-server applications that have one request and one response use UDP rather than go to the trouble of establishing and later releasing a connection.

In UDP, you create a message to be sent, then create a UDP socket and tell it to send the message to the destination. Once you have sent it, the socket cannot be used again. You will need to create a new socket if you want to send more data (or packets).

At the destination end, the application will be listening for data on a UDP socket as well. Since the data from a UDP transmission is, by nature, a single packet, then we know that once we have received it, it can be processed immediately. However, just as in sending data, if we want to listen for more information on that same port, then we must create a new UDP socket to replace the one that has just been used.

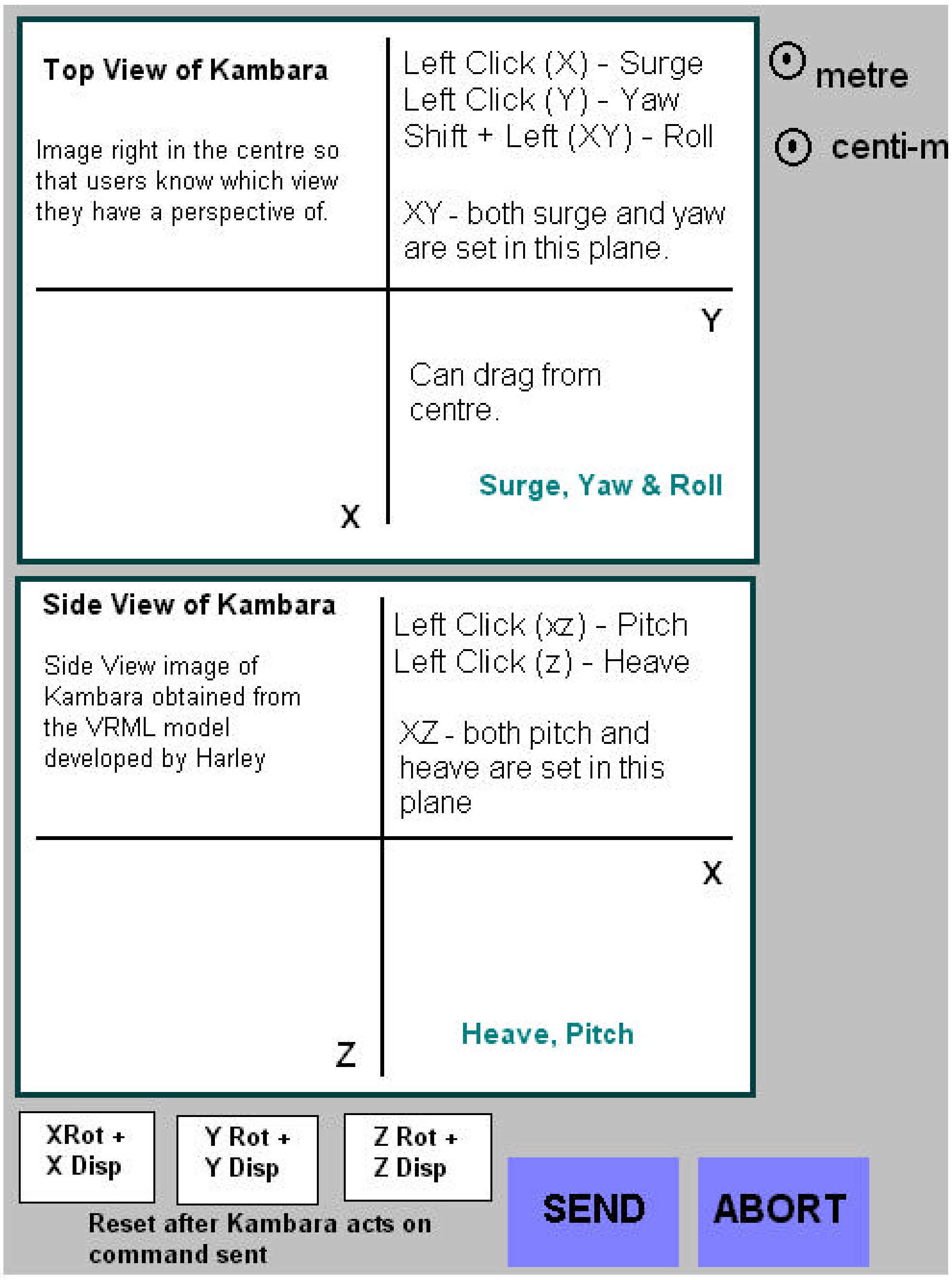
Therefore, if we need to create a new socket for each send and receive, then the communications can only be one way. That is, if the initial sender wanted to know if an answer had been sent, then it too must create a listening UDP socket just like the receiver.

Once the receiver has been given the data, there may still be other problems. UDP does not include any form of error correction apart from what the lower-level standards may have included. Therefore it is possible that the data you may receive nay have been corrupted in some way. You will need to make sure either that your custom protocol has some error management built in, or that the application knows how to detect and ignore erroneous messages without crashing.

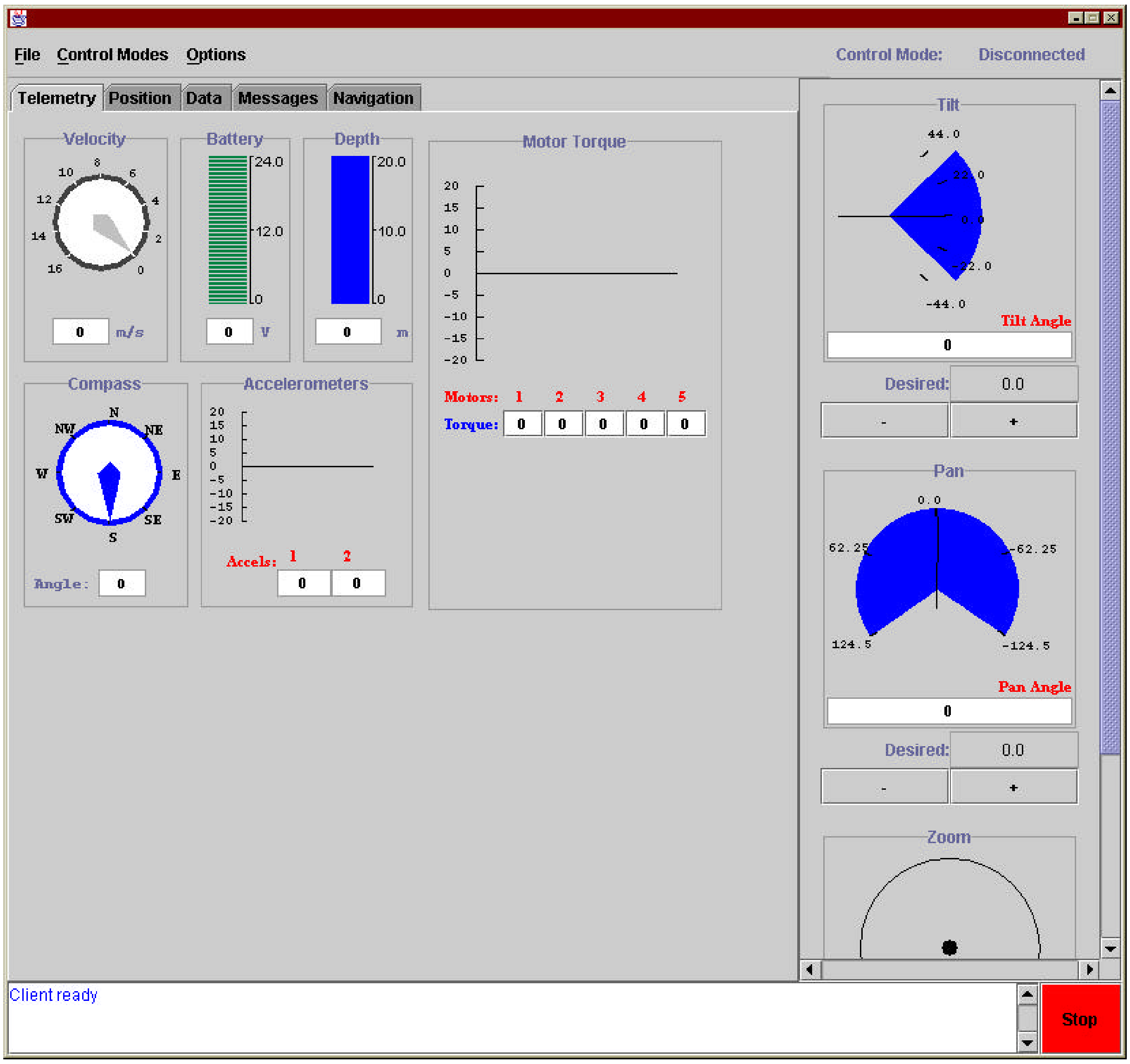
# APPENDIX C: VECTOR OF SIMULATOR DATA

| Field element / range (doubles) | Data Represented |
|---|---|
| 0-2 | Vehicle Position x, y, z values |
| 3-6 | Vehicle Orientation (Quaternion) |
| 7-9 | Vehicle local velocity |
| 10-12 | Vehicle angular velocity |
| 13-17 | Motor torque |
| 18-20 | Vehicle target |
| 21 | Battery voltage |
| 22-23 | Accelerometer totals |
| 24 | Compass angle |
| 25 | Camera pan value |
| 26 | Camera tilt value |
| 27 | Camera zoom value |
| 28-32 | Motor current |
| 33-37 | Motor voltage |
| 38-40 | Derived accelerometer 1 u, v, w values |
| 41-43 | Derived accelerometer 2 u, v, w values |
| 44-46 | Raw accelerometer 1 u, v, w values |
| 47-49 | Raw accelerometer 1 u, v, w values |
| 50-52 | Compass magnetic disturbances |

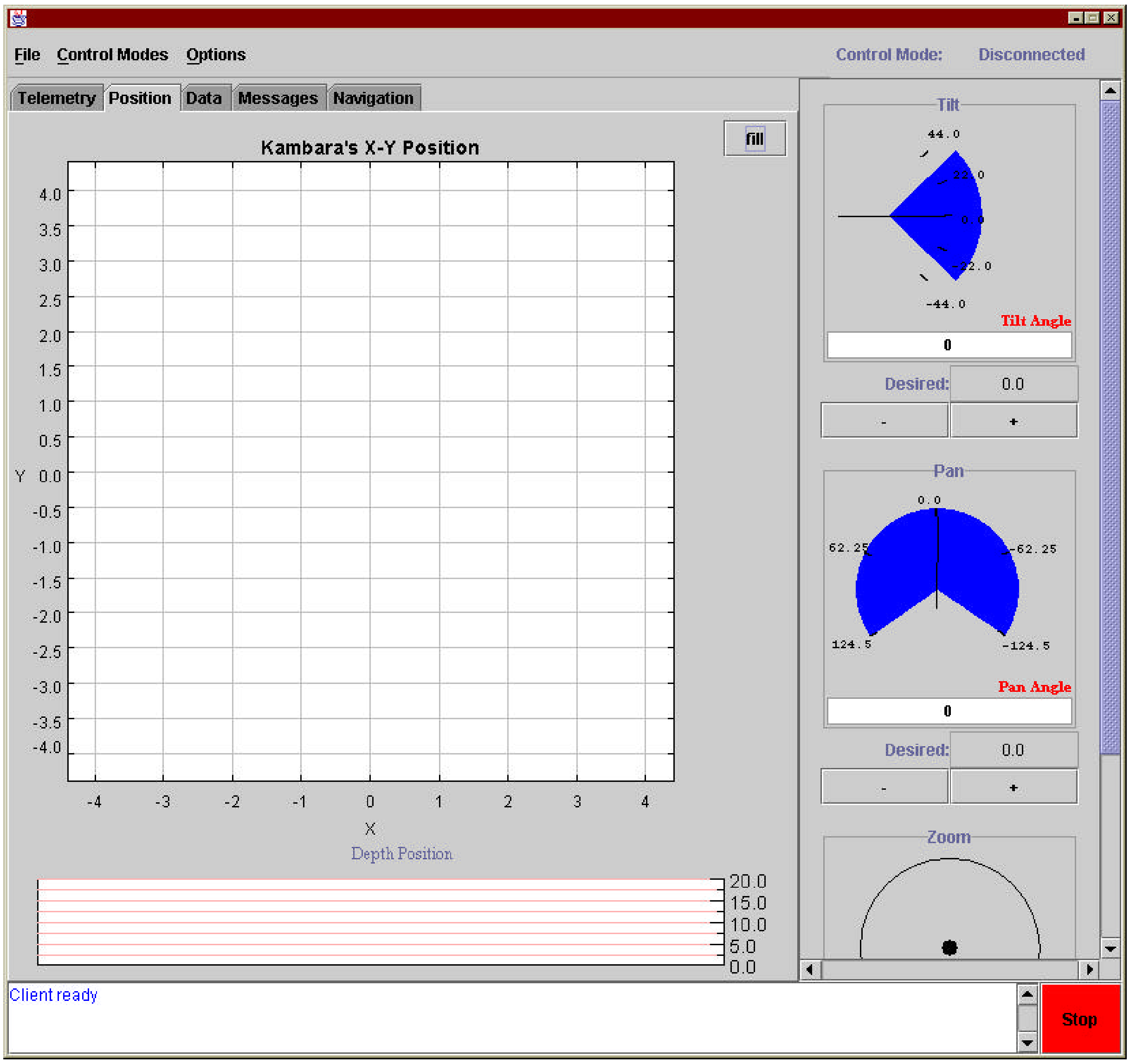
# APPENDIX D: SOFTWARE JOYSTICK DESIGN


Top View of Kambara
Image right in the centre so that users know which view they have a perspective of.
Left Click (X) - Surge
Left Click (Y) - Yaw
Shift + Left (XY) - Roll
XY - both surge and yaw are set in this plane.
Y
Can drag from centre.
X
Surge, Yaw & Roll
metre
centi-m
Side View of Kambara
Side View image of Kambara obtained from the VRML model developed by Harley
Left Click (xz) - Pitch
Left Click (z) - Heave
XZ - both pitch and heave are set in this plane
X
Z
Heave, Pitch
XRot + X Disp
Y Rot + Y Disp
Z Rot + Z Disp
Reset after Kambara acts on command sent
SEND
ABORT

# APPENDIX E1: SCREENSHOT OF TELEMETRYPANEL

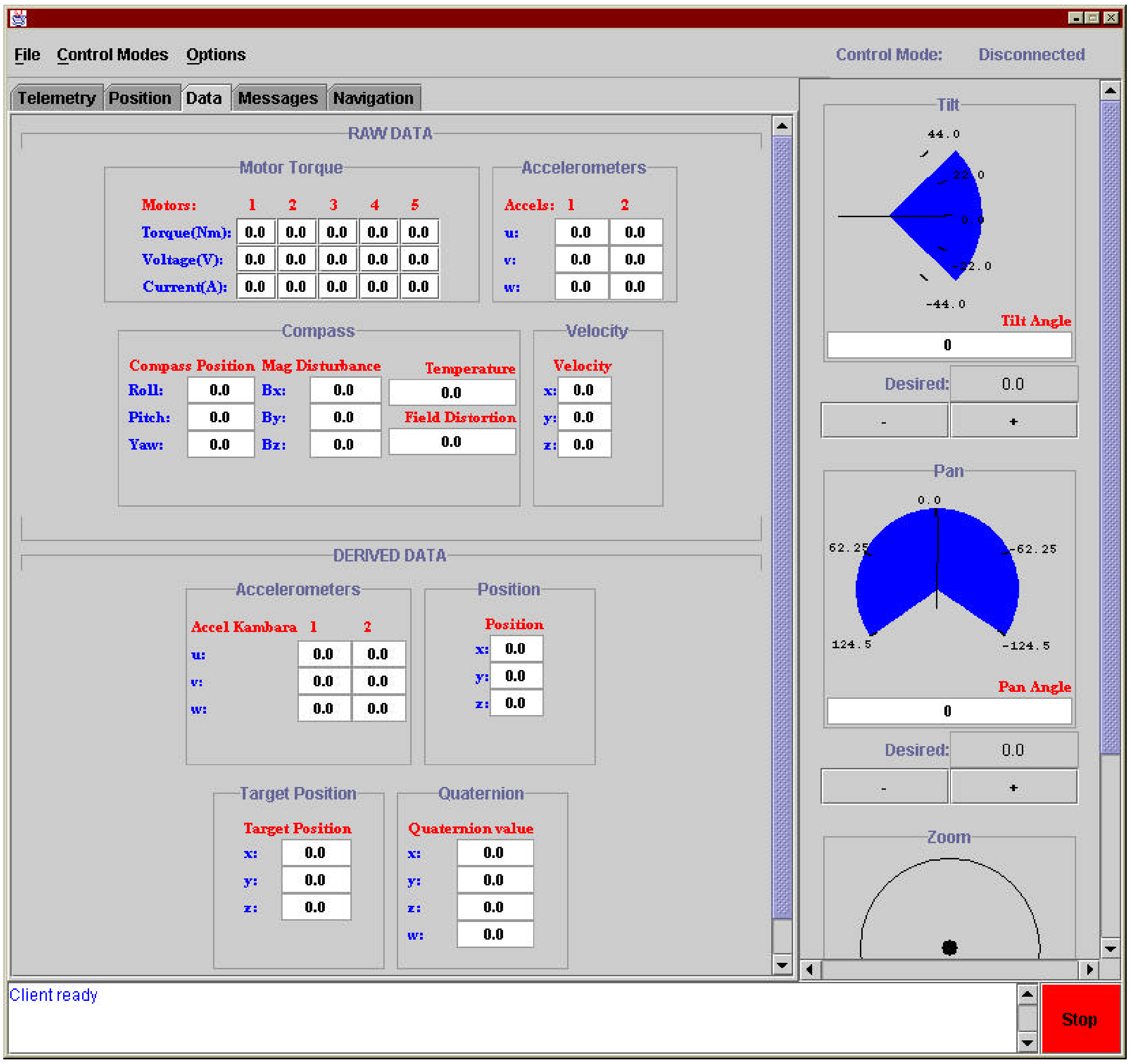


# APPENDIX E2: SCREENSHOT OF POSITIONPANEL



Client ready

# APPENDIX E3: SCREENSHOT OF DATAPANEL



Control Mode: Disconnected

RAW DATA

Motor Torque

| Motors: | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Torque(Nm): | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Voltage(V): | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Current(A): | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

Accelerometers

| Accels: | 1 | 2 |
|---|---|---|
| u: | 0.0 | 0.0 |
| v: | 0.0 | 0.0 |
| w: | 0.0 | 0.0 |

Compass

| Compass Position | | Mag Disturbance | |
|---|---|---|---|
| Roll: | 0.0 | Bx: | 0.0 |
| Pitch: | 0.0 | By: | 0.0 |
| Yaw: | 0.0 | Bz: | 0.0 |

Temperature: 0.0

Field Distortion: 0.0

Velocity

| Velocity | |
|---|---|
| x: | 0.0 |
| y: | 0.0 |
| z: | 0.0 |

DERIVED DATA

Accelerometers

| Accel Kambara | 1 | 2 |
|---|---|---|
| u: | 0.0 | 0.0 |
| v: | 0.0 | 0.0 |
| w: | 0.0 | 0.0 |

Position

| Position | |
|---|---|
| x: | 0.0 |
| y: | 0.0 |
| z: | 0.0 |

Target Position

| Target Position | |
|---|---|
| x: | 0.0 |
| y: | 0.0 |
| z: | 0.0 |

Quaternion

| Quaternion value | |
|---|---|
| x: | 0.0 |
| y: | 0.0 |
| z: | 0.0 |
| w: | 0.0 |

Tilt Angle: 0

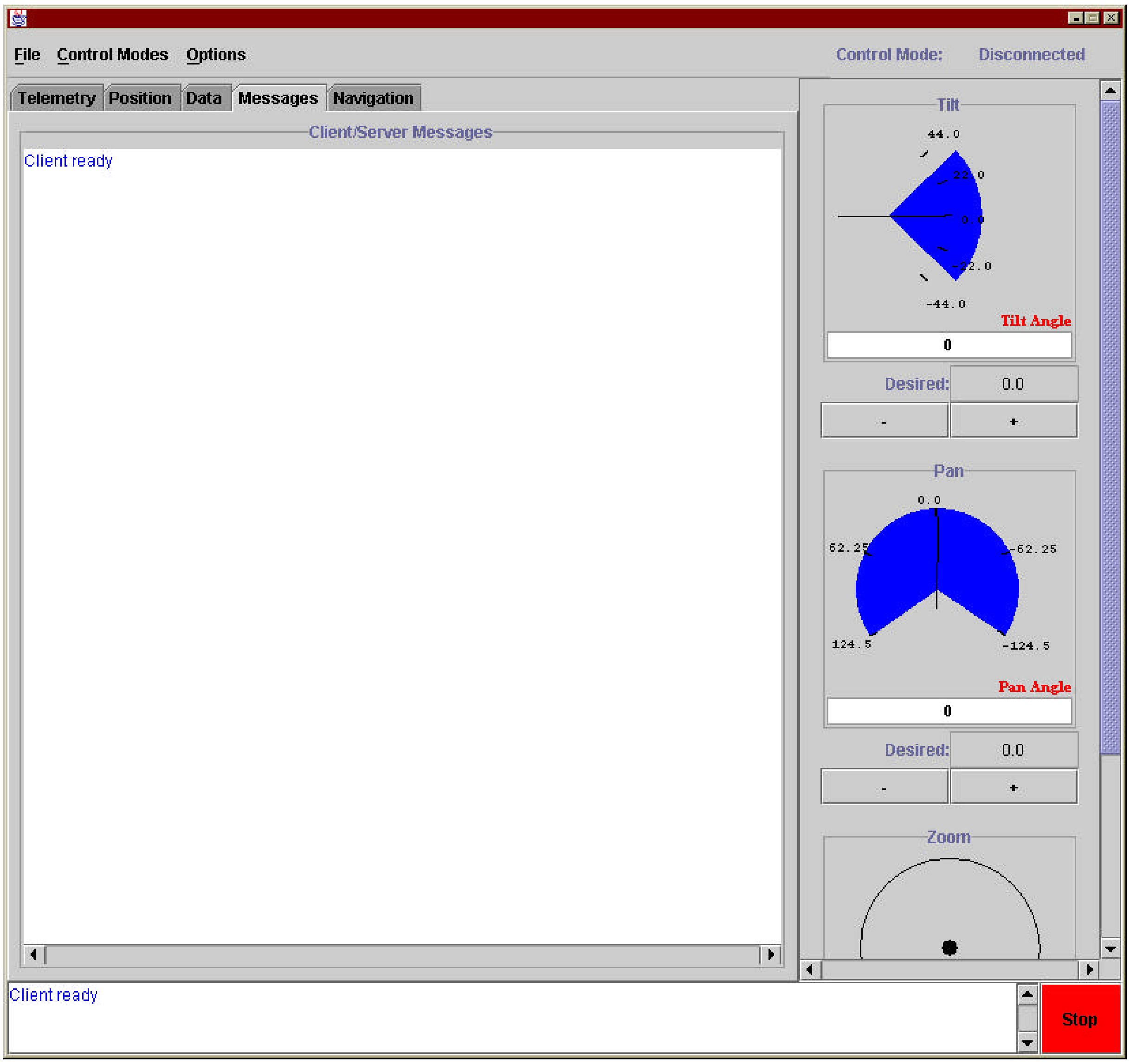
Desired: 0.0

Pan Angle: 0

Desired: 0.0

Client ready

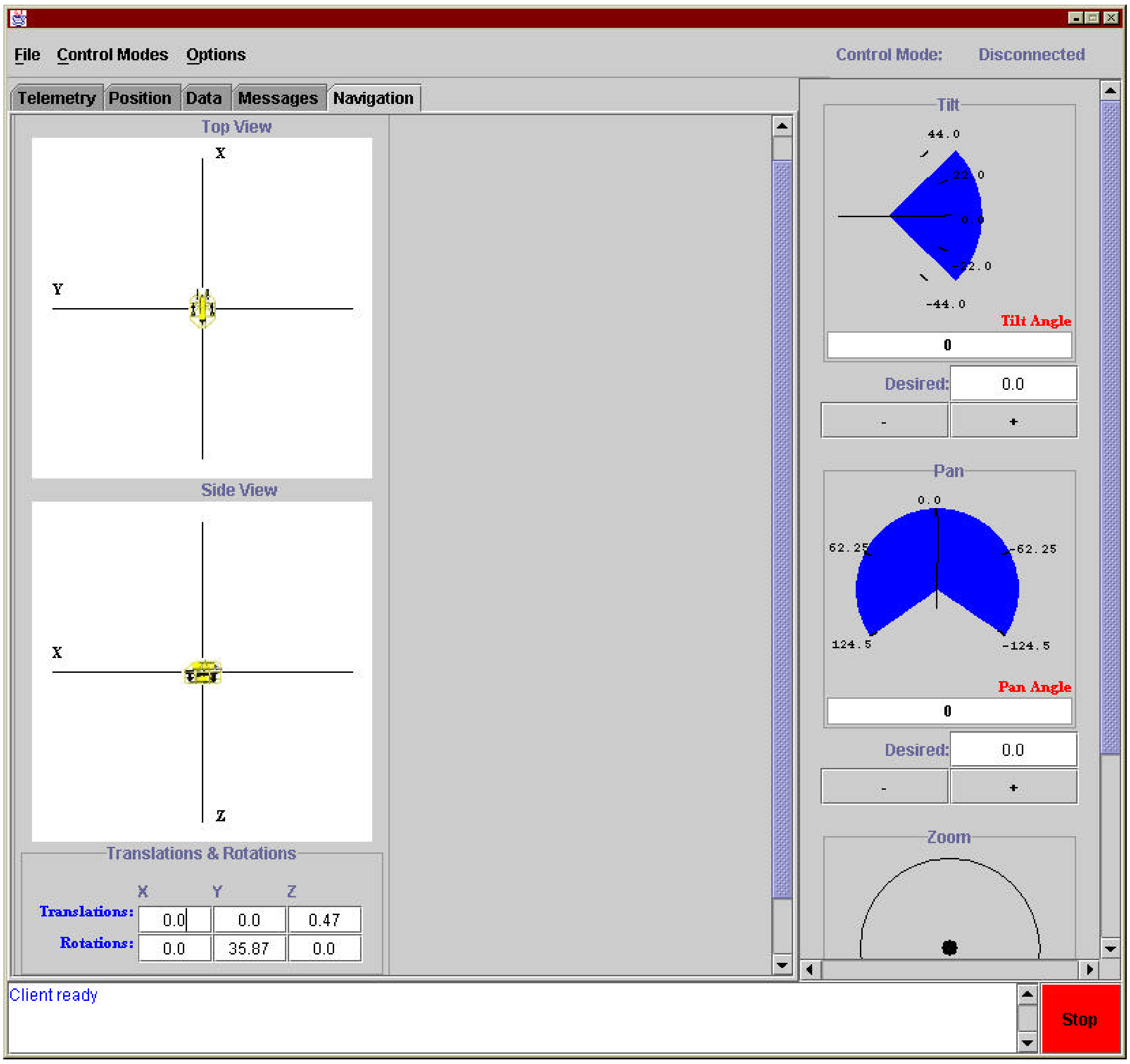
# APPENDIX E4: SCREENSHOT OF MESSAGESPANEL



Client ready

Client ready

# APPENDIX E5: SCREENSHOT OF NAVIGATIONPANEL



Control Mode: Disconnected

Translations & Rotations

| | X | Y | Z |
|---|---|---|---|
| Translations: | 0.0 | 0.0 | 0.47 |
| Rotations: | 0.0 | 35.87 | 0.0 |

Tilt Angle: 0

Desired: 0.0

Pan Angle: 0

Desired: 0.0

Client ready

# APPENDIX F: CONTENTS OF CD

There are four main directories in the CD. These are java3D, software, src and thesis. There is also a file called *bookmarks2.html.* It contains links to all the Internet sites used during the project.

| Directory | Files / Subdirectories | Description |
|---|---|---|
| Java3D | Java3dAPI-documentation | This folder contains the documentation of all the Java3D API 1.1 classes and libraries. |
| | java3d-examples | This folder has a few useful examples relating to use of Java3D. Prominent among these is Interaction, which shows manipulating of mouse behavior. |
| | java3d1_1_3_doc.zip | Zip file for the Java3D API 1.1 Documentation |
| software | Htmlconv12.zip | HTML converter for the Java 2 Plug-In. |
| | ncsaportfolio13beta3.jar | Archive file containing NCSA Portfolio classes. Also contains documentation similar to Java or Java3D documentation. |
| | doc | The documentation for the operator interface is contained in this directory. It is generated using the *javadoc* utility in Java. |
| src | operInterface | Java source code that has been used for the development of the operator interface. There are six prominent directories – **client**, **server**, **menus**, **components**, **control** and **joystick**. |
| | kambara3dq | The *simulator* executable |
| | config.ini | Configuration file for the simulator. |
| | images | All the images used in the thesis are stored in this directory. The images have been further divided in terms of the chapter that they belong to. There are a few backup images and a folder containing some interface snapshots of 1999. |
| thesis | XfigDiagrams | The xfig figures used in creating the images for the thesis are contained in this folder. They include |
| | Thesis.PDF | My thesis document in PDF format. |
| | Thesis.doc | My thesis document in WORD format |
| | Thesis-Outline.txt | An outline of the thesis that was constructed at the beginning of the process of the development of the thesis. |
| | dsw-kambarainfo.txt | David Wettergreen's introduction to the Kambara project. |