IMPLEMENTATION OF A DISTRIBUTED INTERACTIVE SIMULATION INTERFACE IN A SEA KING FLIGHT SIMULATOR

by

2Lt Stuart Peter Rogerson

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science, Graduate Department of Aerospace Science and Engineering, in the University of Toronto

© Copyright by Stuart Peter Rogerson 1997
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
Abstract

The Distributed Interactive Simulation (DIS) Protocol is used to implement a network simulation between a Sea King Simulator and a Canadian Patrol Frigate Landing Safety Officer Simulator. Due to entity contact which was made once the helicopter landed on the deck of the ship, extensive use of smoothing algorithms was required. An investigation into the optimal values for the dead reckoning thresholds and the number of smoothing steps was conducted. The results showed that smoothness of the simulation could be maintained with an acceptable trajectory error.
Acknowledgments

First, I would like to thank Dr. L. D. Reid my supervisor for his support and assistance throughout this project. Thanks to Ralf Keuhnel, Tim Yeung and the cooperative term students at DCIEM for their work on the ship simulator and the final testing. I am also grateful to Wolf Graf for his assistance with the computers in the lab. Thanks also to my fellow students Daniel Sattler, William O’Gorman and Andrew Pierce for their support and assistance. My appreciation also goes out to Paul Dufort for his help with the WGS 84 Coordinate system. And, a very special thanks to my wife Stephanie for her patience, encouragement, and understanding.

I would also like to thank the Department of National Defence for permission to pursue this program. I am also grateful to the Natural Sciences and Engineering Research Council for their financial support.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project Definition</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Literature Search</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Description of the DIS Architecture</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>Definition and concepts of DIS</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>Definition of Terms</td>
<td>7</td>
</tr>
<tr>
<td>3.3</td>
<td>DIS Standards - Units and Coordinate System</td>
<td>9</td>
</tr>
<tr>
<td>3.4</td>
<td>Dead Reckoning Concepts</td>
<td>10</td>
</tr>
<tr>
<td>3.5</td>
<td>PDU Descriptions</td>
<td>11</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Entity Information / Interaction</td>
<td>12</td>
</tr>
<tr>
<td>3.5.1.1</td>
<td>Entity State PDU</td>
<td>12</td>
</tr>
<tr>
<td>3.5.1.2</td>
<td>Collision PDU</td>
<td>14</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Simulation Management</td>
<td>15</td>
</tr>
<tr>
<td>3.5.2.1</td>
<td>Create Entity PDU</td>
<td>15</td>
</tr>
<tr>
<td>3.5.2.2</td>
<td>Remove Entity PDU</td>
<td>15</td>
</tr>
<tr>
<td>3.5.2.3</td>
<td>Start / Resume PDU</td>
<td>16</td>
</tr>
<tr>
<td>3.5.2.4</td>
<td>Stop / Freeze PDU</td>
<td>17</td>
</tr>
<tr>
<td>3.5.2.5</td>
<td>Acknowledge PDU</td>
<td>18</td>
</tr>
<tr>
<td>3.6</td>
<td>Handshaking Requirements</td>
<td>18</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Entity Creation</td>
<td>18</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Entity Removal</td>
<td>19</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Starting, Stopping an Entity</td>
<td>19</td>
</tr>
<tr>
<td>3.7</td>
<td>Communication Requirements</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Simulator Architecture</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Sea King Simulator Architecture</td>
<td>23</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Hardware</td>
<td>23</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Software</td>
<td>24</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Units and Coordinates</td>
<td>26</td>
</tr>
<tr>
<td>4.1.4</td>
<td>New Simulator</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>The Ship Simulators</td>
<td>27</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Configuration One</td>
<td>28</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Configuration Two</td>
<td>28</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Configuration Three</td>
<td>30</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Configuration Four</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>Investigation into Bandwidth Requirement</td>
<td>34</td>
</tr>
</tbody>
</table>
5.2 Dead Reckoning
5.2.1 Selection of Dead Reckoning Model
5.2.2 Selection of DRM Thresholds
5.3 Results

6 Implementation of a Basic DIS Interface
6.1 Communication Algorithms
6.2 Final Comments and Results

7 Smoothing
7.1 Abrupt Motion of Helicopter on Deck
7.2 Isolation of Problem
7.3 Solutions
7.3.1 Lock Helicopter onto Deck
7.3.2 Smoothing
7.3.3 Integration of Acceleration Data
7.3.4 Chosen Solution
7.4 Smoothing Concepts
7.5 Ship Simulator
7.6 Investigation into optimal values for DRM and Smoothing Model
7.7 Final Comments

8 Linking DCIEM and UTIAS
8.1 The Internet
8.2 ISDN Line
8.3 PPP Connection
8.4 Implementation of Link
8.5 Results

9 Implementation of Full DIS Interface
9.1 Simulation Management PDUs and Handshaking
9.2 Time Stamping and Synchronization of System Clocks
9.2.1 Timed Daemon
9.2.2 Timeslave
9.2.3 WAN Synchronization
9.2.4 Transmission Time Compensation
9.3 Coordinate Conversions

10 Evaluation of Full DIS Exercise with Test Ship Simulator

11 Full DIS Exercise Evaluation
11.1 Description of Full DIS Exercise
11.2 Description of Evaluation Runs
11.3 Problems Encountered and Limitations of Evaluation
References  

A. Derivation of Smoothing Equations
B. Conversion from GenHel Inertial Frame to an Earth Fixed Frame which Approximates WGS 84
3.1. Entity State PDU
3.2. Collision PDU
3.3. Create Entity PDU
3.4. Remove Entity PDU
3.5. Start / Resume PDU
3.6. Stop / Freeze PDU
3.7. Acknowledge PDU

5.1. Dead Reckoning Parameters
5.2. Selected Dead reckoning Thresholds

7.1. Updated Dead Reckoning Thresholds
7.2. Modified portion of the Entity State PDU

11.1. Final Evaluation Runs - Varied Parameters
11.2. Helicopter Initial Conditions Before and After Trimming
11.3. Trajectory Error Results
11.4. Trajectory Error at Touchdown

13.1 Selected Dead Reckoning Thresholds
13.2 Trajectory Error for Selected Parameters
3.1. DIS Earth Fixed Frame  
3.2. DIS Body Fixed Frame  
3.3. Entity Creation  
3.4. Entity Removal  
3.5. Starting, Stopping an Entity  

4.1. External view of UTIAS simulator  
4.2. Internal view of cockpit  
4.3. Layout of Connections between Sea King Simulator Computers  
4.4. Visual Display on IRIS5  
4.5. Helicopter and Ship Simulator Configurations  

5.1. Packets per Second for Typical Flight Path with Thresholds from Table 5.2  

7.1. Simulation time step size  
7.2. Time step size of Dedicated Real Time Program  
7.3. Smoothing Concept Diagram  
7.4. Filtering Aspects of Integration scheme  
7.5. Acceleration Resulting from Inputting a Half Foot Jump into the DRM  
7.6. Comparison of Acceleration response for Linear, Cubic and Fifth Order Smoothing Algorithms  
7.7. Maximum acceleration Recorded During Run  
7.8. Average acceleration Recorded During Run  
7.9. Maximum difference Recorded During Run  
7.10. Average Difference Recorded During Run  
7.11. Zoom of Figure 7.7  

8.1. Ping statistics between IRIS5 and SIM2 via Internet  
8.2. Ping statistics between RISC1 and SIM2 using PPP Link  

9.1 Synchronization of IRIS5 and RISC1 using timed. Sample rate of once an hour  
9.2 Synchronization of IRIS5 and RISC1 using timed. Sample rate of once every 10 seconds  
9.3 Timeslave data between HAL and SIM6  
9.4 Offset Determination Algorithm  
9.4. Centre of Gravity / Centre of Bounding Volume Vector diagram  

10.1. Time history of Large Heave excursion For Helicopter's Simulators DRM
11.1 Ship X Acceleration 101
11.2 Ship Y Acceleration 101
11.3 Ship Z Acceleration 102
11.4 Ship Phi Angular Acceleration 102
11.5 Ship Theta Angular Acceleration 103
11.6 Ship Psi Angular Acceleration 103
11.7 Comparison of Ship Motion : X 104
11.8 Comparison of Ship Motion : Y 104
11.9 Comparison of Ship Motion : Z 105
11.10 Comparison of Ship Motion : \( \phi \) 105
11.11 Comparison of Ship Motion : \( \theta \) 106
11.12 Comparison of Ship Motion : \( \psi \) 106
11.13 Comparison of Helicopter Motion : X 107
11.14 Comparison of Helicopter Motion : Y 107
11.15 Comparison of Helicopter Motion : Z 108
11.16 Comparison of Helicopter Motion : \( \phi \) 108
11.17 Comparison of Helicopter Motion : \( \theta \) 109
11.18 Comparison of Helicopter Motion : \( \psi \) 109
11.19 Time History Comparison of \( R_x \), Run # 2 110
11.20 Time History Comparison of \( R_y \), Run # 2 110
11.21 Time History Comparison of \( R_x \), Run # 2 111
11.22 Time History Comparison of \( R_y \), Run # 4 111
11.23 Time History Comparison of \( R_y \), Run # 4 112
11.24 Time History Comparison of \( R_z \), Run # 4 112
11.25 Trajectory Error - Maximums, Sea State 5, \( \Delta \text{position} = 0.01 \) ft 113
11.26 Trajectory Error - Averages, Sea State 5, \( \Delta \text{position} = 0.01 \) ft 113
11.27 Trajectory Error - Maximums, Sea State 1, \( \Delta \text{position} = 0.01 \) ft 114
11.28 Trajectory Error - Averages, Sea State 1, \( \Delta \text{position} = 0.01 \) ft 114
11.29 Trajectory Error - Maximums, Sea State 5, \( \Delta \text{position} = 1.00 \) ft 115
11.30 Trajectory Error - Averages, Sea State 5, \( \Delta \text{position} = 1.00 \) ft 115
11.31 Trajectory Error - Maximums, Sea State 1, \( \Delta \text{position} = 1.00 \) ft 116
11.32 Trajectory Error - Averages, Sea State 1 \( \Delta \text{position} = 1.00 \) ft 116
11.33 Trajectory Error - Maximums, Sea State 5, \( p = 10 \) 117
11.34 Trajectory Error - Averages, Sea State 5, \( p = 10 \) 117
11.35 Trajectory Error - Maximums, Sea State 1, \( p = 10 \) 118
11.36 Trajectory Error - Averages, Sea State 1, \( p = 10 \) 118
11.37 Trajectory Error - Maximums, Sea State 5, \( p = 72 \) 119
11.38 Trajectory Error - Averages, Sea State 5, \( p = 72 \) 119
11.39 Trajectory Error - Maximums, Sea State 1, \( p = 72 \) 120
11.40 Trajectory Error - Averages, Sea State 1, \( p = 72 \) 120
5.1 Dead Reckoning Model - RVW  
7.1 Predictor Equation  
7.2 Linear Smoothing Equation  
7.3 Cubic Smoothing Equation  
7.4 Fifth Order Smoothing Equation  
7.5 Method of Calculating Acceleration from Output of DRM  
9.1 Centre of Gravity to Centre of Bounding Volume Coordinate Transformation  
9.2 Centre of Bounding Volume to Centre of Gravity Coordinate Transformation
Chapter 1

Project Definition

The Flight Simulation Group at the University of Toronto Institute for Aerospace Studies (UTIAS) has developed a Sea King Flight Simulator for landing on the deck of the Canadian Patrol Frigate (CPF). The Defence and Civil Institute of Environmental Medicine (DCIEM) is in the process of developing a Landing Safety Officer (LSO) simulator. The LSO assists the pilot when landing on the deck of the ship. DCIEM has expressed an interest in linking the LSO simulator with the helicopter simulator at UTIAS so that the LSO and pilot can interact in a virtual environment. The goal of this thesis is to create an interface for the helicopter simulator to allow it to be networked with the LSO simulator at DCIEM.

As a result of budget cuts, the Department of National Defense is looking for ways to provide less expensive training. The use of simulators is an increasingly popular method of providing realistic training at a reduced cost. The interconnection of simulators increases the fidelity of the exercise by allowing humans to interact with one another in a virtual environment. This interaction improves the accuracy of the training (Loral, 1994). Unfortunately, in Canada, the networking of simulators is essentially non-existent. It is hoped that through this work some of the issues related to the linking of simulators will be addressed, thus laying the groundwork for future simulation networking in Canada.

The interaction required between the two simulators for deck landing operations is as follows. Since the pilot sits in the right seat of the cockpit, the helicopter usually
otherwise. The ship travels at approximately 10 knots steering a course that creates a relative wind of about 10 degrees to the bow off the port side. The approach is made at an ideal altitude of 45 feet above sea level.

The helicopter then makes a transition to the starboard side of the ship in order to connect the haul down cable which provides tension to assist the pilot to remain central over the deck. This cable is part of the RAST (Recover Assist Secure Traverse). First, a small cable is lowered from the helicopter; a technician on the deck attaches the haul down cable to this line which is then pulled back into the helicopter and locked into place. The helicopter is now connected to the ship via this cable.

Once the cable is in place, the pilot maneuvers the helicopter to a central position over the deck. This is defined by the Bear Trap. The Bear Trap has a jaw which closes on a probe that is attached to the base of the Sea King. This system locks the helicopter down on to the deck as soon as touchdown occurs. The Bear Trap is approximately 1 m².

The LSO, standing in a booth on the starboard side of the deck, assists by applying approximately 1500 lb. of cable tension. As well, the LSO provides the pilot with information as to his position over the deck. Finally, using a combination of his sea legs and the instruments in front of him, the LSO judges when there will be a quiescent period in the ship’s motion. When the conditions are right, the LSO tells the pilot to adopt a high hover. The helicopter then drops in altitude to approximately 10 feet above the deck. Assuming the quiescent period is still there, the LSO instructs the pilot to adopt a low hover of about 3 feet above the deck. In this position, the LSO provides final position corrections so the probe is lined up with the Bear Trap, then the command “Down,
pilot drops the collective and the helicopter lands on the deck. At the exact moment that touchdown occurs the Bear Trap clamps around the probe locking the aircraft down. The deck landing is complete.

For this research project, the goal is to implement the visual section of the deck landing operation only. That is, the LSO will be able to assist the pilot in the landing by providing direction information only. The use of the haul down cable will not be implemented at this time.
Chapter 2

Literature Search

The initial literature search on the networking of simulators led to a standard referred to as SIMNET (SIMulation NETworking). The development of SIMNET commenced in 1983 under funding from the Defense Advanced Research Projects Agency (DARPA). The goal was to provide a method of linking simulators to create a virtual battlefield. As SIMNET was advanced a newer standard was created called Distributed Interactive Simulation (DIS). The main principles of DIS are as follows (IEEE, 1996):

i. A central computer does not control the entire simulation exercise. There isn’t a main computer which determines how each entity affects others in the exercise.

ii. Each simulator is responsible for maintaining a model of at least one entity. This model must react to changes in the environment caused by either internal inputs or those from other simulators in the exercise. In turn, the simulator is responsible for notifying other simulators of observable changes to its own model or models.

iii. A standard protocol is used to disseminate information about position, orientation, and appearance.

iv. Only the receiving simulator can decide how the events in the exercise affect its own model or models.
Canada, the only organization to use the concepts of DIS is DCIEM. DCIEM has developed a ship simulator which allows Naval Officers to practice formation maneuvers with other ships. As part of this project, three simulators were linked together using a partially modified and much reduced version of the DIS standard. The algorithms which implemented the DIS standard for the ship simulator were contained in software developed by the Computer Science Department at the Naval Postgraduate School in Monterey, California. This software is referred to as Naval Postgraduate School NETworking (NPSNET). NPSNET is a complete DIS library developed at the School and available for use by other institutions. Unfortunately, since this software was written in C++, the author was unable to directly use the majority of the source code for the work at UTIAS due to the lack of a C++ compiler. However, this code did provide some useful insights into the development of a DIS interface for the helicopter simulator. In the end, the only file used was a modified version of NPSNET’s pdu.h which defines the packets that are transmitted as part of the DIS standard. These packets will be discussed in Chapter 3.

In the implementation of the DIS protocol, the primary source for the standard is an IEEE Standards Document (IEEE, 1996). It covers all aspects of the DIS protocol, from packet structures to handshaking requirements. The main concepts are discussed in detail in the next chapter.

The final area of information for the DIS interface was the Internet. Various organizations such as the Institute for Simulation and Training (IST), NPSNET, and IEEE
It should be noted that DIS is constantly undergoing changes. At the moment, a new standard called High Level Architecture (HLA) is being developed. HLA is not an upgrade from DIS, but a redesign of the architecture to provide increased flexibility and performance. However, since HLA is still under development, DIS was used for this project.
Chapter 3
Description of the DIS Architecture

3.1 Definition and Concepts of DIS.

The IEEE defines Distributed Interactive Simulation as

“A time and space coherent synthetic representation of world environments designed for linking the interactive, free play activities of people in operational exercises. The synthetic environment is created through real-time exchange of data units between distributed, computationally autonomous simulation applications in the form of simulations, simulators, and instrumented equipment interconnected through standard computer communicative services. The computational simulation entities may be present in one location or may be distributed geographically.”

DIS is the standard for information exchange between the two Simulation Applications. It defines what information is communicated and how that information is transmitted. It outlines handshaking requirements between the simulators, and defines the units and coordinate system of the data. In the following sections, the key elements of DIS required for this specific application are listed.

3.2 Definition of Terms

A knowledge of the following terminology will be required in order to understand this document. These definitions are taken either in part or completely from the IEEE Standard (1996).

*Simulation Application* - The executing software on a host computer that models all or part of the representation of one or more simulation entities. The simulation
experimentation. For this project, there are two simulation applications; the ship simulator at DCIEM and the helicopter simulator at UTIAS.

**Simulation Manager** - The process that provides centralized control of the simulation exercise. These functions include starting, stopping, and shutdown of the exercise. The Simulation Manager also collects and redistributes certain types of data.

**Entity** - an element of the synthetic environment that is created and controlled by a simulation application and effected by the exchange of DIS PDUs. There are two entities in this application: the Ship and the Helicopter.

**Protocol Data Unit (PDU)** - A DIS data message that is passed on a network between simulation applications according to a defined protocol.

**Articulated Part** - A visible part of a simulated entity that is attached to the entity, but can move about a fixed point. An example would be the haul down cable used to assist in centering the helicopter on the RAST Bear Trap.

**DIS Exercise** - An exercise that consists of one or more interacting simulation applications. In this case, DIS exercise refers to the virtual environment created by the two simulators interacting with one another.
All data that is transmitted between the Simulation Applications uses SI, Le Systéme International d’Unités. Consequently all position information is in metres, and all angles are in radians. Time is represented by the number of DIS time units past the hour. A DIS time unit is approximately equal to 1.676 μs. This is derived from the following. The DIS protocol uses an unsigned integer to represent time, so one hour is equal to $2^{31}-1$. Thus the time unit is $3600s / (2^{31}-1)$.

The Earth-fixed coordinate system used in the DIS synthetic environment is WGS 84. WGS 84 is a right-handed geocentric Cartesian coordinate system defined by the Defense Mapping Agency Technical Report 8350.2. The origin of the system is the centre of the Earth. The Z axis points through the North Pole and the X axis passes through the prime meridian along the Equator. Since the system is right-handed the Y axis passes through $90^\circ$ East of longitude on the equator (Figure 3.1). In order to relate the position and orientation of an entity to the Earth fixed frame, it is necessary to define a body fixed frame. DIS again uses a right-handed Cartesian coordinate system. The centre of the body fixed frame is the centre of the bounding volume of the entity. That is the geometric centre of the entity’s volume when all articulated and attached parts are ignored. The x axis passes through a point at the centre of the front of the entity. The y axis passes through a point on the right side and the z axis points directly out the bottom of the entity. See Figure 3.2. The orientation of the entity is described using the Euler angles, $\phi$, $\theta$, $\Psi$ which describe the rotations necessary to change from the Earth Fixed frame to the body fixed frame.
The DIS architecture employs a concept called dead reckoning to reduce the bandwidth required for the transmission of position and orientation data between simulators. Rather than send an update every time an entity moves, the DIS standard requires that simple models are kept of all entities within the simulation. These simple models are then used to predict the entity’s position and orientation so it is not necessary to transmit a movement each time it occurs.

For example, in this case there are two entities; the ship and the helicopter. The helicopter simulator has a complicated model of itself and a simple model of the ship. The ship simulator has the opposite, a complicated model of the ship and a simple model of the helicopter. If no information is received from the other simulator, the simple model is used to predict the position of the opposing entity in the simulation.

In order to determine when an update should be sent, the simulation application must also have a simple model of itself. Every time step, the simple model and the complicated model are compared. When the two differ in position or orientation by a predetermined threshold, it is necessary to send an update. The values $\Delta$position and $\Delta$angle represent these thresholds. $\Delta$position is the maximum allowable difference in feet between the predicted and actual position on either the x, y, or z axis in the GenHel inertial frame (defined in Chapter 4). $\Delta$angle is the maximum allowable angular difference in degrees between the predicted and actual orientation for the helicopter Euler angles $\phi$, $\theta$, or $\psi$ in the GenHel inertial frame. Continuing this example, the helicopter simulator has a simple model of the helicopter. Each time step, the two models are compared. If, for example, they differ by a metre, the simple helicopter model is updated to the same
the ship simulator. The ship simulator, upon receipt of this new information, updates its simple helicopter model and then uses the updated model to predict the helicopter’s position and orientation until another packet arrives.

In order for the dead reckoning to function, it is necessary to send more than just position and orientation data. The most complicated model defined by DIS requires linear velocity, linear acceleration, and angular velocity. The different models are listed in IEEE Standard (1996), Annex B.

3.5 PDU Descriptions

The DIS standard defines 27 different PDUs which can be used to create and manage the synthetic environment. These packets cover a very wide range of possible events in a DIS exercise. For example, it is possible to control the firing of a weapon, resupply of a vehicle, electronic warfare, or just shutting down the simulation. These 27 PDUs are broken down into six different categories:

i. Entity Information / Interaction,

ii. Warfare,

iii. Logistics,

iv. Simulation Management,

v. Distributed Emission Regeneration,

vi. Radio Communications.

For this research project, only two categories are relevant - Entity Information / Interaction and Simulation Management.
header. The header contains critical information such as the time the data contained within the PDU was valid, and the type of PDU that it is. Also contained within each header is information regarding unique identification of the entity. This identification remained unused in this application since only two entities are involved in the exercise.

One noted deficiency of the DIS protocol relates to the PDUs. The data structures defined by the standard are quite complex and convey a lot more information than required for this application. This extra information increases the size of the packets and accordingly requires a larger bandwidth between simulators for communication. It is felt by the author that this is a waste of limited system resources. However, this area of concern has been addressed by HLA.

3.5.1 Entity Information/Interaction

3.5.1.1 Entity State PDU

This is the most important PDU in the DIS standard. It contains the position, orientation, velocity, and acceleration of the entity, as well as information on the appearance, entity type, and the articulated parts. The primary reason for transmitting this PDU is to update the dead reckoning algorithms. A secondary reason for issuing an Entity State PDU is to indicate a time-out is exceeded. The DIS standard requires that a PDU be issued on a regular interval to show that the entity still exists in the exercise. The defined structure of the Entity State PDU is shown in Table 3.1. Upon receipt of an Entity State PDU, the receiving simulation application will update its dead reckoning algorithms so that they agree with the new data.
<table>
<thead>
<tr>
<th>PDU Header</th>
<th>Protocol Version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exercise ID</td>
</tr>
<tr>
<td></td>
<td>PDU Type</td>
</tr>
<tr>
<td></td>
<td>Protocol Family</td>
</tr>
<tr>
<td></td>
<td>Timestamp</td>
</tr>
<tr>
<td></td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td>Padding</td>
</tr>
<tr>
<td>Entity ID</td>
<td>Site</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td>Force ID</td>
<td></td>
</tr>
<tr>
<td># of Articulation Parameters</td>
<td></td>
</tr>
<tr>
<td>Entity Type</td>
<td>Entity Kind</td>
</tr>
<tr>
<td></td>
<td>Domain</td>
</tr>
<tr>
<td></td>
<td>Country</td>
</tr>
<tr>
<td></td>
<td>Category</td>
</tr>
<tr>
<td></td>
<td>Subcategory</td>
</tr>
<tr>
<td></td>
<td>Specific</td>
</tr>
<tr>
<td></td>
<td>Extra</td>
</tr>
<tr>
<td>Alternate Entity Type</td>
<td>Entity Kind</td>
</tr>
<tr>
<td></td>
<td>Domain</td>
</tr>
<tr>
<td></td>
<td>Country</td>
</tr>
<tr>
<td></td>
<td>Category</td>
</tr>
<tr>
<td></td>
<td>Subcategory</td>
</tr>
<tr>
<td></td>
<td>Specific</td>
</tr>
<tr>
<td></td>
<td>Extra</td>
</tr>
<tr>
<td>Entity Linear Velocity</td>
<td>X - Component</td>
</tr>
<tr>
<td></td>
<td>Y - Component</td>
</tr>
<tr>
<td></td>
<td>Z - Component</td>
</tr>
<tr>
<td>Entity Location</td>
<td>X - Component</td>
</tr>
<tr>
<td></td>
<td>Y - Component</td>
</tr>
<tr>
<td></td>
<td>Z - Component</td>
</tr>
<tr>
<td>Entity Orientation</td>
<td>Psi</td>
</tr>
<tr>
<td></td>
<td>Theta</td>
</tr>
<tr>
<td></td>
<td>Phi</td>
</tr>
<tr>
<td>Entity Appearance</td>
<td>Dead Reckoning Algorithm</td>
</tr>
<tr>
<td>Dead Reckoning Parameters</td>
<td>Other Parameters</td>
</tr>
<tr>
<td></td>
<td>Entity Linear Acceleration</td>
</tr>
<tr>
<td></td>
<td>Entity Angular Velocity</td>
</tr>
<tr>
<td>Entity Markings</td>
<td>Character Set</td>
</tr>
<tr>
<td>Capabilities</td>
<td>Parameter Type Designator</td>
</tr>
<tr>
<td>Articulation Parameters</td>
<td>Change</td>
</tr>
<tr>
<td></td>
<td>ID - Part Attached To</td>
</tr>
<tr>
<td></td>
<td>Parameter Type</td>
</tr>
<tr>
<td></td>
<td>Parameter Value</td>
</tr>
</tbody>
</table>

Table 3.1: Entity State PDU
Along with the standard identification information required in a PDU, a Collision PDU contains information about which entity the issuing entity collided with. Also, data is included which can be used to determine the extent of damage created in the collision. This PDU is issued whenever a collision occurs between two entities. The structure of this packet is defined in Table 3.2. Upon receipt of a Collision PDU, the receiving simulation application will transmit its own Collision PDU stating which entity collided with it. If the collision destroyed the receiving entity, the receiving application will remove itself from the exercise.

<table>
<thead>
<tr>
<th>PDU Header</th>
<th>Protocol Version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exercise ID</td>
</tr>
<tr>
<td></td>
<td>PDU Type</td>
</tr>
<tr>
<td></td>
<td>Protocol Family</td>
</tr>
<tr>
<td></td>
<td>Timestamp</td>
</tr>
<tr>
<td></td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td>Padding</td>
</tr>
<tr>
<td>Issuing Entity ID</td>
<td>Site</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td>Colliding Entity ID</td>
<td>Site</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td>Event ID</td>
<td>Site</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Event Number</td>
</tr>
<tr>
<td>Collision Type</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td></td>
</tr>
<tr>
<td>Velocity</td>
<td>X - Component</td>
</tr>
<tr>
<td></td>
<td>Y - Component</td>
</tr>
<tr>
<td></td>
<td>Z - Component</td>
</tr>
<tr>
<td>Mass</td>
<td>x - component</td>
</tr>
<tr>
<td>Location (with respect to other Entity)</td>
<td>y - component</td>
</tr>
<tr>
<td></td>
<td>z - component</td>
</tr>
</tbody>
</table>

Table 3.2. Collision PDU
There are a total of 12 simulation management PDUs. Simulation management PDUs control the DIS exercise. For this project, the simplest method of control between simulation applications was employed, so only five were required. Handshaking will be discussed in detail later in this chapter.

3.5.2.1 Create Entity PDU

This PDU is issued when a simulation application wants to join a DIS exercise. Upon receipt of a Create Entity PDU, the receiving application returns an Acknowledge PDU. The structure of a Create Entity PDU is shown in Table 3.3.

<table>
<thead>
<tr>
<th>PDU Header</th>
<th>Protocol Version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exercise ID</td>
</tr>
<tr>
<td></td>
<td>PDU Type</td>
</tr>
<tr>
<td></td>
<td>Protocol Family</td>
</tr>
<tr>
<td></td>
<td>Timestamp</td>
</tr>
<tr>
<td></td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td>Padding</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Originating Entity ID</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Entity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Receiving Entity ID</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Site</td>
</tr>
</tbody>
</table>

| Request ID |          |

Table 3.3. : Create Entity PDU

3.5.2.2 Remove Entity PDU

This PDU is issued when a simulation application wants to remove itself from a DIS exercise. Upon receipt of this PDU, an Acknowledge PDU is returned. The structure is shown in Table 3.4.
3.5.2.3 Start/Resume PDU

This PDU is issued when a simulation application wants to go from a stopped or frozen state to a running state. Upon receipt, an Acknowledge PDU is returned. The structure is defined in Table 3.5. The field *simulation time* states when the PDU is to transition from a stopped state to a running state. This time should be in the future so that the receiving application has time to process the request.
3.3.2.4 Stop/Freeze PDU

This PDU is issued when a simulation application wants to go from a running state to a stopped or frozen state. Upon receipt, an Acknowledge PDU is returned. The structure is defined in Table 3.6. The PDU should contain information regarding the reason the entity is leaving and the time that this is to occur. As in the Start/Resume PDU, the time should be in the future so there is time for the receiving application to process the request.

<table>
<thead>
<tr>
<th>PDU Header</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol Version</td>
<td></td>
</tr>
<tr>
<td>Exercise ID</td>
<td></td>
</tr>
<tr>
<td>PDU Type</td>
<td></td>
</tr>
<tr>
<td>Protocol Family</td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td></td>
</tr>
<tr>
<td>Originating Entity ID</td>
<td>Site</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td>Receiving Entity ID</td>
<td>Site</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Site</td>
</tr>
<tr>
<td>Real World Time</td>
<td>Hour</td>
</tr>
<tr>
<td></td>
<td>Time Past Hour</td>
</tr>
<tr>
<td>Reason</td>
<td></td>
</tr>
<tr>
<td>Frozen Behavior</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td></td>
</tr>
<tr>
<td>Request ID</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.6. : Stop / Freeze PDU
3.5.2.5 Acknowledge PDU

This PDU is sent in response to the arrival of the above Simulation management PDUs. It’s structure is shown in Table 3.7.

<table>
<thead>
<tr>
<th>PDU Header</th>
<th>Protocol Version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exercise ID</td>
</tr>
<tr>
<td></td>
<td>PDU Type</td>
</tr>
<tr>
<td></td>
<td>Protocol Family</td>
</tr>
<tr>
<td></td>
<td>Timestamp</td>
</tr>
<tr>
<td></td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td>Padding</td>
</tr>
<tr>
<td>Originating Entity ID</td>
<td>Site</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td>Receiving Entity ID</td>
<td>Site</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Site</td>
</tr>
<tr>
<td>Acknowledge Flag</td>
<td></td>
</tr>
<tr>
<td>Response Flag</td>
<td></td>
</tr>
<tr>
<td>Request ID</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.7. : Acknowledge PDU

3.6 Handshaking Requirements

The DIS protocol has defined rules for the exchange of simulation management PDUs to allow the smooth operation of entities in a DIS exercise. These rules cover a wide range of subjects such as resupplying a vehicle and the removal of an entity from a simulation. For this particular application, the areas of interest are Entity Creation, Entity Removal, and Starting and Stopping an Entity. The communication occurs between the Simulation Manager (SM) and the relevant Simulation Application (SA).

3.6.1 Entity Creation

There are three different ways to create a new entity in a DIS exercise. The first requires data to be requested about the new entity by the SM and for the initial position, orientation and appearance to be returned. The second method involves the transmission
when the initial conditions are pre-arranged before the DIS exercise is started (Figure 3.3).

3.6.2 Entity Removal

As shown in Figure 3.4, the removal of an entity is very simple, requiring only the transmission of a Remove Entity PDU followed by an Acknowledge PDU. The DIS standard also requires that a final Entity State PDU be issued by the SA after removal with the appearance field stating the entity is deactivated.

3.6.3 Starting, Stopping an Entity

The method of starting or stopping an entity is shown in Figure 3.5. Again, the handshaking is quite simple; to start an entity a Start/Resume PDU is transmitted followed by an Acknowledge PDU. Likewise, to stop an entity a Stop/Freeze PDU is sent, with an Acknowledge PDU returned.

3.7 Communication Requirements

The method used by the DIS standard to send data back and forth between simulation applications is the Ethernet. In a DIS exercise there are two primary types of data that are transmitted; control data, and simulation data. With simulation data the speed of transmission is the most important factor, since if a packet is lost enroute it is very likely that another packet will be sent shortly after. However, with control data guaranteed arrival is more important than transmission time. To satisfy these requirements, DIS makes use of two elements in the Internet protocol - Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).
TCP is a reliable method of transmitting packets from one computer to another. If a packet fails to arrive, or arrives incorrectly, the packet is retransmitted. In order for this to function, the receiving computer is required to send an acknowledge (not an Acknowledge PDU) for each received packet. This increases the amount of time required to send data from one computer to another. Thus, TCP is best suited for control data in a DIS exercise. Therefore all Simulation Management PDUs should be transmitted using TCP.

UDP is a faster method of transmission between computers since there is no requirement for an acknowledgment to be sent. However, if a packet is lost or corrupted enroute to the computer, no effort is made to resend the data. It is regarded as lost. Thus, UDP is suited for simulation data, that is Entity State PDUs.
Figure 3.1: DIS Earth Fixed Frame

Figure 3.2: DIS Body Fixed Frame
Figure 3.3: Entity Creation

Figure 3.4: Entity Removal

Figure 3.5: Starting, Stopping an Entity
Chapter 4

Simulator Architecture

4.1 Sea King Simulator Architecture

4.1.1 Hardware

The simulation facilities located in the Flight Simulation Laboratory at UTIAS consist of a DC-8 cockpit mounted on a six degree-of-freedom motion base (Figure 4.1). The rear of the DC-8 cab has been modified to accommodate a helicopter cockpit. This cockpit consists of a visual display unit, electronic flight instrument system (EFIS), centre stick, rudder pedals, collective and a seat (Figure 4.2). The cockpit also contains various speakers for the digital sound system.

The simulator consists of six computers each assigned a specific task in the simulation (Figure 4.3). The main flight equations are run on a RISC 6000 series IBM computer (RISC1). This is the main engine of the simulation. RISC1 is linked via the Ethernet to the Perkin-Elmer (PE) computer which acts as an interface between RISC1 and various Input / Output (I/O) channels. The primary output channel is the Single Board Computer (SBC). The SBC sends data to the motion base and the sound system. The input channels to the PE are the two sets of control inputs from the cockpit. The first is the collective, rudder pedals and numerous switches. The second input is the MacFadden Control Loader. This system, running on a 486 PC, controls the centre stick. The MacFadden provides hydraulic actuated force feedback for pilot control inputs. Since
the MacAdem control loader requires data on the present state of the aircraft to provide the correct pilot feedback, this channel is bi-directional.

The two remaining computers are the EFIS and the visual display. The EFIS runs on a Silicon Graphics (SGI) series 3130 workstation (IRIS2) and displays the necessary instruments on a monitor in the cockpit. The visual display system is an SGI 4D/310 workstation (IRIS5). The output from this system is sent to a monitor attached to the outside of the simulator cab. Figure 4.4 is an example of the display showing the rear of the frigate. This image is seen by the pilot through an optical collimating device which provides the perception of depth to the image. RISC1 communicates with IRIS2 and IRIS5 via the local Ethernet in the lab. IRIS5, is also connected to the PE via a direct serial line. Through this link and subsequent connection between the PE and RISC1 the timing for the simulator is controlled. That is, the rate at which IRIS5 updates the visual display, sets the timing for the entire simulation. The update rate is currently 30 Hz.

4.1.2 Software

The flight equations for the Sea King are implemented using a program called General Helicopter Flight Dynamics Simulation (GenHel). Coordination of the flight equations is provided by the function main in the file bhawk_pe_exec.c. This module synchronizes the flight equations with the required I/O functions and provides ship motion for the deck landing task. This ship motion is available independently of the DIS interface. However, the ship trajectory can only follow the available pre-recorded input files.

There are three major steps to the bhawk_pe_exec.c program. The first step is to initialize the system. The network communication algorithms are set up, then the input file
nature of this file will be dealt with later in this thesis. Step two is trimming the flight equations for the initial conditions. With this complete, packets are sent to the EFIS and visual display to show the start conditions to the pilot. The third and final step is running the flight equations in real time.

The first stage of the real time component is reading the pilot control inputs from the PE. Next, certain crash conditions are checked to see if the pilot has exceeded the safe limits of helicopter operation. The conditions are based on altitude, vertical rate of descent and the Euler angles. These checks are critical when running the simulator with the motion base in order to avoid damaging the hydraulic jacks. Following this, a call is made to the function `bhawk_nrt_exec`. This is the main module for the flight equations. This portion of code calls all the other modules related to the flight equations of the Sea King. One iteration is made of these equations then the results are sent to the EFIS, Visual Display, and the PE for the motion and sound. Immediately after this two more iterations are made of the flight equations. Then the whole real time block repeats itself.

It is necessary at this point to clarify why the main program follows the above order. First, the flight equations are run three times for increased accuracy and smoothness. The use of a smaller time step brings these benefits. This is possible because the flight equations take around nine milliseconds to execute, whereas IRIS5 can only update the image on the screen every thirty-three milliseconds. Finally, the flight equations are run once and then the updates are sent to the EFIS and visual display to minimize simulator time delay. By sending the data out earlier, IRIS5 has more time to
A final note on the flight equations, is that there are two areas that require the ship's position. The first is the visual position. This is updated by sending data calculated from the ship motion file with every packet to IRIS5. This applies to Configuration One of Section 4.2.1. When IRIS5 receives this data, it adjusts the visual display based on the position of the helicopter and ship. The second area is the deck position. This is required internally to the flight equations since the helicopter model has landing gear. Since the flight equations are run three times per screen update, it is necessary to interpolate between the screen ship positions. These values are calculated through a call to the function update. The present interpolation scheme is linear.

4.1.3 Units and Coordinates

The GenHel model uses imperial units for all of its calculations. Therefore, the helicopter data is in feet, feet per second, or feet per second per second and the angles are in degrees. The ship motion that is destined for IRIS5 is in feet and degrees. However, the deck position needs to be in feet and radians. Also of importance is that the deck equations need heave which is opposite in sign to the z vector used in a traditional right handed coordinate system.

GenHel assumes a flat Earth for the inertial frame. The origin of the system is arbitrary, but it is assumed to be the start position of the ship at sea level. The x axis points north from the origin, the y axis points to the east, and the z axis points directly down. The origin of the GenHel frame can be assigned a specific longitude and latitude in
Appendix B). The body fixed frame is the same as the DIS body fixed frame except the origin of the GenHel system is the centre of gravity, not the centre of the bounding volume.

4.1.4 New Simulator

The flight simulation facilities at UTIAS are presently undergoing a transition from the present system described in this chapter to a newer system. The most significant improvement will be a far superior image display. The main engine for the flight equations has also been upgraded to a newer RISC 6000 series computer. The computer has a modified operating system called SIMex-Plus optimized for running simulations. This new system will make use of the existing cockpit, motion base and sound system as well as a Fiber Optic Helmet Mounted Display (FOHMD). Using a magnetic head tracker, the FOHMD gives the pilot a far superior field of view over a single fixed monitor in the back of the cockpit which is the primary visual display for the present Sea King simulator.

It was originally hoped that the algorithms created for the DIS interface would be transferred to the new simulator system once it was online. However, due to delays in transferring the GenHel model, the transfer of the DIS algorithms will have to be done sometime in the future.

4.2 The Ship Simulators

Throughout this research project, the ship simulator used underwent various changes. To clarify this issue, it is necessary to outline exactly what the configurations
configurations.

4.2.1 Configuration One

Configuration One is not really a ship simulator. This arrangement refers to the Sea King Simulator before the DIS interface was added. As discussed in Section 4.1.2, the ship motion is based on a pre-recorded data file which is input to the helicopter simulator internally.

4.2.2 Configuration Two

In order to develop the DIS interface for the Sea King Simulator, it was necessary to communicate with a ship. To allow the author to develop the necessary code without involving DCIEM, the author developed a simple ship simulator that ran locally. In order to provide a useful platform for developing the DIS algorithms, the simulator had to meet the following requirements:

i. provide reasonable, but not necessarily accurate ship motion in all six degrees of freedom,

ii. conform to the DIS standard with the use of a Dead Reckoning Model (DRM) to signal the issue of an Entity State PDU,

iii. record the created motion to compare with the motion recreated in the helicopter simulation,

iv. record the data contained in the Entity State PDUs received from the Helicopter simulator, and

v. run in real time.
algorithms as the helicopter simulator.

To provide reasonable ship motion, the heave, pitch and roll motions were created by a sine curve of fixed amplitude and frequency. The phase of each channel was offset by a constant amount. The heading and the velocity were determined by a set of commands input from a data file. Each new command was acted upon every ten seconds. The ship transitioned from the old set of conditions to the new conditions over the following ten second period. If the command inputs were reasonable, the ship motion was sufficient to achieve the objective of sensible, but not necessarily accurate ship motion. To conform with the DIS standard, a ship DRM was implemented in the ship simulator using the same DRM equations as the Helicopter simulator used (Refer to Chapter 5). The coordinate systems used by the ship simulator were similar to the GenHel frames.

The final requirement for the ship simulator was to run in real time. Due to a limit of available computers in the lab, the simulator ran on an older SGI workstation (Hal). This necessitated the use of a much larger time step than the helicopter simulator so that the code could be executed in real time. The actual value was 0.1 seconds compared to the 33 milliseconds used in the Helicopter simulator.

Thus, Configuration Two refers to a Ship simulator running externally to the Helicopter simulator, but running within the Flight Simulation Lab. The two simulators communicated via the Ethernet in the lab. The internal ship motion described in Configuration One was overridden by the DIS interface in the Helicopter Simulator.
Configuration Three is identical to Configuration Two except the Ship simulator no longer ran locally within the Flight Simulation Lab. Instead, the software was exported to a machine located at DCIEM so that the next level of testing could be done before linking with DCIEM's ship simulator. Consequently, the two simulators communicated via the PPP connection described in Chapter 8.

### 4.2.4 Configuration Four

This is the final configuration used in this project. As in Configuration Three, the ship simulator was executed at DCIEM with the helicopter simulator running at UTIAS. Unfortunately due to time constraints at DCIEM, the LSO simulator had not been completed. In order to allow completion of this research, the ship simulator described in Configuration Two and Chapter 7 was modified by DCIEM. They added a visual display and a DRM for the helicopter so that its position was known for every time step. This created a fully functional ship simulator so that the UTIAS interface could be evaluated. Again, communication between the simulators was accomplished via the PPP connection.
Figure 4.1: External View of UTIAS Simulator

Figure 4.2: Internal View of Cockpit
Figure 4.3: Layout of Connections between Sea King Simulator Computers

Figure 4.4: Visual Display on IRIS5
Figure 4.5: Helicopter and Ship Simulator Configurations
Chapter 5

Investigation into Bandwidth Requirement

The first stage in implementing the DIS interface for the Sea King flight simulator was determining the bandwidth required for the connection between the two simulators. Initially, DCIEM wanted to use a modem to act as this link between UTIAS and DCIEM. To determine if this was possible, the maximum number of Entity State PDUs needed to be sent per second was required. This would provide information on the bandwidth necessary for the communication link. The code for this section is contained within the files dis.c and dis_pdu.c. This development stage used Configuration One described in Section 4.2.2. The helicopter simulator was modified to include a crude DIS interface, but there was no link to an external ship simulator.

5.1 Helicopter and Ship Dead Reckoning Parameters

In order to implement the DIS interface in the helicopter simulator, it was necessary to have access to the variables listed in Table 5.1. For the helicopter model in the Sea King simulator, the DIS interface needs to know the value of each data item for the current time step. For the ship model in the helicopter simulator, the interface must be able to substitute its own value from the remote ship simulator which will replace the internal value used when the helicopter simulator is run in Configuration One. The ship data items are easily accessible since the values are assigned within bhawk_pe_exec.c. However, in order to access the helicopter dead reckoning parameters it was necessary to access the flight equations. To do this, an addition was made to the file called strike.f. A
parameters. This array was then accessible, through a common block, to the DIS algorithms in *dis.c*.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Position</td>
<td>Phi</td>
</tr>
<tr>
<td>Y Position</td>
<td>Theta</td>
</tr>
<tr>
<td>Z Position</td>
<td>Psi</td>
</tr>
<tr>
<td>X Velocity</td>
<td>Phi Angular Velocity</td>
</tr>
<tr>
<td>Y Velocity</td>
<td>Theta Angular Velocity</td>
</tr>
<tr>
<td>Z Velocity</td>
<td>Psi Angular Velocity</td>
</tr>
<tr>
<td>X Acceleration</td>
<td>Angular Rate - p</td>
</tr>
<tr>
<td>Y Acceleration</td>
<td>Angular Rate - q</td>
</tr>
<tr>
<td>Z Acceleration</td>
<td>Angular Rate - r</td>
</tr>
</tbody>
</table>

Table 5.1: Dead Reckoning Parameters

5.2 Dead Reckoning

The next stage in creating the interface was the implementation of the dead reckoning equations. This area can be partitioned into two. The first part was the selection and implementation of equations and the second was picking the position and orientation thresholds used to trigger an update.

5.2.1 Selection of Dead Reckoning Model

The DIS architecture provides a variety of equations for use in dead reckoning algorithms. The equation type is based on the complexity of the entity's movement. Some entities are stationary objects while others go through a full range of motions. To classify the different entities, the DRM notation consists of three elements. The first specifies whether the entity is fixed or rotating (F or R). The second describes whether the dead reckoning rates are to be held constant with respect to velocity or acceleration (that is rate of position P or rate of velocity V). Finally, the third element specifies if the
application the DRM is RVW. DRM-RVW was also implemented in the ship simulator.

The equations appropriate for this DRM are shown in Equation 5.1.

\[
\begin{align*}
    x_1 &= x_0 + v_0 \cdot \Delta t + \frac{1}{2} \cdot a_0 \cdot (\Delta t)^2 \\
    v_1 &= v_0 + a_0 \cdot \Delta t \\
    a_1 &= a_0 \\
    \theta_1 &= \theta_0 + \theta_0 \cdot \Delta t \\
    \dot{\theta}_1 &= \dot{\theta}_0 \\
\end{align*}
\]

where \( x \) is the position, \( v \) is the linear velocity, \( a \) is the linear acceleration, \( \theta \) is the orientation, \( \dot{\theta} \) is the angular velocity, \( \Delta t \) is the time step of the simulation, \((\ )_0\) refers to \( t = 0 \), and \((\ )_1\) refers to \( t = \Delta t \).

**Equation 5.1 : Dead Reckoning Model - RVW**

### 5.2.2 Selection of DRM Thresholds.

The final stage of implementing the DRM requires the selection of the position and orientation thresholds (\( \Delta \text{position} \) and \( \Delta \text{angle} \)) which trigger the helicopter simulator’s DIS interface to send an updated Entity State PDU. The update is required when the difference between the complicated model of an entity and the DRM of that same entity
almost constantly which requires a high bandwidth connection. If the threshold is very large, when an update occurs the entity would jump from the DRM estimated position to the correct position. In this application, a jump would be at the least disconcerting, and as a worst case scenario it could make the helicopter jump away from the ship just as the LSO is about to say “Down, Down, Down.” It was hoped through this investigation that it will be possible to select a threshold that keeps the updates within the bandwidth, but does not cause unsightly jumps for the LSO.

The first goal was to identify what size of threshold would be acceptable. Since it was not possible to watch the helicopter move from the ship simulator because the link between the simulators at DCIEM and UTIAS had not been established, the thresholds chosen were based on the pilot’s view of the ship, not the LSO’s view of the helicopter. Thus, the thresholds that should be used in the ship simulator’s DIS interface were identified, not the helicopter simulator’s thresholds. It was assumed that the thresholds would be similar for the helicopter simulator. First, the helicopter simulator’s DIS interface was modified to make the ship jump back and forth the amount of the threshold. Then, setting up a view point of about fifty feet from the hangar face the author subjectively assessed how noticeable the jump was. The distance of fifty feet was chosen because it gave the pilot a reasonable view of the ship, but was not so far back that the pilot couldn’t discern any motion. Then, the threshold was successively reduced until it was just possible to discern movement of the ship. This was then repeated for the orientation threshold. The results are shown in Table 5.2. As a final check on the validity of these thresholds, two other graduate students in the lab flew an approach to the ship
their flight.

<table>
<thead>
<tr>
<th>Δposition</th>
<th>½&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Δangle</td>
<td>0.1°</td>
</tr>
</tbody>
</table>

Table 5.2. : Selected Dead Reckoning Thresholds

The next step was to see what sort of bandwidth was required to support these thresholds. If the required bandwidth was not supportable it would then be necessary to compromise the thresholds by slowly increasing them until the bandwidth requirements were met.

5.3 Results

With DRM implemented the system was ready to be used to determine the bandwidth required for the connection. The experiment consisted of flying a typical approach to the ship followed by a touchdown. This would put the helicopter DRM through the full range of expected motions so that its limits would be adequately tested. This flight profile was then repeated for different thresholds. The results were a time history of the number of packets sent per one second interval for each set of thresholds based on the time the packet was transmitted. Configuration One was used for this experiment with the ship motion provided internally to the simulator.

The results from the flight profiles were excellent. Using the thresholds derived in 5.2.2 the number of packets sent per second was within the expected bandwidth required for a modem. Figure 5.1 contains the actual results for the flight. It is clear that the maximum value for packets per second occurs during the initial deceleration of the helicopter from 30 knots. Once the aircraft is stabilized in the hover, the number of
touchdown, but this is only 4 packets per second. If a 28.8 kilobits per second (kbps) modem is used for the connection between DCIEM and UTIAS, it would be expected to handle 22 packets per second in each direction if no overhead is assumed. This is based on an Entity State PDU being 1280 bits in size (only one articulation parameter is assumed). The maximum value achieved for packets per second during the test stage was eight. Even with a 50 percent overhead on the modem, the bandwidth is sufficient for the deck landing task.
Figure 5.1: Packets per Second for Typical Flight Path

with Thresholds from Table 5.2
Chapter 6

Implementation of a Basic DIS Interface

The next step in the implementation of the DIS interface was to build a shell which would allow a DIS exercise to function at a very basic level. The goal was to adopt the structure of Configuration Two. This would lay the groundwork for further expansion of the interface to support the full DIS standard. The critical elements of this basic shell were as follows:

i. simulation applications would communicate over the Ethernet,

ii. only Entity State PDUs would be transmitted and received, and

iii. the simulation would take place over a Local Area Network (LAN) within the Laboratory.

6.1 Communication Algorithms

In order to send packets between Simulation Applications it is necessary to communicate over the Ethernet through the UNIX network communication functions. This section deals with the algorithms created to access these functions and allow bi-directional communication between the two simulators. The code is contained in the file dis_comms.c.

The first step in communicating over the Internet is to open a socket on the local computer. The address type is an Internet socket, AF_INET which supports UDP and TCP protocols. The second item required in the creation of a socket is the type of socket.
The second step is to get the port number that the algorithms are going to use for communication. For the DIS application, the following non-standard port was created, *DIS 1999 UDP*. This means the name of the port is DIS, the number is 1999, and the protocol type is UDP. It was necessary to add this line to the file *services* on all machines involved in the DIS exercise. Referring back to 3.7, it was stated that for simulation data UDP was the appropriate protocol. This is consistent with the goals of this section to transmit only Entity State PDUs.

Once the information on the service has been gathered, it is necessary to set various options for the socket. First the input buffer size is specified. It is possible through time delays in the transmission of data, or through slow downs in the simulator software, for multiple packets to build up in the input buffer. Once the buffer is full, any other packets arriving will be discarded. To avoid this, the buffer size is increased to accommodate multiple packets.

The next step is to assign components of the destination data structure the same values as the local machine’s. The two parts are the address type, and the port number. The final option allows the socket to be queried to see if a packet has arrived. If no packet has arrived the default action is to make the application wait until a packet does come. However, this is not acceptable for a real time simulation. Instead, by use of the *ioctl* command, the query returns a zero if a packet is not available.

Following this, the network communication algorithms must get the Internet address of the remote machine to which the packets are being sent. This is done through
structure containing all the destination information. It should be noted that this method of communication is valid for sending data from one machine to another. However, it is common practice in the DIS standard to broadcast packets to multiple machines instead. This method was not pursued for these algorithms since it was not necessary for two simulators. An alternate method of getting data on the remote computer was added using the `gethostbyaddr` command. This call works the same as `gethostbyname` except the IP address is given instead of the name of the remote computer.

The final step in the initialization is to empty the input buffer of all data which may have built up during previous runs. The DIS interface can then send and receive packets from a remote computer via the Internet. The type of packets sent were Entity State PDUs. The implemented structure of this packet was taken from the file `dis_pdu.h` which was provided by the NPSNET research group mentioned in Chapter 2.

### 6.2 Final comments and Results

Upon completion of the network communication algorithms, ship simulator and the initial developments for bandwidth determination, the basic DIS shell was complete. This interface allowed a very crude DIS to occur within the Flight Simulation Lab at UTIAS as detailed in Configuration Two. The pilot was able to interact with a moving ship controlled by the ship simulator running on a computer remote to the helicopter simulator. The two simulation applications communicated position and orientation information through the use of Entity State PDUs. Dead Reckoning algorithms were used to reduce the bandwidth required for communication.
One problem was evident from the start; the ship motion, as seen by the pilot, was not smooth. The ship appeared to follow the correct path, but instead of transitioning smoothly from one location to another, the ship would jump. During the initial investigation, it became apparent that each jump of the ship was associated with the arrival of updated data in a Entity State PDU. From the testing done with dead reckoning thresholds in Chapter 5, the updated information should cause a jump which the pilot cannot notice; however, this was obviously not the case. Instead, the predicted ship position was always lagging behind the updated correct data by more than the threshold amount. So, when an update was received, the ship would have to make a larger jump to catch up to the correct position.

The root of the problem was timing. Both simulators were supposed to be running in real time - the ship simulator at 0.1 seconds per time step and the helicopter simulator at 33 milliseconds per time step; however, the helicopter simulator was running at 42 milliseconds per time step. Since all the time steps in the calculations are fixed quantities, calculation errors occurred. For the DIS application, the dead reckoning equations were using 33 milliseconds as their fixed time step. For example, assume the ship is moving at a constant 16 feet per second; the ship simulator's DRM for the ship would take 10 time steps (one second) to move the ship 16 feet. The helicopter simulator's DRM for the ship would take 30 time steps to do the same. However, due to the slower real time step, the helicopter simulator's DRM for the ship would only execute 24 time steps in a second. Thus, the two DRMs which should produce the exact same results since they both started with the same initial conditions were no longer synchronized. Consequently, when the
thresholds indicated that the ship simulator should issue an update for the ship DRM, the ship DRM located in the helicopter simulator would only have moved 12.7 feet, forcing an instantaneous jump of 3.3 feet plus the threshold $\Delta$position.

Once the problem had been identified as an incorrect time step, the solution was straightforward. The complexity of the visual display was reduced so that IRIS5 had no problems updating the image at the correct rate of 30 Hz. Unfortunately, this method reduced the quality of an already poor display of the rear of the ship. Nonetheless, with the new simulator coming on line in the near future, this was not regarded as a major issue.
Chapter 7

Smoothing

7.1 Abrupt motion of Helicopter on Deck

With the implementation of the crude DIS interface, detailed testing was carried out in order to isolate more subtle problems. A major difficulty was immediately noticed; soon after touchdown on the deck of the ship, the helicopter experienced a very violent acceleration forward which catapulted the helicopter right off the deck. Further investigation revealed that the ship had to be moving forward for this to occur; thus, the problem was most noticeable in the $x$ direction as this axis had the highest velocity. It was also discovered that this violent acceleration forward only occurred during the arrival of an Entity State PDU. However, this did not happen for every Entity State PDU received. Similar to the problem discussed in the previous chapter, this was again a discrepancy between the DRM and the correct information contained in the Entity State PDU. When the ship did jump, it moved approximately a half foot more than the expected threshold jump of half an inch. Over a single time step of 33 milliseconds this corresponds to an $x$ acceleration of the ship equal to 450 ft/s$^2$. Since the landing gear on the Sea King is modeled as stiff springs, it is understandable why the helicopter reacted violently to such a spike.
It was discovered that the source of the problem was an incorrect time step. Unlike the error in the previous chapter, these inconsistencies only occurred occasionally. Figure 7.1 shows a plot of time step size versus time for a typical helicopter simulation running Configuration Two. One can see that the simulator does not stick to an exact time step of 33 milliseconds; rather, larger time steps occur. Thus, the next step was to isolate the cause of this inconsistent time step and see if it would be possible to prevent it from occurring.

It was initially suspected that the cause of these slowdowns was the visual display on IRIS5. The system was already working close to its update limit in order to create the image and it was theorized that if the ship was seen from a particular position or angle, IRIS5 would be unable to compute the display within the assigned time. However, an extensive comparison between helicopter location and time step data revealed no such correlation.

The next possibility considered was that other processes running in the background on the simulator computers were interfering with the simulation. Originally, it was believed that other users created a workload on the CPU that prevented the simulation from completing the calculations within the time step. While this was true, the problems still existed even when there were no other users on the system. Research made it apparent that the source of these inconsistencies was processes that the operating system (OS) on the computer had to run. Since the workstations are not dedicated to the simulation task it was unavoidable that background processes would eventually interfere with real time programming. Figure 7.2 is chart of time step size versus time. This data
results shown it was concluded that the source of the time step inconsistencies was housekeeping processes by the computer’s OS.

7.3 Solutions

Since the time inconsistencies are at the OS level of the computer, it is not possible to remove these slow downs; thus, the DIS interface needed to be modified so that it could gracefully handle these situations. There are various methods to compensate for the jumps. The following is a list of options that were considered.

7.3.1 Lock Helicopter onto deck

Since the jumps experienced by the ship only directly affect the helicopter once touchdown occurs it is possible to lock the helicopter down onto the deck once it lands. Once locked down, the helicopter’s position and orientation would be set exactly the same as the ship’s. Thus, when a jump occurs, the gear equations would be out of the loop and the helicopter would remain attached to the ship. This would be a simple solution to the problem and would require no extra CPU usage.

The primary disadvantage of this method is its inflexibility. Once the flight equations are taken off line from the simulation, it is impossible to properly re-trim them so that the helicopter can takeoff again during the run. As well, this method left a gray area as to when the lockdown should occur since it is desirable for the pilot to see the reaction of the gear to a touchdown on the deck. If the lockdown occurred just as the first wheel touched then the pilot would not be able to see the dynamic response of the other wheels landing on the deck. However, if the lockdown occurred once all wheels
7.3.2 Smoothing

Another method of addressing this issue is smoothing as is discussed in Lin et. al. (1995). Rather than allowing the ship to jump from the predicted position to the new updated position, the ship transitions from the old position to the new one by use of a smoothing algorithm (Figure 7.3). The primary advantage is the flexibility of this method. The helicopter would be able to take off and land multiple times on the deck. If a new update is received while smoothing is still taking place, the smoothing would just start again from the present position. However, there are definite disadvantages; the most significant is the introduction of error in the trajectory of the ship. With the instant update, the difference between the trajectory of ship seen by the pilot and the actual ship was no larger than the dead reckoning thresholds. With this method, the potential exists for much larger errors between the two. The second disadvantage is the increase in CPU time required to process the smoothing algorithms.

7.3.3 Integration of Acceleration Data

The third possible solution is to take the acceleration data received from the ship simulator and integrate it to re-create the ship’s trajectory. This method would prevent any spikes in acceleration data. When a new update is received a cubic polynomial would be fit to the acceleration data to provide accelerations for the future. Then, this data would be integrated to create the position of the ship. To keep the trajectory error to a minimum, a filtering scheme would be employed. The updated position information would
has been sent through a High Pass filter (Figure 7.4). Again, this method would have the flexibility of the smoothing scheme, but also increased complexity requiring more CPU time. How this method would be able to reduce the trajectory error compared to the smoothing method is unknown and might be an improved method.

### 7.3.4 Chosen Solution

Based on the possible solutions, smoothing was chosen. It had the advantage of being sufficiently flexible, but not overly complicated. Although the integration method had potential, it was thought of after smoothing had already been implemented. Thus, this avenue was not pursued since smoothing was found to function satisfactorily. If time had permitted, it would have made an interesting study to see how effective the integration method could have been.

### 7.4 Smoothing Concepts

For the following discussion it is important to define a few terms. For this case, only smoothing along the x axis will be discussed. However, the discussion can be extended for the other axes and Euler angles. Also note that this discussion is written for the helicopter simulator receiving updated data from the ship simulator. The point where the dead reckoning algorithms have predicted the ship to be is $x_1$. This is at time $t_1$ with a velocity of $v_1$ and acceleration of $a_1$. The predicted point in the future at time $t_2$ is $x_2$ with a velocity of $v_2$ and acceleration of $a_2$. The new updated position for the ship at time $t_1$ is $x_n$ with a velocity of $v_n$ and acceleration of $a_n$. The time step of the simulation is $\Delta t$ and the number of smoothing steps is given by $p$. 
Equation 7-1: Predictor

The first step is to take the updated values of \( x_n, v_n \) and \( a_n \) and predict forward a time of \( \Delta t \cdot p \). For increased accuracy a higher order predictor could be used. However, for this application, the simple form shown in Equation 7.1 was implemented. The output of this predictor is \( x_2, v_2, a_2 \). Next, it is necessary to fit a smoothing equation between points \( x_1 \) and \( x_2 \). There are plenty of possibilities for these equations. For this application linear, cubic and fifth order equations were examined. The smoothing equations are given in Equations 7.2, 7.3, and 7.4 respectively. For a full derivation of the cubic and fifth order equations refer to Appendix A.

\[
x_i = x_1 + \frac{x_2 - x_1}{p} \cdot i
\]

where \( i \) is an integer from 0 to \( p \),

\( i = 0 \) at time \( t_1 \),

\( i = p \) at time \( t_2 \), and

\( (\_)_i \) is the value at \( t = i \cdot \Delta t \)

Equation 7-2: Linear Smoothing Algorithm
\[ t_i = i \cdot \Delta t \] and \( i \) is an integer from 0 to \( p \)

\[
\alpha_1 = v_1
\]

\[
\alpha_2 = \frac{-1}{t_2} \cdot (2 \cdot v_1 + v_2) + \frac{3}{t_2^2} \cdot (x_2 - x_1)
\]

\[
\alpha_3 = \frac{1}{t_2} \cdot (v_1 + v_2) + \frac{2}{t_2^3} \cdot (x_1 - x_2)
\]

where \( t_i = i \cdot \Delta t \) and \( i \) is an integer from 0 to \( p \)

\[ x_i = \alpha_0 + \alpha_1 \cdot t_i + \alpha_2 \cdot t_i^2 + \alpha_3 \cdot t_i^3 \]

\[ v_i = \alpha_1 + 2 \cdot \alpha_2 \cdot t_i + 3 \cdot \alpha_3 \cdot t_i^2 \]

\[ a_i = 2 \cdot \alpha_2 + 6 \cdot \alpha_3 \cdot t_i \]

**Equation 7-3 : Cubic Smoothing Algorithm**

\[
\alpha_0 = x_1
\]

\[
\alpha_1 = v_1
\]

\[
\alpha_2 = \frac{1}{2} \cdot a_1
\]

**Equation 7.4 : Fifth Order Smoothing Algorithm**

Continued on next page
\[ a_3 = \frac{1}{2 \cdot t_2^3} \left[ 20 \cdot (x_2 - x_1) - 4 \cdot t_2 \cdot (2 \cdot v_2 + 3 \cdot v_1) + t_2^2 \cdot (a_2 - 3 \cdot a_1) \right] \]

\[ a_4 = \frac{1}{2 \cdot t_2^4} \left[ -30 \cdot (x_2 - x_1) + 2 \cdot t_2 \cdot (7 \cdot v_2 + 8 \cdot v_1) - t_2^2 \cdot (2 \cdot a_2 - 3 \cdot a_1) \right] \]

\[ a_5 = \frac{1}{2 \cdot t_2^5} \left[ 12 \cdot (x_2 - x_1) - 6 \cdot t_2 \cdot (v_2 + v_1) + t_2^2 \cdot (a_2 - a_1) \right] \]

where \( t_i = i \cdot \Delta t \) and \( i \) is an integer from 0 to \( p \)

\[ x_i = \alpha_0 + \alpha_1 \cdot t_i + \alpha_2 \cdot t_i^2 + \alpha_3 \cdot t_i^3 + \alpha_4 \cdot t_i^4 + \alpha_5 \cdot t_i^5 \]

\[ v_i = \alpha_1 + 2 \cdot \alpha_2 \cdot t_i + 3 \cdot \alpha_3 \cdot t_i^2 + 4 \cdot \alpha_4 \cdot t_i^3 + 5 \cdot \alpha_5 \cdot t_i^4 \]

\[ a_i = 2 \cdot \alpha_2 + 6 \cdot \alpha_3 \cdot t_i + 12 \cdot \alpha_4 \cdot t_i^2 + 20 \cdot \alpha_5 \cdot t_i^3 \]

**Equation 7.4 : Fifth Order Smoothing Algorithm continued**

In order to select the correct smoothing equation, it is necessary to clarify what level of smoothness was sought. Lin et. al. (1995) recommends a linear smoothing algorithm for most tasks. The cubic algorithm was not recommended in Lin's paper because the disadvantages of a larger error in the trajectory and the use of more CPU time outweighed the small improvement in smoothness. However, it needs to be noted that for Lin et. al. (1995) the criteria for smoothness was that the image display was smooth. In this DIS application, the requirement for smoothness is "gear equation smoothness." That is, the smoothing algorithms must prevent large acceleration spikes from entering the flight equations through the deck helicopter interface. Since the level of smoothness required was much higher, an investigation into the response of the different algorithms was required.
from the use of the different smoothing algorithms responding to an input of a half foot jump in position. This is indicative of the original problem that the smoothing algorithms were attempting to overcome. Figure 7.5 shows the acceleration created by such an input when only the DRM is used as was the case before smoothing was implemented. Equation 7.5 describes how the acceleration was calculated from the output of the DRM. Figure 7.6 is a comparison between the different types of smoothing algorithms for the same input. The linear smoothing algorithm also used Equation 7.5 to calculate the acceleration. However the cubic and fifth order algorithms provided this information explicitly (Equations 7.3 and 7.4). It is clear from the chart that although the linear algorithms reduce the acceleration spikes drastically, it cannot reduce them to a reasonable value. The primary difference between the cubic and fifth order response is smoothness. The fifth order algorithms imply that acceleration will be a continuous function where as the cubic equations will only make velocity continuous. However, the maximum accelerations experienced by both are quite similar. Consequently, to avoid the use of more CPU time for processing the increased complexity of the equations, the cubic smoothing algorithms were adopted.
where \( t = 0 \) at \( n = 0 \), and

\[
\begin{align*}
    v_n &= \frac{x_{n+1} - x_n}{\Delta t} \\
    a_n &= \frac{v_{n+1} - v_n}{\Delta t}
\end{align*}
\]

\( (\phantom{\text{a}})_n \) refers to \( t = n \cdot \Delta t \)

Equation 7.5 : Method of Calculating Acceleration from Output of DRM

7.5 Ship Simulator

From the experience in the previous sections it became evident that the implementation of the test ship simulator described in Section 4.2.2 was inadequate. In order to justify the optimal values for the ship simulator's dead reckoning thresholds and the number of smoothing steps, the decisions made needed to be based upon accurate ship motion. Although the ship motion created by the present simulator was reasonable it was not correct and could not fulfill this requirement.

The ship motion used by the original helicopter simulator consisted of half an hour of data created by a program called Fredyn. The details of how this program creates the ship motion is classified as it belongs to the Department of National Defence. However, the output is accurate ship motion with data for all six degrees of freedom for a Sea State of 5. This is a very rough sea which borders on the limits for helicopter operations on a ship. The data is available at a sampling rate of 2 Hz.

It was decided to modify the test ship simulator to use the time history of motion that was available from Fredyn. This would provide a realistic basis upon which to base
advantage to the motion being Sea State 5. Since this was close to the limit of normal operations, adjusting the algorithms to handle this motion would guarantee that smoother Sea States would function correctly. For use in the ship simulator, the data needed to be interpolated from 2 Hz to 30 Hz to provide motion data for each time step. A cubic polynomial was used as the interpolation scheme.

7.6 Investigation into optimal values for DRM and Smoothing Model

The objective of this section was to find the optimal values for the adjustable parameters in the helicopter and ship’s DIS interface which minimize trajectory error, but still provide sufficient smoothness to avoid jumps. The parameters that can be adjusted are the dead reckoning thresholds in the ship simulator, $\Delta$position and $\Delta$angle which determine when an update is sent from the ship simulator and the number of smoothing steps $p$ in the helicopter simulator.

It is beneficial at this point to discuss the detrimental effects of trajectory error. Consider a trajectory error of 5 feet in inertial space along the y axis. From the pilot’s point of view, this error is irrelevant; he will line up with the ship that is being displayed in the visual display unit. However, the LSO on the deck of the ship will be looking at the helicopter to determine when it is centrally over the deck so that the command “down, down, down” can be issued. When the pilot believes himself to be central, he could touchdown. However, the LSO will not issue the command since he believes the helicopter to be almost off the deck. Thus, it is clear that a large trajectory error is unacceptable. Exactly how large the error can be is unknown, but it should at least be
Without this level of accuracy, the LSO will not be able to assist the pilot to line up centrally for landing.

The first step in selection of the optimal values is to pick starting values for the ship simulator's Δposition and Δangle. If their values are based solely on what smoothing allows, the threshold values can be much larger than the values listed in Table 5.2 since all jumps will be evened out. However, it is still important to reduce the trajectory error and the thresholds are expected to have an effect on this. It was reasonable to begin by picking thresholds that corresponded to the maximum acceleration experienced in the ship simulator's ship motion which the DRMs try to track. For this motion data, Δt = 0.1 seconds. The maximum linear acceleration occurs in Z and has a value of 4.992 ft/s². This corresponds to a Δposition of 0.00555 ft. The maximum angular acceleration is in pitch with a value of 5.44 degrees/s². This equates to a Δangle of 0.00604 degrees. Because these values were only starting points, the thresholds were rounded up to the values in Table 7.1.

<table>
<thead>
<tr>
<th>Δposition</th>
<th>0.01 ft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Δangle</td>
<td>0.01°</td>
</tr>
</tbody>
</table>

Table 7.1. Updated Dead Reckoning Thresholds

For consistency, the dead reckoning thresholds used in the helicopter simulator were also modified so they matched the values listed in table 7.1. Consequently, the bandwidth required for the link between DCIEM and UTIAS also changed. However, since the original results revealed a large margin before saturation of the link occurred, it was assumed that the bandwidth would still be sufficient for this task.
error. To measure smoothness, two quantities were used; the maximum and average acceleration along the x axis experienced during the run. These maxima and averages were based on the absolute value of the acceleration. For trajectory error the same principle was applied; the maximum and average x difference in inertial space between the real ship motion and the ship motion displayed to the pilot. Recording the acceleration was straightforward since the information was available for each time step. However, in order to record the difference data some modification of the code was required. First, the time history of x for the ship calculated from the helicopter simulator's DRM was recorded. Then, after the simulation was complete, the data file used by the ship simulator to produce the ship motion, was read by the helicopter simulator. This data was then interpolated, and compared to the DRM's recorded x time history to provide the trajectory error. The one inconsistency of this method is the interpolation scheme. For the ship simulator, the input ship motion is interpolated from 2 Hz to 10 Hz. For the helicopter simulation, it was necessary to interpolate the input ship motion from 2 Hz to 30 Hz to allow a direct comparison. It was assumed that the difference in the interpolation frequencies was negligible. It was expected that as \( p \) increased the acceleration data would decrease and the trajectory error would increase.

Next, a series of DIS exercises were run between the ship and helicopter simulator using the modified ship simulator in Configuration Two. Each run was 30 seconds in length with full ship motion present. The helicopter just hovered since its motion was irrelevant to the outcome of the experiment. The procedure was repeated with \( p \) varying.
The charts show that maximum and average acceleration are related in the expected manner to p. A power fit of the points proved to be an excellent match. For comparison, the maximum and average accelerations produced by the ship simulator have been added. It is interesting to note that the average acceleration curve approaches the average real ship acceleration as an asymptote. However, the maximum acceleration curve crosses the maximum real ship acceleration at $p = 71$ (Figure 7.11). Based on this information, $p = 72$ was chosen as the optimal value. This value is relevant only if smoothness is considered important. The reason why p was incremented by one was to allow for a small amount of error in the results.

Analysis of the trajectory error data does not reveal any conclusive results. As p increased it was expected that the difference would also increase. This was clearly not the case. Further examination of the trajectory error results revealed that the difference is dependent upon the length of the simulation rather than the value of p. This is because of an unforeseen occurrence. The issue of the helicopter simulation not running in real time for every time step has already been discussed (Section 7.2). However, the same conditions also apply to the ship simulator. Again, housekeeping by the OS was preventing the real time portion of the software being completed within the required time step. The number of time delays experienced by the ship simulator varied for each run making it difficult to obtain consistent results. Sometimes only one delay would occur during the 30 second run, but in some cases it was much higher, around 15. Thus the data that was used by the helicopter simulator for direct comparison with the recorded time
history of the ship at the end of the simulation was not synchronized with the actual data transmitted by the ship simulator. Consequently, the method chosen to quantify the trajectory error was flawed.

To overcome this unforeseen obstacle a new approach was developed to measure the trajectory error. Consider a vector \( \vec{R} \) which points from the centre of gravity of the ship to the centre of gravity of the helicopter. \( \vec{R} \) has three components, \( \Delta x \), \( \Delta y \), and \( \Delta z \) measured in the GenHel inertial frame. For every time step of the DIS exercise, each simulator records the components of \( \vec{R} \). Whenever an Entity State PDU is sent, the values of these components for that time step are transmitted to the other simulator. In order to transmit this data, the Entity State PDU was modified as shown in Table 7.2. Thus, with the complete set of \( \vec{R} \) data for every time step coupled with regular time stamped updates of \( \vec{R} \) from the other simulator, it was possible to plot a time history of the helicopter simulator's \( \vec{R} \) and the ship simulator's \( \vec{R} \). Thus, the data is synchronized, and one is able to quantify the trajectory error. Unfortunately, the test ship simulator only records received helicopter data, and is unable to calculate \( \vec{R} \). Therefore, a full analysis of trajectory error had to wait for the final test runs with the full ship simulator at DCIEM using Configuration Four. This was done in Chapter 11.
## Dead Reckoning Parameters

<table>
<thead>
<tr>
<th>Dead Reckoning Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Padding</td>
</tr>
<tr>
<td>R Vector - X</td>
</tr>
<tr>
<td>R Vector - Y</td>
</tr>
<tr>
<td>R Vector - Z</td>
</tr>
<tr>
<td>Other Parameters</td>
</tr>
<tr>
<td>Entity Linear Acceleration</td>
</tr>
<tr>
<td>Entity Angular Velocity</td>
</tr>
</tbody>
</table>

### Table 7.2: Modified Portion of Entity State PDU

### 7.7 Final Comments

The reason for the implementation of the smoothing algorithm in the DIS interface was to overcome the violent jerks experienced by the helicopter in contact with the deck of the ship. Through the use of cubic smoothing algorithms with the number of smoothing steps \( p \) set to 72, the maximum and average accelerations experienced by the helicopter simulator’s DRM were reduced to reasonable values. An actual flight test was conducted using Configuration Two, demonstrating that the helicopter was no longer thrown off the deck and only experienced minor vibrations while in contact with the ship when a jump occurred. However, it was unknown at this point how trajectory error has been affected. If the error was too high, a compromise would have to be reached between smoothness and error.
Figure 7.1: Helicopter Simulator Time Step Size

Figure 7.2: Time Step Size of Dedicated Real Time Program
Point 1: Ship's position provided by helicopter simulator's DRM at $t = t_1$.

Point 2: Ship's position provided by new update from Ship Simulator at $t = t_1$.

Point 3: Position ship would jump to at $t = t_1 + \Delta t$ if no smoothing occurred.

Point 4: Future position of ship at $t = t_1 + p \cdot \Delta t$ predicted from data received from ship simulator.

**Figure 7.3: Smoothing Concept Diagram**

**Figure 7.4: Filtering Aspects of Integration Scheme**
Figure 7.5: Acceleration Resulting from Inputting a Half Foot Jump into the DRM

Figure 7.6: Comparison of Acceleration Responses for Linear, Cubic, and Fifth Order Smoothing Algorithms
Figure 7.7: Maximum Acceleration Recorded During Run

Figure 7.8: Average Acceleration Recorded During Run
Figure 7.9: Maximum Difference Recorded During Run

Figure 7.10: Average Difference Recorded During Run

66
Figure 7.11: Zoom of Figure 7.7
Chapter 8

Linking DCIEM and UTIAS

In order to move the testing of the DIS interface from Configuration Two to Three, a link between DCIEM and UTIAS needed to be established. It was mentioned earlier that this link would be created via a modem. However, it is important to discuss the other possibilities that existed for the connection and why these options were not selected.

8.1 The Internet

The simplest method of connection would be to use the already existing link between DCIEM and UTIAS via the Internet. Since the network communication algorithms already use the local Ethernet in the lab to transmit data, no changes would be necessary to use this medium except the destination address. However, there are some obvious problems with the Internet connection. The first area of concern is speed. There is a standard command called ping that is used over the Internet to determine the transmission time between two computers. Ping sends a small packet once a second which is returned by the destination computer. From this data, the minimum, average and maximum round trip time is calculated. Figure 8.1 shows the ping statistics between IRIS5 and SIM2, a computer at DCIEM. It is clear that the time to send a packet is quite variable and sometimes very large. Normally, it is assumed that computers which are geographically close will have small round trip times. However, this is not the case with DCIEM. Since DCIEM is a Defense Research Establishment (DRE), it is attached to the
travel the 3 km from UTIAS to DCIEM, it has to go via Ottawa. This implies a much slower connection as well as highly variable transmission times. The second problem with the Internet is a firewall created by the Department of National Defense. As a security measure, all non-standard Internet services are blocked from accessing the DREnet. Since a DIS application is not a standard Internet service, all attempts to transmit packets to DCIEM were stopped. Thus, it was obvious that the Internet could not be used to link the simulators.

8.2 ISDN Line

The next option for a permanent connection was to use an Integrated Services Digital Network (ISDN) line. This would provide a bandwidth of 128 kbps. This is more than ample since the worse case scenario of one packet sent each time step at an update rate of 30 Hz requires a bandwidth of 38.4 kbps; however, there are two main disadvantages. First there is the cost involved in such a connection. Although the monthly rental fee is only around $50 per month, there is the acquisition cost of ISDN capable routers for both ends which are approximately $2000 each. The second concern was time. Due to time constraints, it was not feasible to have the ISDN line installed during the time frame of this thesis. However, it is likely that if UTIAS continues its work with DCIEM the installation of an ISDN line will occur.

8.3 PPP Connection

The final option for the connection was to use conventional modems to establish the link. It was proposed to establish a Point to Point Protocol (PPP) connection using
Internet. It acts as an invisible layer routing packets over the phone lines requiring no modification to existing network communication algorithms. The cost of this system was minimal since the modems and computers assigned to act as the routers were already in DCIEM's inventory. The time required to setup the PPP was modest, as it took approximately two weeks for DCIEM to implement.

The main concern with a PPP connection is the limited bandwidth. Even though the results discussed in Chapter 5 suggest that the system should be able to keep up with the number of Entity State PDUs transmitted, the limitation of the bandwidth implies a time delay in the transmission of data. If we assume no overhead on the link, then a total of 22 packets can be transmitted a second in each direction. This implies a one way transmission time of 45 milliseconds. Whether or not this would have caused a problem was unknown at the time.

8.4 Implementation of Link

For this discussion, the PPP gateway at UTIAS is referred to as "the client" and "the host" refers to the gateway at DCIEM. The PPP connection used standard telephone lines in the lab. A 486 PC acted as the gateway to direct packets across the link. The software was set up so that only the client could dial up and establish a connection to host at DCIEM. To prevent other machines outside of the lab from using the PPP link, the lab's tie to the Internet was disconnected and replaced by a connection to this gateway PC. The final modification was a change to the default gateway that RISC1 used to communicate with computers outside the LAN. Thus, when a request was made by the
the client machine. Once this data was received, the client created the connection to the host and forwarded the packets on. The connection remained open until a predetermined amount of inactivity had passed. The default is currently 10 minutes.

During testing of the link one problem became evident. A PPP connection provides bi-directional communication across the Internet in a limited fashion in that all communication must be initiated by the client. For example, consider the Telnet service. The client must start the Telnet application and then all data associated with Telnet is allowed to pass either way across the PPP link. The host cannot Telnet into the client because it did not initiate the communication. The same is true with the DIS application. The network communication algorithms are currently designed so that each simulator initiates its own connection to the other simulator. Consequently, the connection from UTIAS to DCIEM works properly since the client initiates the connection to DCIEM. However, the ship simulator was unable to create a connection because it ran on the host side. To work around this, it was first necessary to ensure the link between UTIAS and DCIEM was established. Then the destination address of the packets going from DCIEM to UTIAS is changed. It is set to the host machine instead of RISC1. The PPP link then believes that the link has already been initiated and allows the DCIEM/UTIAS link to function correctly.

8.5 Results

As with the Internet connection, the ping command was used to gather data on how effective the PPP connection was. The results are shown in Figure 8.2. It is clear
However, five times during the six minute testing period, a large slowdown did occur. It was unclear if this would cause any serious problems in the DIS exercise. The other area of concern is the time for the round trip. If the PPP link is assumed to have no overhead, it would be expected that the transmission of a 64 byte packet used by Ping would complete the round trip in approximately 35 milliseconds. It was not expected that the PPP connection could actually meet this target, but to take five times longer than the theoretical value shows there must have been considerable overhead. Using these results it can be shown that only approximately 11 Entity State PDUs can be transmitted per second. Although for the hover task there is still a large margin before saturation of the bandwidth occurs, it is a definite area of concern considering the thresholds were reduced while attempting to diminish the trajectory error.

As a final point, DCIEM has brought up the possibility of replacing the modems used with the higher speed US Robotics 56 kbps modems. This would almost double the available bandwidth for the exercise. If bandwidth problems did occur later on in the evaluation of the link, then this may need to be implemented.
Figure 8.1: Ping Statistics between IRIS5 and SIM2 via the Internet

Figure 8.2: Ping Statistics between RISC1 and SIM2 using PPP Link
Implementation of Full DIS Interface

The final stage in building the DIS interface was to upgrade the DIS algorithms from the very basic structure to a level which would comply with the IEEE standard (1996). This involved the following steps:

i. creating the ability to receive and send Simulation Management PDUs,

ii. implementation of correct handshaking to initialize and terminate a DIS exercise,

iii. time stamping of all PDUs and the synchronization of system clocks, and

iv. the conversion of all data transmitted to the correct coordinate systems.

9.1 Simulation Management PDUs and Handshaking

The additional PDUs were as follows: Create Entity, Remove Entity, Start / Resume, Stop / Freeze, and Acknowledge. The function and structure of these packets is discussed in Chapter 3. The implementation of their structures is in the file dis_pdu.h. The PDU header file from NPSNET did not contain a complete definition of the Simulation Management PDUs so it was necessary for the author to implement the structure in IEEE (1996). The collision PDU has already been discussed, but it was not implemented in this project. This was left as an item for future work.

Sending and receiving these new packets required modification of the communication algorithms in the file dis_comms.c. Each new PDU required an individual algorithm to send it to the other simulation application. However, the receiving function
that the algorithms used to send Entity State PDUs were also used to send the Simulation Management PDUs. This is inconsistent with the discussion in section 3.7. According to the standard, the simulation management PDUs should use TCP and the Entity State PDUs UDP. It was decided that since the DIS interface was still in the experimental phase, all packets would use UDP until a requirement for TCP existed. It should also be noted that the information these packets were supposed to contain, especially the time to start or stop an entity, was not used. Instead the arrival or transmission of the packet signaled the event. It was not necessary to use this time information because if one simulator left the virtual environment, the exercise ceased to exist. Again, if the size of the DIS exercise is expanded, this data will have to be included.

With the ability to send and receive simulation management PDUs, it was possible to implement handshaking. The different levels of handshaking were described in section 3.6. It was decided for this application that the simplest form of handshaking would be sufficient. Figures 3.3, 3.4, and 3.5 show this type of interaction. Note that in order to use this level of handshaking, it is necessary to pre-arrange all initial conditions for each DIS exercise.

The final item with respect to handshaking was the choice of Simulation Manager (SM). The DIS protocol allows one or more computers in the DIS exercise to be the SM. Since this application only involved two simulators, it was decided that the SM or MASTER would be the helicopter simulator and the ship simulator would be the SLAVE. The ship simulator was started first and then it entered a wait mode looking for a Create Entity PDU from the helicopter simulator. Once this was received, the ship simulator
mode. The helicopter simulator then continued on with the necessary trim calculations for the flight equations. This took approximately 30 seconds. Following this, the Helicopter simulator issued a Start PDU and the exercise began.

There are two methods of shutting down the simulator. The first uses the Remove Entity PDU. When the helicopter simulator receives this from the ship simulator the DIS interface is shut down and the ship as seen by the pilot continues its forward motion based on the last known conditions. However, the heave, pitch and roll channels are set to zero. If a Stop Freeze PDU is received from the ship simulator the exact same procedure is followed. Finally, if the helicopter simulator is stopped a Stop / Freeze PDU is issued and the simulation is terminated. Note that there is no option for the helicopter simulator to enter a frozen state so this was not implemented as a possible action.

9.2 Time Stamping and Synchronization of System Clocks

It should be noted at this point that the DIS standard lacked a detailed specification for synchronizing the simulator’s clocks. It is absolutely necessary that the simulators are synchronized for the DIS application to function properly. However, the IEEE Standard (1996) devotes only one small section to this topic. Consequently, the following discussion is only partly based on this section, with additional non-standard modifications made by the author to fill the gaps.

Within the protocol, all packets have a storage location for the time the packet was sent. This time stamp is the number of DIS time units past the hour. In conjunction with the time stamp, the Least Significant Bit (LSB) of the number is set to one to signify that
arbitrary point. An absolute time stamp implies the simulation application clocks are
synchronized to Universal Time (UTC). If the clocks are not synchronized then the time
stamp is relative. For this application it is desirable to use an absolute time stamp. The
justification for this choice will follow later in this section.

The UNIX operating system provides numerous methods for obtaining system
time. Unfortunately the majority of these commands either do not return the time to a
high enough accuracy or the format is not close to the DIS standard. In order to obtain a
useful time stamp, a function was written called \textit{currenttime} located in the file
\textit{dis\_comms.c}. This function makes use of three UNIX system calls; \textit{gettimeofday}, \textit{time}, and \textit{gmtime}.

The function \textit{gettimeofday} returns the number of seconds that have passed since
00:00:00 UTC Jan 1\textsuperscript{st}, 1970 and the number of microseconds past the second. To get the
number of seconds that have passed since the hour, it is necessary to know how many
seconds have passed from Jan 1\textsuperscript{st}, 1970 to the beginning of the present hour. This
information is obtained through two calls. The first is to \textit{time} which returns the number
of seconds since 00:00:00 UTC Jan 1\textsuperscript{st}, 1970. (This does not return the number of
microseconds, so the accuracy is not sufficient enough to be used alone.) Next, the output
of \textit{time} is sent to \textit{gmtime} which dissects the time into months, days, hours etc.
Specifically, the algorithm \textit{currenttime} uses the years since 1900, days since the beginning
of the year, and the number of hours since the start of the day. Combining this
information with the results of \textit{gettimeofday}, the number of seconds and microseconds
since the beginning of the hour can be calculated. Converting this output to the DIS
calculating the number of leap years. Consequently, the function will have to be updated in the year 2000. The second limitation is that the algorithm assumes the simulation is started and completed on the same day, UTC time. Depending on the time zone, this could cause a problem even during the day.

The next step related to timing is the synchronizing of the simulation application’s system clocks. Up to this point, all DIS exercises have been run on a LAN where transmission times between simulators are negligible. However, the link between DCIEM and UTIAS is over a Wide Area Network (WAN). The transmission time between the simulators is significant and the delays the Entity State PDUs experience need to be compensated for. Thus it is preferable that the time stamp is absolute. UNIX provides the programmer with two distinct system calls that can be used to synchronize the clocks on different computers. They are the \textit{timed} daemon and the \textit{timeslave} command.

\textbf{9.2.1 Timed Daemon}

\textit{Timed} is a daemon that synchronizes the system clock with the average time of other machines in the LAN who are also running \textit{timed}. Each \textit{timed} network has at least one machine which is designated the master. This computer queries the other machines in the network to calculate the average time of all the system clocks and then sets its clock to this time. It then periodically sends out synchronization messages to other machines in the LAN so they can adjust their clocks. It is also possible to set up the master so that it only trusts its own clock. This is the preferred method if the master is synchronized to a highly accurate external standard.
timed. For this configuration, IRIS5 was set up as a master which trusted no other machines. The reason for this will be explained shortly. Figure 9.1 shows the difference between RISC1 and IRIS5 sampled once an hour. Figure 9.2 is for a sample rate of once every ten seconds. It is clear from both charts that although there are variances in the clock difference, the discrepancies are always less than one time step of 33 milliseconds. Thus it can be concluded that the timed daemon is an effective method of synchronizing system clocks over a LAN. Herein lies the problem. DCIEM and UTIAS are not on a LAN; they are on a WAN. Timed is not designed to work over a WAN, thus a different method of synchronization is required.

9.2.2 Timeslave

Timeslave is a command specific to Silicon Graphics machines and is therefore not available on RISC1. Rather than setting the system clock to the average network time as with timed, the clock is set to the time on the machine that the local machine is slaved to. The primary advantage of this method is that you can timeslave the local machine to a remote one. To account for varying transmission times between the two computers, timeslave incorporates a sophisticated averaging scheme. The only assumption is that the trip a packet takes to the destination is the same time as the trip back.

It was originally hoped that it would possible to timeslave IRIS5 to a machine at DCIEM and then use timed to synchronize RISC1 to DCIEM via IRIS5 which would be set as the master for the LAN. Unfortunately timeslave could not synchronize the clocks to be within a reasonable margin because of the unreliable Internet connection. Figure 9.3
are unknown, but are at least a millisecond which is still provides unacceptable results. The sample rate of the data was 90 seconds for the beginning of the test period in order to initiate the filters used by the timeslave command and then 3.5 minutes thereafter. It is quite clear that the error between the two systems is too large to be used reliably. Consequently, another method was required to synchronize the simulation applications to compensate for transmission delays between the simulators.

9.2.3 WAN Synchronization

Medeuil (1992) describes the possibility of using a Global Positioning Satellite (GPS) card in each computer in the DIS exercise to provide a very accurate synchronized clock for time stamping. The fundamental requirement for GPS is accurate and reliable time so that phase differences between signals received can be used to triangulate the position. To accomplish this, the GPS satellites have atomic clocks, and the receivers have a quartz clock. The receiver clocks are synchronized to the atomic clocks onboard the satellites. Although these quartz clocks are not as accurate as an atomic, for the purposes of a DIS exercise, the accuracy is more than sufficient. However due to the cost of the GPS units it is unlikely that this system would be implemented locally at UTIAS.

A much simpler and less expensive solution is to use algorithms that average the transmission time between simulators and then use a simple formula to calculate an offset to be applied to all incoming packet's time stamps. This is what was implemented in dis_comms.c. The functions are called getoffset_MASTER and getoffset_SLAVE. The algorithms make use of the PDUs defined for simulation management to determine the
used to end it. The Acknowledge PDU is the packet that is used to determine the offset amount.

First, \texttt{t\_start}, the time that the an Acknowledge PDU is transmitted from the MASTER, is recorded. When the SLAVE receives this packet, it returns an Acknowledge PDU. This packet contains the time that it was transmitted by the SLAVE. This is \texttt{t\_mid}. Soon afterward, the MASTER receives this packet, and records \texttt{t\_end}, the time the PDU arrived. The transmission time, \texttt{t\_tran}, is the difference between \texttt{t\_start} and \texttt{t\_end} divided by two. It is assumed that the transmission time between the MASTER and SLAVE is the same as the time to transmit between SLAVE and MASTER. Finally, \texttt{t\_offset} is given by \texttt{t\_start + t\_tran - t\_mid}. Figure 9.4 is a graphical description of this.

The algorithms are run twice, based on the MASTER SLAVE principle. At first, the Helicopter simulator runs \texttt{getoffset\_MASTER} while the ship simulator has already started \texttt{getoffset\_SLAVE}. Once the helicopter simulator determines the offset, the reverse is repeated for the ship simulator so that it can also determine the offset. The results of the offset are displayed similar to the \texttt{ping} command with a minimum, average and maximum value. This provides a check to see how effective the algorithms are. In general the maximum, minimum, and average values are all within 0.1% of each other. There are however two limitations to the procedure. The first is the occasional long transmission time over the PPP link. As discussed in section 8.5, occasionally the time to send a packet across the link is very slow. If this occurs during the offset determining algorithms, the results are erroneous. It is important to check the minimum, maximum, and average values to see if this occurred. If it did, then the exercise must be restarted.
clocks is much larger than it really is because the time resets from 3599 seconds to 0. Also, since the system clocks are not synchronized, the time period over which the offset is not accurate can be quite large. Thus it is important to avoid running an exercise between approximately five minutes to and five minutes after the hour.

9.2.4 Transmission Time Compensation

Once the offset between the two simulation application clocks is known, it is possible to correct the incoming data in the Entity State PDUs to account for transmission time. The first step is to take the time stamp and apply the offset so the time is in synch with the local machine’s clock. Next, the number of time steps between when the data was sent and the current time is calculated. With this value, a predictor is used to take the received data and extrapolate it to the current time. For the time delays experienced in this application, the predictor used the same simple formula used by the dead reckoning algorithms given in Equation 5.1. However if time delays were longer, it would be necessary to use a higher order predictor to retain a reasonable amount of accuracy.

9.3 Coordinate Conversions

The last stage in bringing the DIS interface into full compliance was transforming the x, y, z, φ, θ, and Ψ from GenHel coordinates to the coordinate system used by the DIS protocol. This consisted of two steps. The first was transforming the origin of the GenHel system from the centre of gravity to the centre of the bounding volume. The second was transforming from the GenHel inertial frame to the WGS 84 Earth fixed frame.
where \( \vec{P}_B \) is the vector from the centre of gravity to the centre of bounding volume expressed in GenHel body fixed frame coordinates.

\( \vec{r}_{bvI} \) is the vector from the origin of the GenHel inertial frame to the centre of the bounding volume with respect to the GenHel inertial frame.

\( \vec{r}_{cgI} \) is the vector from the origin of the GenHel inertial frame to the centre of gravity with respect to the GenHel inertial frame.

\( L_{EB} \) is the rotation matrix that transforms from the GenHel body fixed frame to the GenHel inertial frame.

**Equation 9.1 : Centre of gravity to centre of bounding volume transformation equation**

Using Figure 9.5, Equations 9.1 and 9.2 show how to transform the origin from the centre of gravity to the centre of bounding volume and vice versa. The vector \( \vec{P} \) represents the distance from the centre of gravity to the centre of bounding volume. In the body fixed frame this is a constant. To convert this vector to the inertial frame, the rotation matrix \( L_{EB} \) is used. \( L_{EB} \) is defined in Etkin and Reid (1996). This gives Equations

\[
\vec{r}_{cgI} = \vec{r}_{bvI} - [L_{EB}] \cdot \vec{P}_B
\]

**Equation 9.2 : Centre of bounding volume to centre of gravity transformation equation**
For this application, both simulators used the centre of gravity for their reference point. Since the value of \( \bar{P} \) was unknown for both the helicopter and the ship, the values used in the code are arbitrary. Considering both simulators used only the centre of gravity for all calculations and the same values are used at both ends, this did not affect the results.

For a full derivation and statement of an approximate conversion from the GenHel inertial frame to the WGS 84 coordinate system refer to Appendix B. The algorithms are implemented in the functions \( WGS\_Coord\_Conversion \) and \( WGS\_Euler\_Conversion \) located in the file \( dis\_pdu.c \).
Figure 9.1: Synchronization of IRIS5 and RISC1 using *timed*

Sample Rate of Once an Hour

Figure 9.2: Synchronization of IRIS5 and RISC1 using *timed*

Sample Rate of Once Every Ten Seconds
Starting May 4th
Resolution - 3.5 minutes

Figure 9.3: Timeslave Data between HAL and SIM6

![Graph showing error vs time]

Figure 9.4: Offset Determination Algorithm

\[ t_{\text{tran}} = \frac{(t_{\text{end}} - t_{\text{start}})}{2} \]

\[ t_{\text{offset}} = t_{\text{start}} + t_{\text{tran}} - t_{\text{mid}} \]
Figure 9.5: Centre of Gravity / Centre of Bounding Volume Vector Diagram
Chapter 10

Evaluation of Full DIS Exercise with Test Ship Simulator

An in depth analysis of the full DIS compliant interface led to the removal of various bugs from the code. Upon completion, a final test was required before the link with DCIEM's LSO simulator could be attempted. The test was a subjective assessment of the quality of the exercise. The aim was to determine the effect of the PPP connection on the network simulation. For this comparison three trials were conducted. The first run used the original helicopter simulation with the built in ship motion (Configuration One). The second trial used the DIS interface with the ship simulator running on a computer within the flight simulation lab (Configuration Two). The final appraisal ran the test ship simulator at DCIEM and used the PPP connection to link the two simulators (Configuration Three). In general, the trials showed the visual quality of the DIS exercise to be quite similar to the original simulator. The most noticeable difference was increased vibration of the helicopter while in contact with the deck of the ship. However, the amount of extra vibration was not a serious issue and it was felt that the pilot would attribute the quivering to turbulence or other random noise.

The only problem that manifested itself was larger than normal excursions in the periodic channels. This only occurred when the simulators were communicating over the PPP link. Figure 10.1 is a time history of this occurrence in the helicopter simulator's DRM for ship heave. The statistics gathered with the ping command between RISCI and
across the link. It appeared that this intermittent delay, if it occurred at the correct time, caused the ship to make these large deviations. Consider heave motion; the dead reckoning and smoothing algorithms are linear equations and are not able to predict the cyclic motion that heave undergoes. Thus, an update is required to notify the algorithm to change the motion's direction. If that update is delayed, the motion will continue on in its original course. This leads to a large excursion away from the normal periodic behavior.

The solution to the problem was relatively straightforward. A time out was added to the procedure ship_move that would set a flag if a packet had not been received during the last rx_timeout seconds. If this occurred, pitch, roll and heave in the helicopter simulator's DRM for the ship were smoothed back to zero feet or degrees. For x, and y, the accelerations were set to zero and the heading was locked to the last value. Once the next packet arrived, full motion would resume. Although this method inevitably increased the amount of trajectory error, it was far superior to occasionally having the ship drop ten to twenty feet below the helicopter for no apparent reason. However, it was uncertain if this solution would cause serious problems for the LSO watching the helicopter. It was quite possible, since the helicopter simulator's DRM for the ship motion was no longer tracking the received data from the ship simulator, that the helicopter would move right into the deck of the ship if it had landed. It was hoped that since these large excursions are rare events, the discrepancies would go unnoticed. However, if the problem were significant, it would be necessary to improve the quality of the connection through either faster modems or the use of an ISDN line.
Figure 10.1: Time History of Large Ship Heave Excursion
For Helicopter Simulator’s DRM
Chapter 11

Full DIS Exercise Evaluation

11.1 Description of Full DIS Exercise

The final DIS exercise was to link the Sea King Simulator to the LSO Simulator at DCIEM (Configuration Four). As mentioned in Section 4.2.4, the LSO simulator had not been completed. Thus DCIEM modified the author’s test ship simulator to create a fully functional simulator complete with visual display. The ship motion was again provided by the Fredyn program, with Sea State 1 and 5 ship motion data available.

There was a disadvantage to not using the hoped for LSO simulator for the DIS exercise. The algorithms developed by the author to comply with the DIS standard for the helicopter simulator were interfacing with the same algorithms at the ship simulator. It had been hoped that by linking with a simulator that had independently developed DIS interface algorithms, any inconsistencies in the UTIAS implementation of the protocol would be discovered. However, the advantage of this simulator was that all interactions with regard to ship data had already been extensively tested, thus making the final link up simpler.

The only remaining issue was how the pilot and LSO would talk to one another. As a temporary measure for this validation, commercial telephones were used in the respective simulators. Although an unrealistic method of communication, and awkward for the pilot to fly while holding a telephone, it was sufficient for the final evaluation of the DIS interface.
The evaluation consisted of 9 runs. Each trial involved the pilot flying an approach to the ship and then landing on the deck. This trajectory was observed by the LSO who made subjective comments on the quality of the motion and the location of the touchdown point. The pilot made a similar assessment based on his visual display. Finally, motion in all six degrees of freedom for both the ship and helicopter, as well as the value of $\bar{R}$ were recorded for analysis. Through an analysis of this data, the runs provided the following information for an assessment of the created interface:

i. a subjective analysis of the quality of the DIS exercise,

ii. an examination of the smoothness of the ship and helicopter motion,

iii. a subjective assessment of the trajectory error; and,

iv. a numeric evaluation of the trajectory error.

The objective of the first run was to subjectively assess the quality of the DIS exercise and ensure that all motion was sufficiently smooth. Sufficiently smooth implied that the entity motion was visually smooth to the eye and that no abrupt and unnatural motion occurred when the helicopter was in contact with the deck of the ship. Sea state five was used, providing the most dramatic motion upon which to base a subjective assessment. The parameters for dead reckoning and smoothing were set to the optimal values selected in previous chapters. That is $\Delta$position and $\Delta$angle were equal to 0.01 degrees or feet and $p = 72$. The time histories of all degrees of freedom for both the ship and the helicopter were analyzed to ensure that nothing unusual had occurred. As well, these time histories were compared to verify that the trajectory followed by the two
since as stated in section 7.6, the data between the two simulators was not synchronized

The remaining eight runs were designed to quantify the trajectory error experienced in the DIS exercise and ideally discover a relationship between either $p$ or the thresholds and the trajectory error. The three variables that were adjusted for each run were $p$, the sea state, and the dead reckoning thresholds. Since the initial values for $\Delta \text{position}$ and $\Delta \text{angle}$ were already equal, and that the trajectory error analysis was done primarily with the position, the thresholds were treated as one variable. The principal data from each run were the recorded values of $\bar{R}$. The runs and their respective parameters are listed in Table 11.1.

<table>
<thead>
<tr>
<th>Run #</th>
<th>Sea State</th>
<th>$\Delta \text{position}$ [ft]/ $\Delta \text{angle}$ [degrees]</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>0.01</td>
<td>72</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>0.01</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1.00</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1.00</td>
<td>72</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0.01</td>
<td>72</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0.01</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1.00</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1.00</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 11.1: Final Evaluation Runs - Varied Parameters

The expected relationships were as follows. It was believed that as $p$ was increased, the trajectory error would also increase. The reasoning for this is that the longer the smoothing period is, the longer the ship is not following the correct course. A second justification for this belief is that the point in the future that the ship motion is smoothed to is based on predicting forward from the data just received. The further into the future the algorithms predict, the larger the error. The second relationship was
increased the trajectory error would also increase. This was expected because as the threshold is increased, the error between the actual path and the predicted one could be larger. A smaller threshold would also have the benefit of increasing the update rate, thus making the ship trajectory more accurate.

### 11.3 Problems Encountered and Limitations of Evaluation

Upon initial analysis of the collected data, it became clear that there were some problems that needed to be addressed. First, there was an inconsistency with the initial conditions of the helicopter. The starting position and orientation of the helicopter input into the flight equations were identical to the values input into the ship simulator. However, it was not realized until after the evaluation runs were complete that the initial trimming process of the helicopter equations slightly altered these initial conditions. Table 11.2 shows these differences. It was clear that if the trajectory error were small, this discrepancy between the initial conditions would overshadow the results for the entire length of the run. Due to the nature of the DIS protocol, these differences would disappear after the first few packet updates. Thus, it was decided to ignore the first 20 seconds of data collected.
The second problem was related to landing on the deck. Some of the runs had parameters that would provide insufficient smoothness to keep the helicopter on the deck when a jump as discussed in Chapter 7 occurred. Even if this violent acceleration did not occur, the helicopter would still jump around on the deck providing unrealistic motion. Thus, to prevent nonsensical values for $\bar{R}$ being analyzed, it was decided to ignore all data after a point just before touchdown. Since some runs were shorter than others the value of 33.33 seconds was chosen as the cut off point to provide a consistent evaluation. This gave a period of 13.33 seconds for analysis, which is equivalent to 400 data points.

The final comment on the evaluation runs concerns the limitation of the subjective assessment of the trajectory error. First, both the Pilot and the LSO in the evaluation runs were the author and a co-op student at DCIEM. Consequently, they were untrained observers of the deck landing task. Thus, the comments made are perhaps not as accurate as could be obtained by an actual LSO and a Sea King Pilot. Also, a trained pilot would know where to place the helicopter with respect to the deck to line the probe up with the Bear Trap. In addition, the assessment of the trajectory error in the x direction was quite difficult. Along the y axis, the centre line on the deck acts a reference point and in the z, the wheels provide an excellent method of assessing the error. However, there is no

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Trimming</th>
<th>After Trimming</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>-250.0 [ft]</td>
<td>-245.472382 [ft]</td>
</tr>
<tr>
<td>$Y$</td>
<td>-65.0 [ft]</td>
<td>-63.786808 [ft]</td>
</tr>
<tr>
<td>$Z$</td>
<td>-55.0 [ft]</td>
<td>-55.0 [ft]</td>
</tr>
<tr>
<td>$\phi$</td>
<td>0.0 [deg]</td>
<td>-2.744024 [deg]</td>
</tr>
<tr>
<td>$\theta$</td>
<td>2.0 [deg]</td>
<td>3.793808 [deg]</td>
</tr>
<tr>
<td>$\psi$</td>
<td>15.0 [deg]</td>
<td>14.99985 [deg]</td>
</tr>
</tbody>
</table>

Table 11.2. : Helicopter Initial Conditions Before and After Trimming
would probably go unnoticed. Finally, if the parameters for the run were incapable of providing smooth motion, the position of the helicopter as seen by the LSO would be very jumpy. Hence it was difficult to accurately assess exactly where the helicopter was. Consequently, anything but large errors in all three axes would go unnoticed.

11.4 Results

Run #1 provided a platform to assess the smoothness of the ship motion and the quality of the DIS exercise. Figures 11.1 to 11.6 show linear and angular acceleration data for the ship DRM motion produced at UTIAS with the sea state at 5. Also included on each chart is the maximum and average acceleration of the original ship simulator motion at DCIEM and the maximum and average accelerations at UTIAS. Note that the maximum and average accelerations are all calculated from the absolute values of the acceleration data. It is quite clear from the charts that the movement of the ship is reasonably smooth and the accelerations are close to the original motion. For the linear motions, the maximum and average accelerations for the ship motion at UTIAS were always slightly larger than the motion at DCIEM. The reverse is true for the angular acceleration. Even though the linear motions are larger than the original data, it is suggested that the amplitude is small enough that the pilot will assume that the extra motion is normal for this operation and will not have a detrimental affect on the quality of the exercise.

The other major result from the first evaluation run was the subjective assessment of the quality of the DIS exercise. The data obtained was excellent. At UTIAS, the ship
helicopter as seen by the LSO was also smooth and no unusual movement occurred. Comparing visual displays, the helicopter appeared to be in the same spot over the deck and touched down at the same point on both simulators. Consequently, there were no noticeable trajectory errors within the limitations described in section 11.3. Figures 11.7 to 11.12 compare the full ship motion at DCIEM and the ship DRM motion at UTIAS. Again, it is reiterated that this data does not provide accurate trajectory error information; what is does show is that the paths followed by the ship at UTIAS and at DCIEM are very similar. Figures 11.13 to 11.18 present the same information for the helicopter. That is the complicated model of the helicopter simulator at UTIAS and the DRM for the helicopter at DCIEM. In both cases, there are no unusual differences in the motion. The only noticeable errors are in the initial conditions for the helicopter's orientation. This problem has already been discussed in section 11.3.

The subsequent evaluation runs, numbered 2 to 9, were designed to assess qualitatively and quantitatively the trajectory error caused by the use of smoothing algorithms. The numeric results are presented in Table 11.3. The data is the difference between $\ddot{R}$ for the helicopter simulator and that of the ship simulator. The subjective examination of the touchdown point revealed no noticeable trajectory error for all runs. Nonetheless, this statement must be considered under the restrictions stated in section 11.3.

Figures 11.19 to 11.21 are time histories comparing $\ddot{R}$ for the ship simulator and the helicopter simulator for Run # 2. This is the run with the optimal values selected for
previous section. Figures 11.22 to 11.24 present similar results for Run # 4. This is the run which produced the largest trajectory error (see next section). However, in this case the x axis covers the entire set of data recorded since the number of $\vec{R}$ values recorded were much less than in Run # 2. This is due to the threshold being quite large, thus creating a much lower update rate for Entity State PDUs which contain the $\vec{R}$ data from DCIEM.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>72</td>
<td>0.01</td>
<td>1.791794</td>
<td>0.612805</td>
<td>1.807506</td>
<td>1.1247179</td>
<td>0.249138</td>
<td>0.5322925</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>10</td>
<td>0.01</td>
<td>2.886001</td>
<td>1.415083</td>
<td>0.508313</td>
<td>2.0548163</td>
<td>0.4776613</td>
<td>0.1288793</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>10</td>
<td>1.00</td>
<td>4.553444</td>
<td>2.600119</td>
<td>1.924211</td>
<td>4.0139454</td>
<td>0.8512023</td>
<td>0.8675248</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>72</td>
<td>1.00</td>
<td>0.827438</td>
<td>1.916977</td>
<td>0.1607777</td>
<td>0.508816</td>
<td>0.4384886</td>
<td>0.0905278</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>72</td>
<td>0.01</td>
<td>0.267322</td>
<td>0.246072</td>
<td>0.115236</td>
<td>0.1425254</td>
<td>0.1103193</td>
<td>0.0428851</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>10</td>
<td>0.01</td>
<td>0.283669</td>
<td>0.197269</td>
<td>0.055378</td>
<td>0.1204604</td>
<td>0.1184477</td>
<td>0.0250934</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>10</td>
<td>1.00</td>
<td>0.218514</td>
<td>0.448309</td>
<td>0.201469</td>
<td>0.1027184</td>
<td>0.146252</td>
<td>0.0482083</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>72</td>
<td>1.00</td>
<td>0.36091</td>
<td>0.823464</td>
<td>0.243264</td>
<td>0.1547049</td>
<td>0.1293225</td>
<td>0.0331931</td>
</tr>
</tbody>
</table>

Table 11.3: Trajectory Error Results

A final data source was the subjective assessment of the smoothness of the entity motion. The ship motion viewed at UTIAS was always smooth when $p$ was equal to 72. Changing the thresholds had no noticeable affect. However, the helicopter motion viewed by the LSO at DCIEM was very jumpy when the thresholds were at 1.0 foot and 1.0 degree. This was expected since no smoothing was done on the incoming helicopter data at DCIEM.
It has already been stated that the amount of acceptable trajectory error should at least be smaller than the size of the Bear Trap. Thus, a trajectory error larger than 3 ½ feet is unacceptable. It appears from Table 11.3 that Run # 1, with the optimal values based on the research in Chapter 7, still fulfilled this criterion. In fact all the runs except # 4 met this requirement. Since the subjective assessment of each run revealed no noticeable trajectory error, it could be said that the average errors listed in Table 11.3 are not noticeable and thus acceptable. However, Table 11.4 contains the trajectory error just before touchdown for each run. These values are smaller than the average values listed in Table 11.3. In addition, Run # 4 which contained the highest error at touchdown of 3.089 ft. is one of the cases where the helicopter was thrown immediately off the deck upon touchdown. Thus, the subjective assessment of the error was quite difficult to make. Consequently, the only conclusions that can be drawn from this is that errors of approximately one foot are not noticeable to the untrained eye. Whether or not these values would be problematic for a realistic training exercise is unknown.

<table>
<thead>
<tr>
<th>Run</th>
<th>S. State</th>
<th>P</th>
<th>delta</th>
<th>Abs Rx Difference</th>
<th>Abs Ry Difference</th>
<th>Abs Rz Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>72</td>
<td>0.01</td>
<td>0.350182</td>
<td>0.383862</td>
<td>0.455967</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>10</td>
<td>0.01</td>
<td>1.189621</td>
<td>0.606568</td>
<td>6.751709</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>10</td>
<td>1.00</td>
<td>3.089146</td>
<td>0.733575</td>
<td>0.935883</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>72</td>
<td>1.00</td>
<td>0.468765</td>
<td>0.259681</td>
<td>0.087055</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>72</td>
<td>0.01</td>
<td>0.033764</td>
<td>0.026036</td>
<td>0.004465</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>10</td>
<td>0.01</td>
<td>0.084843</td>
<td>0.090865</td>
<td>0.016613</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>10</td>
<td>1.00</td>
<td>0.136658</td>
<td>0.139089</td>
<td>0.036154</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>72</td>
<td>1.00</td>
<td>0.029068</td>
<td>0.132161</td>
<td>0.061365</td>
</tr>
</tbody>
</table>

Table 11.4. Trajectory Error at Touchdown

Continuing the discussion on trajectory error, the next stage was to determine two possible relationships; between \( p \) and the trajectory error and between the dead reckoning
the data in Table 11.3. Figures 11.25 to 11.29 have the thresholds Δposition and Δangle held constant at 0.01 [feet/degrees] and 11.30 to 11.32 have them at 1.0 [feet/degrees]. It was originally suggested that as \( p \) was increased the trajectory error would also increase. However, from Figures 11.25 to 11.32, no clear relationship stands out. In most cases the trajectory error in two degrees of freedom will decrease with increasing \( p \) and one will increase or the exact opposite will occur with two increasing and one decreasing. Thus, no solid conclusions can be made about the relationship between \( p \) and trajectory error.

The second relationship of interest is that between the dead reckoning thresholds and the trajectory error. Figures 11.33 to 11.40 graphically represent this relationship with \( p \) held constant. Again, as with the previous discussion, there are no clear relationships. With a Sea State 5 and \( p = 10 \), the trend was increased Δposition implies increased error. However, when \( p = 72 \), the trend was the exact opposite. The rest of the charts reveal different behavior for different degrees of freedom with no recurring trend. Thus, as with the previous relationship, nothing noteworthy can be concluded.

It is unfortunate that the final evaluation runs did not provide information on the relationships between \( p \), the thresholds and the trajectory error. If it is found that the amount of trajectory error is unacceptable, it would be useful to know what values need to be modified so that the error could be reduced. However, it was possible to quantify the trajectory error that existed in the DIS exercise; it can be concluded that, to the untrained eye, this error is not significant and thus acceptable.
Figure 11.1: Ship X Acceleration

Figure 11.2: Ship Y Acceleration
Figure 11.3: Ship Z Acceleration

Figure 11.4: Ship Phi Angular Acceleration
Figure 11.5: Ship Theta Angular Acceleration

Figure 11.6: Ship Psi Angular Acceleration
Figure 11.7: Comparison of Ship Motion - X

Figure 11.8: Comparison of Ship Motion - Y
Figure 11.9: Comparison of Ship Motion - Z

Figure 11.10: Comparison of Ship Motion - $\phi$
Figure 11.11: Comparison of Ship Motion - $\theta$

Figure 11.12: Comparison of Ship Motion - $\psi$
Figure 11.13: Comparison of Helicopter Motion - X

Figure 11.14: Comparison of Helicopter Motion - Y
Figure 11.15: Comparison of Helicopter Motion - Z

Figure 11.16: Comparison of Helicopter Motion - ϕ
Figure 11.17: Comparison of Helicopter Motion - θ

Figure 11.18: Comparison of Helicopter Motion - ψ
Figure 11.19: Time History Comparison of $R_x$, Run # 2

Figure 11.20: Time History Comparison of $R_y$, Run # 2
Figure 11.21: Time History Comparison of $R_z$, Run # 2

Figure 11.22: Time History Comparison of $R_x$, Run # 4
Figure 11.23: Time History Comparison of $R_y$, Run # 4

Figure 11.24: Time History Comparison of $R_z$, Run # 4
Figure 11.25: Trajectory Error - Maximums, Sea State 5, Δposition = 0.01 ft

Figure 11.26: Trajectory Error - Averages, Sea State 5, Δposition = 0.01 ft
Figure 11.27: Trajectory Error - Maximums, Sea State 1, Δposition = 0.01 ft

Sea State 1, Delta = 0.01

Figure 11.28: Trajectory Error - Averages, Sea State 1, Δposition = 0.01 ft
Figure 11.29: Trajectory Error - Maximums, Sea State 5, Δposition = 1.00 ft

Sea State 5, Delta = 1.0

Figure 11.30: Trajectory Error - Averages, Sea State 5, Δposition = 1.00 ft
Figure 11.31: Trajectory Error - Maximums, Sea State 1, Δposition = 1.00 ft

Sea State 1, Δ = 1.0

Figure 11.32: Trajectory Error - Averages, Sea State 1, Δposition = 1.00 ft
Figure 11.33: Trajectory Error - Maximums, Sea State 5, $p = 10$

Figure 11.34: Trajectory Error - Averages, Sea State 5, $p = 10$
Figure 11.35: Trajectory Error - Maximums, Sea State 1, $p = 10$

Figure 11.36: Trajectory Error - Averages, Sea State 1, $p = 10$
Figure 11.37: Trajectory Error - Maximums, Sea State 5, $p = 72$

Figure 11.38: Trajectory Error - Averages, Sea State 5, $p = 72$
Figure 11.39: Trajectory Error - Maximums, Sea State 1, \( p = 72 \)

Figure 11.40: Trajectory Error - Averages, Sea State 1, \( p = 72 \)
Chapter 12

Future Work

The next essential step in the development of the DIS interface at UTIAS is conversion of the algorithms to operate on the new Helicopter Simulator. This is critical to future work since the simulator discussed in this paper has very limited capabilities. In general, the transition should be straight forward. However, some problems may occur with the sending and receiving of packets. The SIMex-PLUS operating system, which the new helicopter simulators runs on, is very restrictive about what processes are allowed and when they can be executed while a real time simulation is operating. This ensures that the simulation continues to run at the correct time step. Since input/output operations are CPU intensive, they are limited by SIMex-PLUS. Consequently, this may conflict with the sending and receiving of PDUs.

The next stage is a logical extension of the transition to the new simulator. The new simulator system provides crash detection through the visual database. Thus, this ability should be used and Collision PDUs should be added to the list of packets that are communicated back and forth between simulators.

The final addition to the interface is the cable tension for hauling the helicopter down onto the deck. In the present implementation of the helicopter simulation, this is controlled by the pilot operating a switch in the cockpit. In the real world, the tension is controlled by the LSO. It is recommended that cable tension be added as an articulated parameter in the Entity State PDU. The helicopter simulator will notify the LSO through
the helicopter when tension is added. This will leave the simulators in a position where proper landing operations can be simulated completely.

It is clear from the results in Chapter 11, that more research is needed to determine how best to reduce the trajectory error. Obviously, the first step of this investigation would be to determine exactly what level of trajectory error is acceptable for the deck landing task. This information, coupled with a detailed investigation into how $p$ affects the trajectory error, would allow the optimal values for $p$ and the thresholds to be determined so that the trajectory error is at a minimum and smoothness is sufficient. It is recommended that this analysis be delayed until the algorithms have been implemented on the new helicopter simulator. Since the new system should adhere rigidly to the correct time step, one source of error in the analysis will be reduced.

In addition to these upgrades to the interface, it is the author's opinion that if studies into landing operations are to be investigated then a more permanent and higher bandwidth connection should be established between DCIEM and UTIAS. In conjunction with this, a method of synchronizing the system clocks of the computers to an external standard should be established.

There are also two other areas worthy of investigation which relate to the DIS interface, but do not necessarily benefit this application. They are as follows:

i. implement and investigate the integration method of removal of acceleration spikes; and,

ii. the use of a higher order predictor in the smoothing algorithms.
Chapter 13

Conclusions

The original goal of this project was to implement a DIS interface for the helicopter simulator at UTIAS so that it could communicate with the LSO simulator at DCIEM. Although it was not possible to link up with the LSO simulator, an interface was created with a ship simulator at DCIEM that produced satisfactory results for deck landing operations. It was discovered that the two criteria for evaluating the quality of the exercise were the smoothness of the motion and the magnitude of the trajectory error. These criteria could be adjusted by modifying the following:

i. the number of smoothing steps; and,

ii. the dead reckoning thresholds, \( \Delta \text{position} \) and \( \Delta \text{angle} \).

Through the use of cubic smoothing algorithms, which filtered the incoming ship motion from DCIEM, it was possible to avoid large unnatural motions of the helicopter when in contact with the deck. It was found that as \( p \) was increased, the smoothness of the motion increased. From this investigation, the value of \( p = 72 \) was chosen since it gave similar maximum and average accelerations at UTIAS when compared to the motion at DCIEM.

The thresholds chosen for the interface are listed in Table 13.1. It was found during the final evaluation runs that it was necessary to have thresholds this small in order to maintain sufficient smoothness of the helicopter motion at DCIEM. This was necessary since no filtering was carried out at DCIEM.
Table 13.1: Selected Dead Reckoning Thresholds

The trajectory error of the system was regarded as acceptable based on the subjective assessment of an untrained observer. The numeric values are listed in Table 13.2. Unfortunately, the evaluation of the DIS exercise did not reveal any clear relationships between p or the thresholds and the trajectory error. The optimal values for the parameters which give the best compromise between smoothness and trajectory error still require further investigation. Thus, the values have not been modified from those listed above. It is recommended that this additional analysis be delayed until the conversion of the algorithms to the new simulator.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1247179</td>
<td>0.249138</td>
<td>0.5322925</td>
</tr>
</tbody>
</table>

Table 13.2: Trajectory Error for Selected Parameters

The use of a PPP link between the two simulators provided sufficient bandwidth between the simulators. However, the occasional long transmission delays required the addition of an algorithm which returned the ship motion to zero. This motion, which may be dissimilar to the actual ship motion, could cause problems for the visual display at DCIEM. Consequently, it is recommended that a higher bandwidth connection be established between DCIEM and UTIAS.
References


“Enumeration and Bit Encoded Values for use with Protocols for Distributed Interactive Simulation Applications.”, IST-CF-96-17., Institute for Simulation and Training, Orlando, Florida, October 1996.


Appendix A

Derivation of Smoothing Algorithms

Cubic Smoothing Equations

Given,

\[
\begin{pmatrix}
  x_1 \\
  v_1 \\
  t_1
\end{pmatrix}
\quad \text{and} \quad
\begin{pmatrix}
  x_2 \\
  v_2 \\
  t_2
\end{pmatrix}
\]

one can say,

\[
x_1 = \alpha_0 + \alpha_1 \cdot t_1 + \alpha_2 \cdot t_1^2 + \alpha_3 \cdot t_1^3
\]

(1)

\[
v_1 = \alpha_1 + 2 \cdot \alpha_2 \cdot t_1 + 3 \cdot \alpha_3 \cdot t_1^2
\]

(2)

\[
x_2 = \alpha_0 + \alpha_1 \cdot t_2 + \alpha_2 \cdot t_2^2 + \alpha_3 \cdot t_2^3
\]

(3)

\[
v_2 = \alpha_1 + 2 \cdot \alpha_2 \cdot t_2 + 3 \cdot \alpha_3 \cdot t_2^2
\]

(4)

This provides 4 equations with 4 unknowns. If \( t_1 = 0 \), then equations (1) and (2) are simplified to,

\[
x_1 = \alpha_0
\]

(5)

\[
v_1 = \alpha_1
\]

(6)

Using equations (3) to (6) the coefficients \( \alpha_0, \alpha_1, \alpha_2, \alpha_3 \) can be solved for. This yields:

\[
\alpha_0 = x_1
\]

(7)

\[
\alpha_1 = v_1
\]

(8)
Thus for any time \( t = i \cdot \Delta t \), where \( i = 0 \) at time \( t_1 \), one can calculate the position, velocity and acceleration from:

\[
\alpha_2 = \frac{-1}{t_2} \cdot (2 \cdot v_1 + v_2) + \frac{3}{2} \cdot (x_2 - x_1) \quad (9)
\]

\[
\alpha_3 = \frac{1}{t_2} \cdot (v_1 + v_2) + \frac{2}{3} \cdot (x_1 - x_2) \quad (10)
\]

Thus for any time \( t = i \cdot \Delta t \), where \( i = 0 \) at time \( t_1 \), one can calculate the position, velocity and acceleration from:

\[
x = \alpha_0 + \alpha_1 \cdot t + \alpha_2 \cdot t^2 + \alpha_3 \cdot t^3 \quad (11)
\]

\[
v = \alpha_1 + 2 \cdot \alpha_2 \cdot t + 3 \cdot \alpha_3 \cdot t^2 \quad (12)
\]

\[
a = 2 \cdot \alpha_2 + 6 \cdot \alpha_3 \cdot t \quad (13)
\]
Given,
\[
\begin{pmatrix}
  x_1 & v_1 & a_1 & t_1 \\
  x_2 & v_2 & a_2 & t_2
\end{pmatrix}
\]

and \((x_2 \quad v_2 \quad a_2 \quad t_2)\)

one can say,
\[
x_1 = \alpha_0 + \alpha_1 \cdot t_1 + \alpha_2 \cdot t_1^2 + \alpha_3 \cdot t_1^3 + \alpha_4 \cdot t_1^4 + \alpha_5 \cdot t_1^5
\]
(14)

\[
v_1 = \alpha_1 + 2 \cdot \alpha_2 \cdot t_1 + 3 \cdot \alpha_3 \cdot t_1^2 + 4 \cdot \alpha_4 \cdot t_1^3 + 5 \cdot \alpha_5 \cdot t_1^4
\]
(15)

\[
a_1 = 2 \cdot \alpha_2 + 6 \cdot \alpha_3 \cdot t_1 + 12 \cdot \alpha_4 \cdot t_1^2 + 20 \cdot \alpha_5 \cdot t_1^3
\]
(16)

\[
x_2 = \alpha_0 + \alpha_1 \cdot t_2 + \alpha_2 \cdot t_2^2 + \alpha_3 \cdot t_2^3 + \alpha_4 \cdot t_2^4 + \alpha_5 \cdot t_2^5
\]
(17)

\[
v_2 = \alpha_1 + 2 \cdot \alpha_2 \cdot t_2 + 3 \cdot \alpha_3 \cdot t_2^2 + 4 \cdot \alpha_4 \cdot t_2^3 + 5 \cdot \alpha_5 \cdot t_2^4
\]
(18)

\[
a_2 = 2 \cdot \alpha_2 + 6 \cdot \alpha_3 \cdot t_2 + 12 \cdot \alpha_4 \cdot t_2^2 + 20 \cdot \alpha_5 \cdot t_2^3
\]
(19)

This provides 6 equations with 6 unknowns. If \(t_1 = 0\), then equations (14), (15), and (16) are simplified. This leads to the following matrix equation which can be used to solve for the coefficients \(\alpha_3, \alpha_4, \alpha_5\):

\[
\begin{bmatrix}
  t_2^3 & t_2^4 & t_2^5 \\
  3 \cdot t_2^2 & 4 \cdot t_2^3 & 5 \cdot t_2^4 \\
  6 \cdot t_2 & 12 \cdot t_2^2 & 20 \cdot t_2^3
\end{bmatrix}
\begin{bmatrix}
  \alpha_3 \\
  \alpha_4 \\
  \alpha_5
\end{bmatrix}
= \begin{bmatrix}
  x_2 - x_1 - v_1 \cdot t_2 - \frac{1}{2} \cdot a_1 \cdot t_2^2 \\
  v_2 - v_1 - a_1 \cdot t_2 \\
  a_2 - a_1
\end{bmatrix}
\]

(20)

since,

\[
\alpha_0 = x_1
\]
(21)
Equation (20) can be solved using Cramer’s rule to provide the following solution:

\[
\begin{bmatrix}
\alpha_3 \\
\alpha_4 \\
\alpha_5
\end{bmatrix} = \frac{1}{2 \cdot t_2^3} \begin{bmatrix}
20 \cdot (x_2 - x_1) - 4 \cdot t_2 \cdot (2 \cdot v_2 + 3 \cdot v_1) + t_2^2 \cdot (a_2 - 3 \cdot a_1) \\
-30 \cdot (x_2 - x_1) + 2 \cdot t_2 \cdot (7 \cdot v_2 + 8 \cdot v_1) - t_2^2 \cdot (2 \cdot a_2 - 3 \cdot a_1) \\
12 \cdot (x_2 - x_1) - 6 \cdot t_2 \cdot (v_2 + v_1) + t_2^2 \cdot (a_2 - a_1)
\end{bmatrix}
\]

Thus for any time \( t = i \cdot \Delta t \), where \( i = 0 \) at time \( t_1 \), one can calculate the position, velocity and acceleration from:

\[
x = \alpha_0 + \alpha_1 \cdot t + \alpha_2 \cdot t^2 + \alpha_3 \cdot t^3 + \alpha_4 \cdot t^4 + \alpha_5 \cdot t^5
\]

\[
v = \alpha_1 + 2 \cdot \alpha_2 \cdot t + 3 \cdot \alpha_3 \cdot t^2 + 4 \cdot \alpha_4 \cdot t^3 + 5 \cdot \alpha_5 \cdot t^4
\]

\[
a = 2 \cdot \alpha_2 + 6 \cdot \alpha_3 \cdot t + 12 \cdot \alpha_4 \cdot t^2 + 20 \cdot \alpha_5 \cdot t^3
\]

with equations (21) to (24).
Appendix - B

Conversion from GenHel Inertial Frame to an Earth Fixed Frame which Approximates WGS 84

Assume the Earth is represented by a sphere of Radius R. Let x represent the northerly distance over the Earth’s surface, and let y represent the easterly distance. Also, let the Longitude, \( \lambda \) be positive East and the Latitude, \( \phi \) be positive North (Figure 3.1). From this, one can say:

\[ dx = R \cdot d\phi \]  
\[ dy = R \cdot \cos\phi \cdot d\lambda \]  

Next, we assume that the flight equations in GenHel, which are for a flat-Earth, produce \( dx, dy, \) and \( dz \), where \( dz \) is positive down. One can then write an expression for \( \lambda \) and \( \phi \) as follows:

\[ \phi = \phi_o + \int \frac{dx}{R} \]  
\[ \lambda = \lambda_o + \int \frac{dy}{R \cdot \cos\phi} \]

Now, if \( u^E \) represents the velocity in the x direction in the inertial frame, and \( v^E \) represents the velocity in the y direction in the inertial frame, it is true that:

\[ dx = u^E \cdot dt \]
Thus, (3) and (4) can be rewritten as:

\[ \phi = \phi_0 + \frac{x}{R} \quad (7) \]

\[ \lambda = \lambda_0 + \int_0^t \frac{v^E \cdot dt}{R \cdot \cos \phi} \quad (8) \]

where at \( t = 0, x = y = 0, \phi = \phi_0, \lambda = \lambda_0 \)

Given that

\[ \cos \phi = \cos (\phi_0 + \frac{x}{R}) \]

and the trigonometric expansion

\[ \cos (\phi_0 + \frac{x}{R}) = \cos \phi_0 \cdot \cos \frac{x}{R} - \sin \phi_0 \cdot \sin \frac{x}{R} \]

if we assume that \( \frac{x}{R} \) is small, then one can write Equation 9.

\[ \cos \phi \equiv \cos \phi_0 - \sin \phi_0 \cdot \frac{x}{R} \quad (9) \]

If we make the further assumption that \( \frac{x}{R} \approx 0 \), one can write:

\[ \cos \phi \equiv \cos \phi_0 \quad (10) \]

Thus, Equation 8 becomes

\[ \lambda = \lambda_0 + \frac{y}{R \cdot \cos \phi_0} \quad (11) \]
Now, since the GenHel inertial frame, $F_E$, is tangent to the sphere, with the x-axis to the North and the y-axis East, with z pointing to the centre of the Earth one can use the previous development to create a set of equations which can be used to convert from the GenHel inertial frame $F_E$ to Earth Fixed Frame $F_{WGS}$. Give the origin of $F_E$ lies at $(\phi_o, \lambda_o)$ the point $(x, y, z)$ in $F_E$ can be transformed to the point $(\phi, \lambda, \rho)$ in $F_{WGS}$ using the following equations:

$$
\phi = \phi_o + \frac{x}{R} \\
\lambda = \lambda_o + \frac{y}{R \cdot \cos \phi_o} \\
\rho = R - z
$$

(12)

For the reverse, that is $F_{WGS} \rightarrow F_E$, the following are used:

$$
x = (\phi - \phi_o) \cdot R \\
y = (\lambda - \lambda_o) \cdot R \cdot \cos \phi_o \\
z = R - \rho
$$

(13)

If the point $(\phi, \lambda, \rho)$ in $F_{WGS}$ needs to be expressed in Cartesian coordinates $(X, Y, Z)$, the following can be used:

$$
X = \rho \cdot \cos \lambda \cdot \cos \phi \\
Y = \rho \cdot \sin \lambda \cdot \cos \phi \\
Z = \rho \cdot \sin \phi
$$

(14)

To transform the other direction:
So far, this development has assumed a spherical Earth. However, the WGS 84 system is not spherical. To increase the accuracy of this approximation, the Earth can modeled as an oblate spheroid. This implies that the Earth’s Radius, $R$ varies.

To use the oblate spheroid form of the Earth, the expressions already developed apply, except $R = \rho_o$. The expression for $\rho_o$ is derived as follows:

Given the equation for an oblate spheroid (Refer to Figure 3.1):

$$\frac{X^2 + Y^2}{a^2} + \frac{Z^2}{b^2} = 1$$

(16)

and that

$$\rho_o^2 = X_o^2 + Y_o^2 + Z_o^2$$

(17)

$$Z_o = \rho_o \cdot \sin \phi_o$$

(18)

one can combine equations (16) to (18) as follows:
\[
\rho_o^2 = a^2 \cdot (1 - \frac{Z_o^2}{b^2}) + Z_o^2
\]

\[
\rho_o^2 = Z_o^2 (1 - \frac{a^2}{b^2}) + a^2
\]

\[
\rho_o^2 = \rho_o^2 \cdot \sin \phi_o \cdot (1 - \frac{a^2}{b^2}) + a^2
\]

Solving for \( \rho_o \) yields:

\[
\rho_o^2 = \frac{a^2}{(1 - \sin^2 \phi_o \cdot (1 - \frac{a^2}{b^2}))}
\]

(20)

where

\[
a = 6378137m
\]

\[
b = 6356752.4142
\]

(DMA, 1987)

Since Equation (20) is based on the initial conditions of the DIS exercise, \( \rho_o \) needs to be calculated only once per simulation run.
matrix $L_{EB}$ for $F_E \rightarrow F_B$ where $F_B$ is the body fixed frame.

First, the rotation matrix from $F_{WGS} \rightarrow F_E$ is $L_{WE}$. The Euler angles $(\Phi, \Theta, \Psi)$ in $L_{WE}$ are as follows:

$$\Psi = \lambda_0$$

$$\Theta = -\left(\frac{\pi}{2} + \phi_0\right)$$

$$\Phi = 0$$

Therefore,

$$L_{WE} = \begin{bmatrix}
-\sin \phi_0 \cos \lambda_0 & -\sin \lambda_0 & -\cos \phi_0 \cos \lambda_0 \\
-\sin \phi_0 \sin \lambda_0 & \cos \lambda_0 & -\cos \phi_0 \sin \lambda_0 \\
\cos \phi_0 & 0 & -\sin \phi_0
\end{bmatrix}$$

(22)

Now,

$$L_{BW} = L_{BE} \cdot L_{EW}$$

(23)

where $L_{EW} = L_{WE}^T$

Thus the elements of the matrix $L_{BW}$ can be used to solve for the required Euler angles $(\phi', \theta', \psi')$. The solutions, derived from Equation 4.4.3 in Etkin and Reid (1996), are as follows:

$$\theta' = -\sin^{-1}(L_{BW}(1,3))$$

$$\phi' = \tan^{-1}(L_{BW}(2,3)/L_{BW}(3,3))$$

$$\psi' = \tan^{-1}(L_{BW}(1,2)/L_{BW}(1,1))$$

(24)
find the Euler angles that relate $F_E \rightarrow F_B$, the same principles are followed with $L_{BE}$ substituted for $L_{BW}$ in Equation 24 where

$$L_{BE} = L_{BW} \cdot L_{WE}$$

(25)

where $L_{BW} = L_{WB}^T$