DESIGN AND EVALUATION OF STANDARD TELEROBOTIC CONTROL SOFTWARE

THESIS
Kevin P. Anchor, 1st Lt, USAF
AFIT/GE/ENG/95D-01

Approved for public release; distribution unlimited
DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.
DESIGN AND EVALUATION OF STANDARD TELEROBOTIC CONTROL SOFTWARE

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Kevin P. Anchor, B.S.E.E.
1st Lt, USAF

December 1995

Approved for public release; distribution unlimited
ACKNOWLEDGEMENTS

First, I would like to thank my wife, Vicky, for providing support during my time at AFIT. While I worked on this thesis project, she took care of everything else with only a little help from me. I could not have finished this effort without her help. I would also like to thank my thesis advisor, Maj Dean Schneider, for his help, support, and encouragement. He helped make this work interesting and fun for me and valuable for the Air Force. My thanks also to the members of my thesis committee, Lt Col Paul Bailor and Maj Keith Shomper. Their comments helped make this document much better and their questions from before and during my thesis defense allowed me to refine the document as well.

Finally, I would like to extend a special thanks to three men who I respect very much: Lt Col William Hobart, Lt Col Paul Bailor, and Maj Dean Schneider. Without their help and support, I truly would not have been able to complete this thesis effort. I deeply appreciate all of their efforts on my behalf.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>viii</td>
</tr>
<tr>
<td>I. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2. Problem</td>
<td>3</td>
</tr>
<tr>
<td>1.3. Approach/Methodology</td>
<td>3</td>
</tr>
<tr>
<td>1.4. Materials and Equipment</td>
<td>4</td>
</tr>
<tr>
<td>1.5. Overview</td>
<td>4</td>
</tr>
<tr>
<td>II. Literature Review</td>
<td>6</td>
</tr>
<tr>
<td>11. Current Work on Telerobotic Control Architectures</td>
<td>6</td>
</tr>
<tr>
<td>11.1. Software Architecture Using Port-Based Objects</td>
<td>6</td>
</tr>
<tr>
<td>11.1.2. Onika</td>
<td>7</td>
</tr>
<tr>
<td>11.1.3. Unified Telerobotic Architecture Project (UTAP)</td>
<td>8</td>
</tr>
<tr>
<td>11.1.4. Commercial Uses of UTAP</td>
<td>9</td>
</tr>
<tr>
<td>11.2. Real-Time Operating Systems</td>
<td>9</td>
</tr>
<tr>
<td>11.2.1. VxWorks</td>
<td>10</td>
</tr>
<tr>
<td>11.2.2. LynxOS</td>
<td>11</td>
</tr>
<tr>
<td>11.2.3. Real-Time executive for Military Systems (RTEMS)</td>
<td>11</td>
</tr>
<tr>
<td>11.2.4. Proprietary Operating Systems</td>
<td>12</td>
</tr>
<tr>
<td>11.2.5. Chimera</td>
<td>12</td>
</tr>
<tr>
<td>11.3. Summary</td>
<td>13</td>
</tr>
<tr>
<td>III. Methodology</td>
<td>15</td>
</tr>
<tr>
<td>III.1.1. UTAP Philosophy</td>
<td>15</td>
</tr>
<tr>
<td>III.1.2. UTAP Features Not Fully Supported</td>
<td>17</td>
</tr>
<tr>
<td>III.1.3. Problems with the UTAP Specification</td>
<td>19</td>
</tr>
<tr>
<td>III.2. The Pre-UTAP AFIT Architecture</td>
<td>20</td>
</tr>
<tr>
<td>III.3. Architectural Changes Needed</td>
<td>22</td>
</tr>
<tr>
<td>III.3.1. Message-Passing</td>
<td>22</td>
</tr>
<tr>
<td>III.3.2. The Interface Layer</td>
<td>23</td>
</tr>
<tr>
<td>III.3.3. State Information</td>
<td>25</td>
</tr>
<tr>
<td>III.3.4. Mapping Chimera Modules into UTAP Modules</td>
<td>26</td>
</tr>
</tbody>
</table>
## III.4. Implementing the Changes

111.4.1. General Problems in Implementation Phase ............................................. 27
111.4.2. Object Knowledgebase ........................................................................ 29
111.4.3. Operator Input ..................................................................................... 30
111.4.4. Robot Servo Control ........................................................................... 31

## III.5. Testing the Changes ........................................................................ 31
111.5.1. Simulation ......................................................................................... 31
111.5.2. Informal Testing .................................................................................. 32
111.5.3. Formal Testing ................................................................................... 32

## III.6. Measuring Performance .................................................................. 33
111.6.1. Missed Cycles .................................................................................. 33
111.6.2. Step Response .................................................................................... 33

## III.7. Summary ......................................................................................... 35

## IV. Analysis .............................................................................................. 36
111.1. Commanded versus Actual Position Error ............................................ 36
111.2. Analysis .............................................................................................. 40
111.3. Summary ............................................................................................ 41

## V. Conclusions and Recommendations ...................................................... 42
111.1. Conclusions ......................................................................................... 42
111.2. Recommendations .............................................................................. 43
111.2.1. Recommendations for Future Research ............................................ 43
111.2.2. Recommendations to Improve the UTAP Specification ............... 45
111.3. Summary ............................................................................................ 46

## Bibliography ............................................................................................. 48

## APPENDIX A ......................................................................................... 50

## APPENDIX B .......................................................................................... 54

## APPENDIX C .......................................................................................... 56

## APPENDIX D .......................................................................................... 59

## APPENDIX E .......................................................................................... 78
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>III.1</td>
<td>UTAP Architecture (adapted from [6])</td>
<td>16</td>
</tr>
<tr>
<td>III.2</td>
<td>UTAP Overview</td>
<td>17</td>
</tr>
<tr>
<td>III.3</td>
<td>Pre-UTAP AFIT Architecture</td>
<td>21</td>
</tr>
<tr>
<td>III.4</td>
<td>Chimera/UTAP Interface Layer</td>
<td>24</td>
</tr>
<tr>
<td>IV.1</td>
<td>Error Plot for Joint 1 at Nominal Trajectory</td>
<td>37</td>
</tr>
<tr>
<td>IV.2</td>
<td>Error Plot for Joint 2 at Nominal Trajectory</td>
<td>37</td>
</tr>
<tr>
<td>IV.3</td>
<td>Error Plot for Joint 3 at Nominal Trajectory</td>
<td>38</td>
</tr>
<tr>
<td>IV.4</td>
<td>Error Plot for Joint 4 at Nominal Trajectory</td>
<td>38</td>
</tr>
<tr>
<td>IV.5</td>
<td>Error Plot for Joint 5 at Nominal Trajectory</td>
<td>39</td>
</tr>
<tr>
<td>IV.6</td>
<td>Error Plot for Joint 6 at Nominal Trajectory</td>
<td>39</td>
</tr>
<tr>
<td>V.1</td>
<td>Summary of Conclusions</td>
<td>46</td>
</tr>
<tr>
<td>V.2</td>
<td>Summary of Research Recommendations</td>
<td>47</td>
</tr>
<tr>
<td>V.3</td>
<td>Summary of Recommendations for UTAP Specification</td>
<td>47</td>
</tr>
<tr>
<td>D.1</td>
<td>Joint 1 Error for Nominal Trajectory</td>
<td>60</td>
</tr>
<tr>
<td>D.2</td>
<td>Joint 2 Error for Nominal Trajectory</td>
<td>61</td>
</tr>
<tr>
<td>D.3</td>
<td>Joint 3 Error for Nominal Trajectory</td>
<td>62</td>
</tr>
<tr>
<td>D.4</td>
<td>Joint 4 Error for Nominal Trajectory</td>
<td>63</td>
</tr>
<tr>
<td>D.5</td>
<td>Joint 5 Error for Nominal Trajectory</td>
<td>64</td>
</tr>
<tr>
<td>D.6</td>
<td>Joint 6 Error for Nominal Trajectory</td>
<td>65</td>
</tr>
<tr>
<td>D.7</td>
<td>Joint 1 Error for Fast Trajectory</td>
<td>66</td>
</tr>
</tbody>
</table>
D.8. Joint 2 Error for Fast Trajectory ................................................................. 67
D.9. Joint 3 Error for Fast Trajectory ................................................................. 68
D.10. Joint 4 Error for Fast Trajectory ............................................................... 69
D.11. Joint 5 Error for Fast Trajectory ............................................................... 70
D.12. Joint 6 Error for Fast Trajectory ............................................................... 71
D.13. Joint 1 Error for Slow Trajectory ............................................................... 72
D.14. Joint 2 Error for Slow Trajectory ............................................................... 73
D.15. Joint 3 Error for Slow Trajectory ............................................................... 74
D.16. Joint 4 Error for Slow Trajectory ............................................................... 75
D.17. Joint 5 Error for Slow Trajectory ............................................................... 76
D.18. Joint 6 Error for Slow Trajectory ............................................................... 77
List of Tables

| III.1. Standard Chimera Module Functions | ............................................................ | 22 |
| III.2. Baseline System Description | ............................................................ | 26 |
| III.3. Chimera Features That Map Directly To UTAP | ............................................................ | 26 |
| III.4. Predefined Positions | ............................................................ | 32 |
| III.5. Gains Used for Testing and Data Acquisition | ............................................................ | 34 |
| IV.1. Integral Error for Slow Trajectory | ............................................................ | 40 |
| IV.2. Integral Error for Nominal Trajectory | ............................................................ | 41 |
| IV.3. Integral Error for Fast Trajectory | ............................................................ | 41 |
ABSTRACT

This thesis represents the first implementation of a proposed Air Force standard telerobotic control architecture. This architecture was developed by the NASA Jet Propulsion Laboratory and the National Institute of Standards and Technology under contract to the Air Force Materiel Command Robotics and Automation Center of Excellence (RACE) as the Unified Telerobotics Architecture Project (UTAP).

The AFIT Robotics and Automation Applications Group (RAAG) Lab B facility computational structure was redesigned to be compliant with the UTAP architecture. This thesis shows that the UTAP specification to be implementable. However, if the underlying operating system does not support generic message passing, an interface layer must be implemented to access operating system functions.

The UTAP compliant controller implemented the robot servo control, object knowledge base, and user interface components of the specification. The controller performed adequately although there was degradation in the performance as evidenced by increased error during trajectories. We believe this error can be reduced by re-tuning the controller gains.

Further study of the UTAP specification is recommended: additional functions such as external sensor readings should be added; implementation of the specification on different operating systems and robot platforms will prove the transportability of the specification.
DESIGN AND EVALUATION OF STANDARD TELEROBOTIC CONTROL SOFTWARE

I. Introduction

The Robot Institute of America defines a robot as "a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks" [11]. This definition describes the two main advantages that robots offer: variable programmed motion and performance of a variety of tasks. Because an operator can program the robot’s motions, the operator can control what task the robot performs. Programming the robot also results in predictable and repeatable actions by the robot. Robots are also useful because they can perform tasks that are difficult or dangerous for humans to perform. Yet, despite these advantages, robots lack the ability to think and the ability to adapt to new situations as humans can because computational technology is not yet advanced enough to do so [11].

A telerobotic system, or a telerobot, combines the advantages of robots with a human operator’s ability to react and think by including the human in the task. Telerobotic systems can be classified into three broad types. The first type is an operator-controlled system, where the operator uses one or more input devices to control exactly what the robot does. The key idea is that the operator has complete, real-time control over the robot. The second type of telerobotic system is an operator-supervised system in which a control program gives instructions to the robot; in this type of system, the operator does not have direct, real-time control, but he can change the program or stop
the robot at any time. The final type is the shared-control system. The operator of this type of system uses one or more input devices to control certain aspects of the robotic task, while the robot’s controller controls the remaining aspects of the task.

In all of these systems, the robot’s controller receives instructions in the form of input or programs from a human operator and converts these instructions into signals which operate the actual mechanical and electrical parts of the robot. This controller must provide a method to get input from the human operator and a method to provide output or feedback to the user. Some framework must exist so that the operator knows how to provide input and examine output; this framework is called the telerobotic control architecture.

1.1. Motivation

The Air Force uses telerobots to perform critical tasks such as C-5A/B painting and paint stripping, surface cleaning, and sealing and desealing of aircraft fuel tanks. The Robotics and Automation Center for Excellence (RACE), at Kelly Air Force Base, Texas, is attempting to improve the efficiency and productivity of these robots throughout the Air Force by defining an open telerobotics control architecture to be implemented on all Air Force robots. Since this architecture will explicitly define an interface for all functions which a telerobotic controller needs to operate, old functional modules can be replaced by new modules in a plug-and-play type of environment. This easy replacement of modules means that a telerobot which has this type of architecture can easily be changed from one task to another when one set of modules is replaced by other modules that have the same interface. Also, reuse of existing modules will be easier under this type of architecture.
RACE has defined such an architecture, called the Unified Telerobotic Architecture Project, or UTAP [6].

However, the UTAP architecture has not yet been implemented at any Air Force site, so its effect on performance and its ease of implementation are not yet fully understood. Also, existing robotic controllers will have to be redesigned to some extent to become compliant with the UTAP specification, which consists of modules of control services which pass messages to change the state or mode of the system. Since the UTAP specification may become an Air Force standard, the difficulty in converting an existing controller to be compliant with the UTAP specification must be determined. This information will assist the Air Force in determining if the retrofit of existing controllers should be required if the UTAP specification is adopted as a standard.

1.2. Problem

Currently, the PUMA robot in the AFIT Robotics Laboratory has an architecture which consists of modules that execute on a periodic basis and which update the state of the system by directly changing a global state table. In order to examine the UTAP specification and to determine its effect on real-time system performance, we redesigned the software architecture of the PUMA robot in the AFIT Robotics Laboratory to be compliant with the UTAP specification. Current performance is compared with the new architecture's performance through the use of performance metrics.

1.3. Approach/Methodology

I have conducted my thesis research using the following general steps which are discussed further in later sections:
a. Analyzed both the current AFIT architecture and the proposed UTAP architecture. A system description and pictorial representation of the current architecture of the PUMA robot are included in this document.

b. Determined the changes needed to make the non-UTAP compliant system compliant with the specification and designed a system which incorporates these changes. This task was the most important and time-consuming step of the research project.

c. Implemented and tested the changes in an iterative process.

d. Conducted performance measurements and compared the performance of the new system to the old performance through the use of metrics. The metrics used were developed throughout the course of the project as a better understanding of the two architectures was gained.

1.4. Materials and Equipment

All necessary equipment was already available in the AFIT Robotics Laboratory. This equipment included the VMEbus hardware, the PUMA 560 manipulator, the Chimera 3.2 real-time operating system, and the software modules which make up the current architecture for the PUMA manipulator. Also, the existing Sun workstation network, which uses the SunOS 4.1.3 operating system, was used for program development and for collection and analysis of data.

1.5. Overview

This thesis report is divided into five chapters. Chapter I contains background information and is an introduction to the topic. Chapter II is a literature review of current robotic architecture work. Chapter III describes the procedure used to develop the new architecture, while Chapter IV contains my evaluation and analysis of the results. Conclusions drawn from this research and my recommendations for future research are located in Chapter V.
This thesis also has several appendices. Appendix A is a User’s Manual which lists the steps needed to run the UTAP-compliant modules, while Appendix B is a Programmer’s Manual which lists the steps necessary to compile UTAP-compliant modules and other necessary modules under the Chimera Operating System. Appendix C is a list of all UTAP messages which shows what messages were implemented for this project. Appendix D contains the plots for the data collected during performance measurement. Appendix E is a listing of all new source code developed for this project.
II. Literature Review

In this literature review, I present an initial examination of the current work in the field of robotic architectures. These examinations are limited to work which supports the Air Force’s need for an open telerobotic architectural standard. The review discusses the Unified Telerobtic Architecture Project (UTAP) specification, which is being considered by RACE to become the Air Force architecture standard for robotic systems, as well as commercial applications of this specification. This review also motivates the need for my thesis research, which is to investigate the complexity of converting a telerobotic system to be compliant with the UTAP specification and the performance of the converted system.

The review first looks at Chimera and Onika, which are building blocks for an architecture such as the Unified Telerobtic Architecture Project (UTAP) architecture. Then, it discusses the UTAP architecture. Finally, it describes some commercial work on the UTAP project.

II.1. Current Work on Telerobotic Control Architectures

II.1.1. Software Architecture Using Port-Based Objects [10]

Stewart, Volpe, and Khosla describe the need for a real-time operating system which supports module reuse to avoid redeveloping code at great expense when much of the required code is already available. The authors then focus on defining a software framework which allows code to be reused easily. This framework is based on code sections called modules, which have defined communication interfaces. The authors make use of object-oriented software design and port automaton digital control design theory to build the framework. Using these concepts, a framework which allows software module
reuse, hardware-independent interfaces, real-time communication between tasks, and
dynamic, or on-the-fly, reconfiguration of modules is described.

Port-based objects and resource ports are the keys to this framework. Port-based
objects are modules which have input and output ports for real-time inter-object
communication, while resource ports are modules which communicate with hardware
objects such as sensors and actuators. In essence, resource ports function as device
drivers. Because the port-based philosophy requires that the interface of a module be
explicitly defined, modules can be connected together easily. Also, a module can be
replaced by another module as long as the interfaces of the two modules are identical,
regardless of the internal differences.

Another important feature of this framework is that modules can be dynamically
reconfigured, which means that one or more modules can be changed while the real-time
system is operating without affecting the system’s stability.

The authors’ work has been incorporated into the Chimera Real-Time Operating
System, which was also developed at Carnegie Mellon University. Thus, the important
concepts of module reuse, defined interfaces, and changeable tasks are all present in
Chimera. However, their work does not explicitly define the interfaces necessary for
telerobotic systems because Chimera is a general-purpose real-time operating system.
Thus, the work does not completely address the Air Force need for a telerobotic
architecture.

II.1.2. Onika [3]

Building on the foundation of the Chimera operating system, Gertz, Stewart, and
Khosla describe a graphical approach to building reusable software for dynamically
reconfigurable systems. The authors present Onika, their icon-based graphical programming interface, as a software development environment for real-time robotic systems.

Onika graphically depicts modules as icons; the user simply connects icons together to form programs, while Onika ensures that all interfaces match. Any number of modules can be connected together and then treated as a single module, so Onika is capable of forming low-level or high-level connections of software modules.

In keeping with the work done by Stewart, Volpe, and Khosla in [10], Onika allows for dynamic reconfiguration of modules and module reuse. To dynamically reconfigure the system, the user connects the appropriate icon which represents the new module into the system, while Onika handles all of the underlying work. To reuse a module, the user only has to copy the icon representing the module into the system.

Thus, Onika provides all of the advantages described by Stewart, Volpe, and Khosla, and it allows applications to be created by manipulating graphic elements which represent the modules. However, like the previous work, it does not define the interfaces for a telerobotic system.

II.1.3. Unified Telerobotic Architecture Project (UTAP)

Since no previous work completely addressed the Air Force's need, the Robotics and Automation Center for Excellence (RACE) sponsored the Unified Telerobotic Architecture Project (UTAP). The purpose of UTAP is to define a standard telerobotic architecture to be used throughout the Air Force and possibly in the commercial sector as well. The Intelligent Systems Division of the National Institute of Standards and Technology and the Jet Propulsion Laboratory worked jointly on this project [6], [5].
The UTAP document describes the architecture as "a modularized arrangement of control services." This modular arrangement implies an interface. The majority of the UTAP document [6] explicitly defines the interface of each module. Both the software and hardware interfaces for the architecture are defined. See Figure III.1 for an overview of the UTAP architecture. The UTAP specification is explained in detail in the next chapter.


Using the UTAP specification as a guide, Advanced Cybernetics Group, Inc. (AGC) has implemented a programming environment which is compatible with the UTAP specification. AGC's environment is built on the commercially available Adept V+ robotic programming language. In addition to describing the environment's implementation, DaCosta gives examples of commercial and Air Force sites that are using the AGC product to perform tasks with telerobotic systems.

This paper shows that the philosophy of the UTAP document is sound and that commercially available products can be used to implement the UTAP specification; however, the ACG work only implements the Information Model, which is one small part of the UTAP-specification. The main thrust of UTAP, which is standardized interfaces to standard modules across any robotic system, is not implemented.

II.2. Real-Time Operating Systems

A variety of real-time operating systems (RTOS) are available today. Each operating system has several advantages and disadvantages which contribute to its overall usefulness for this thesis project. A basic requirement for an operating system is that it must support the available hardware; several RTOS were eliminated from further
consideration because they do not support the hardware which AFIT uses. This AFIT hardware consists of a Sun SparcStation development environment with Motorola 68000 real-time processors in a VMEbus chassis. Thus, any RTOS considered must support cross-compilation from the Unix environment to Motorola 68000 executable code.

I will discuss several of these operating systems and describe the reasons why each was rejected, or in the case of Chimera accepted, for use in this project.

II.2.1. VxWorks [13]

VxWorks is a commercial product developed by Wind River Systems. According to Wind River Systems, VxWorks is "a high performance, scalable real-time operating system which executes on a target processor; a set of powerful cross-development tools which are used on a host development system; and a full range of communications software options such as Ethernet or serial line for the target connection to the host."

VxWorks is based on a microkernel called "wind", which supports the runtime system and provides intertask communication through a variety of mechanisms such as shared memory or message queues. Both Unix and Windows development platforms are supported, and VxWorks supports many target processors including the Motorola 68000. In addition, VxWorks can be ported to other hardware as needed by using an additional toolkit. C and C++ tools are provided for VxWorks.

The primary advantage of VxWorks is that it is a widely-used commercial product, so it has support from Wind River Systems and from other users on the Internet (comp.os.vxworks is an existing Usenet newsgroup). Also, it supports ANSI-C and POSIX standards. Finally, the system includes a debugger and many other development tools can be purchased to ease development.
However, the primary disadvantage of VxWorks is cost. This disadvantage will be eliminated next year because AFIT will be getting a copy of this OS from Wind River Systems.

### 11.2.2. LynxOS [4]

LynxOS is also a commercial product developed by Lynx Real-Time Systems. LynxOS is also a kernel-based system and was designed to be like Unix from a programmer’s prospective. It can be used in a cross-development environment or the development tools can be run on the target processor. C and C++ languages are supported by the OS, and Ada support is available from a third-party, Alsys, Inc., of Burlington, MA.

LynxOS shares the same advantages and disadvantages as VxWorks. Lynx Real-Time Systems provides support and training (for a fee) and support is also available through the Internet via the comp.os.lynx newsgroup. LynxOS also conforms to POSIX real-time extensions.

The main disadvantage, again, is cost for the OS, support, and training.

### 11.2.3. Real-Time executive for Military Systems (RTEMS) [12]

RTEMS was developed by On-Line Applications Research Corporation under contract to the Research, Development, and Engineering Center of the U.S. Army Missile Command. It supports multi-tasking, priority-based scheduling, rate monotonic scheduling, and intertask communication, as well as other features. There are two versions which both support the same functionality: RTEMS/C and RTEMS/Ada. RTEMS/C supports several different target platforms, while RTEMS/Ada only supports the Motorola 68000 family. Ada95 support is planned for implementation this year.
The advantages of this OS are that it is free and is provided for Dual-Use/Technology Transfer. All documentation and source code for the OS and development tools can be retrieved from the RTEMS World-Wide Web Page. Support is provided by the developer. Also, the ability to use C, Ada, or Ada95 for development is very powerful.

The main disadvantage of this OS is that it has not been used at AFIT. Thus, there would be no baseline system with which the new system performance could be compared. Also, since AFIT does not currently use this OS, we do not know how stable and mature it is.

II.2.4. Proprietary Operating Systems

Proprietary operating systems are mentioned solely because many robotic systems are packaged with their own proprietary OS. For example, the Adept robot in the AFIT Robotics Laboratory uses a proprietary OS and a proprietary programming language called V+.

Each proprietary OS will have its own advantages and disadvantages. In general, though, a proprietary OS will take good advantage of the particular hardware that it executes on; however, existing software will have to be modified or completely rewritten to execute under that OS. In the case of the Adept, any existing software for other robotic systems would have to be rewritten in V+.

II.2.5. Chimera [1]

Chimera was developed at Carnegie Mellon University by David B. Stewart and Pradeep K. Khosla. The Air Force Institute of Technology Robotics Laboratory currently uses the Chimera Real-Time Operating System to control the VMEbus-based Motorola
processors which are used to control the PUMA 560 manipulator. Based on the work of Stewart and Khosla on dynamically reconfigurable software for robotic systems, Chimera supports the Motorola MC680x0 family of processors. Chimera supports "static and dynamic scheduling, extensive error detection and handling, a full set of library utilities, several different multiprocessor communication and synchronization primitives, and a fully integrated host workstation environment." Chimera supports programs written in both C and C++ which are compiled with a modified version of the GNU C compiler, gcc. Chimera can schedule tasks using the rate monotonic algorithm or by using a dynamic priority scheme such as earliest deadline first. The policy of scheduling is separated from the mechanism, as expected in modern operating systems [7], so that the scheduler can be modified by the user.

The advantages of this OS are that it is free and it is the operating system that AFIT currently uses. Because it is already in use, we have working, tested application software that runs on this operating system. Also, Chimera automatically tracks certain performance measurements such as the number of missed cycles for each task. However, since Chimera is not a commercial product, its support is not reliable. Also, we have experienced random errors using this software at AFIT since it is essentially a beta version. Finally, Chimera is not widely used outside of the academic or laboratory setting, so the generality of results obtained from developing software using this operating system may be limited.

II.3. Summary

The Air Force has many telerobotic systems. In order to maximize the productivity and efficiency of these systems, the Robotics and Automation Center for
Excellence (RACE) has sponsored UTAP, a project designed to define a standard interface for telerobotic systems. The work of Stewart, Khosla, Gertz, and others at Carnegie Mellon University has laid a foundation for the UTAP interface, and the work of companies such as Advanced Cybernetics Group, Inc. shows that the UTAP architecture has merit in the commercial sector as well as in the Air Force.

Now that an overview of the UTAP architecture has been given, it must be examined to determine the monetary and time costs of converting current telerobotic systems into systems that are compliant with this architecture. Also, the performance of systems using this architecture must be determined because real-time robotic systems must meet all time constraints.
III. Methodology

This chapter describes the methodology used in this thesis effort. Since the main goal of the thesis effort is to implement and evaluate the UTAP specification, the first topic that is addressed is that specification. First, I describe my understanding of UTAP and its intent. Next, I discuss the pre-UTAP architecture used to control the PUMA robot in the AFIT Robotics Laboratory. Then, I focus on what architectural decisions were made to support the UTAP architecture and how these changes were implemented and tested. Finally, I describe how performance between the pre-UTAP and UTAP architectures was measured and compared.

III.1. The Unified Telerobotic Architecture Project: What Is It?

III.1.1. UTAP Philosophy [6]

The Unified Telerobotic Architecture Project is described in [6]. The intent of the specification can be summed up fairly easily: the purpose of the UTAP specification is to provide an open, system-independent description for building telerobotic applications quickly, cheaply, and easily. Just as the C language is fairly independent and transportable among many different operating systems, a UTAP-compliant robotic application should be transportable to many different robotic systems with little effort to get the application working on a different computer system, operating system, or robot. Other goals of the specification, such as code reusability, abstraction, and understandability, line up with the goals of software engineering.

UTAP has an object-oriented philosophy. Each module has defined inputs, outputs, and responsibilities, and all data inside of a module, or "object", is self-contained and hidden from other modules. Data that is needed by multiple modules or data that is
considered to be "global" is stored by the *Object Knowledgebase*, which is itself just another module. Data and control flow is passed between modules through the use of predefined messages. Appendix C contains a list of these messages. Thus, the modules which make up the system and the interface between these modules is explicitly defined.

![UTAP Architecture Block Diagram](image)

**Figure III.1. UTAP Architecture (adapted from [6])**

Figure III.1 shows the overall UTAP architectural block diagram. Each box represents a module and each line represents a communication channel. Arrows on the line show which way communication can occur. Communication can only occur between modules that are connected together in the block diagram.

For any particular application, not all of the boxes, or modules, need to exist. Likewise, some modules may have multiple instances. A configuration file specifies what
modules compose the system. For example, if the *Object Modeling* module is not needed for some task, then the corresponding configuration file would indicate the absence of that module from the system.

Figure III.2 shows at a more general level what the UTAP specification defines for each module. The inputs, outputs, and responsibilities of each module from Figure III.1 are defined. Likewise, the communication channels and the messages which can be passed over those channels are also explicitly defined.

**Figure III.2. UTAP Overview**

III.1.2. UTAP Features Not Fully Supported [6]

Although the main philosophy is as described above, the UTAP specification has several other parts. The UTAP Information Model allows the workspace, or the three dimensional area in which the robot can operate, to be described using geometric shapes and patterns, or motions within the workspace. The Information Model is not considered in this thesis project.
Another aspect of the specification is that multiple instances of some module types may exist in a system. For example, there may be more than one Tooling Control module or more than one Robot Servo Control module. The UTAP specification proposes that a system configuration file describes what modules are present in the system and how many instances of each module exist. However, the format and operation of this configuration file are not yet defined. To simplify this project, I did not choose to use a system configuration file. Instead, I require the user to manually load the correct system configuration. Note that a Chimera program which automatically loads and executes all needed modules can be written. Such a program would eliminate the need for the user to manually load all necessary components every time the UTAP application is to be executed.

Each individual module also requires a configuration file; however, the UTAP specification does not define a format for this file. Because Chimera also requires that each module has a configuration file of a specific format which contains specific data, I chose to use the Chimera configuration file as the UTAP configuration file. Chimera allows local, or user-specific, data to be added to the file and the operating system provides services to easily read the files.

Another feature of UTAP that is not supported in my work is the notion of each module "posting" what messages it will respond to. Upon initialization, each module is expected to post to the system Object Knowledgebase what messages that it can respond to. Before another module sends a message to that module, it can check the Object Knowledgebase to determine if the message will be responded to. To simplify this project, this feature is not supported.
III.1.3. Problems with the UTAP Specification

Although the UTAP specification is a very good attempt to develop an open telerobotic system, it suffers from several weaknesses. The major problem is that the specification is very general so it can be applied to many different systems; however, this generality means the specification does not define or describe many items in enough detail.

The biggest problem is that the specification does not completely define the interface between modules. All available messages are listed, but the messages are not fully defined. Additionally, the semantic meaning of some messages is open to interpretation, and how to pass data is not defined. The current version of the UTAP specification purposely ignores the how-to-pass issue in order to simplify the specification. However, I see this as a major issue because the method of passing data will greatly influence whether a UTAP-compliant system will be portable. For example, one module which uses pointers, or pass by reference, will not work with another module which uses pass by value. Thus, the plug-replacement nature of modules may not work as expected.

Although each module’s inputs, outputs, and responsibilities are defined, the definitions are ambiguous and vague. For example, the Object Knowledgebase is defined by the following:

RESPONSIBILITY: Store information about objects in the task environment including geometry and task information.
INPUT: Object Information
OUTPUT: Object Information

From this definition, I decided to use the Object Knowledgebase as a repository for all global data; however, another implementor may interpret this definition differently.
As stated above, the UTAP specification also requires system and module configuration files, but the format and content of these files is not fully defined. Thus, different implementations may use different file formats. If this occurs, some of the portability of the system will be lost.

Finally, "conformance" to the UTAP specification is not well-defined. Clearly, a UTAP-compliant system does not have to support all messages associated with a particular type of module, or else the concept of "posting" would not be needed. So, how many messages must a module support to be considered UTAP-compliant? A definition for compliance or conformance to the specification needs to be specified in the UTAP document to clear up this question.

III.2. The Pre-UTAP AFIT Architecture

The current architecture at AFIT is based on the Chimera Dynamically Reconfigurable Module standard developed by Stewart [9]. An overview of the architecture is presented in Figure III.3. Chimera modules are executed cyclically, and system state data is maintained in a Global State Variable Table, or GSVT. At the beginning of each cycle, any variables needed by the module are copied from the GSVT into the local state variable table for that module. The module then executes until it becomes blocked for any reason such as waiting for I/O. When the module completes execution, the variables updated by the module are copied from the local state variable table back into the GSVT. The operating system handles locking so that concurrent access to the GSVT does not cause any problems. The data stored in the GSVT is defined by a system-wide configuration file. The local state variable table for each module is defined by declaring a structure inside the module. Thus, the local state variable table can
contain any data that is needed by the module; it does not have to only contain a subset of the values stored in the GSVT. Each module also has a configuration file, called an RMOD file, which defines what variables will be copied to and from the GSVT, the frequency at which the module should be executed, the name of the module, and any other information needed by the module.

Every Chimera module must follow a specific format. If the module is named *ModuleName*, then it will have several functions of the following form: *ModuleNameXXX*. Table III.1 lists the functions that are important for this project and specifies what role each function plays. The most important fact to note is that these modules are run by the Chimera operating system whenever the associated event occurs. So, whenever a module is spawned by the user, the *ModuleNameInit* function is executed. Similarly, the *ModuleNameCycle* function is executed at the rate specified in the RMOD file.
III.3. Architectural Changes Needed

Once the UTAP specification and the existing AFIT architecture were understood, the next step was to determine what changes were needed to develop a UTAP-compliant architecture in the AFIT Robotics Laboratory. Each of the major architectural design decisions is discussed below.

III.3.1. Message-Passing

The first major design decision to be made was how to implement the UTAP message-passing scheme. Since messages can pass data or control flow, the obvious choice was to implement the messages as procedure or function calls. The C language and the Chimera operating system both support this option. The advantages of this option were that data could be passed by value or reference very easily, the built-in support for this implementation, and the ease with which new messages, or functions, could be implemented. The disadvantage of this method is that the system performance would be hurt because of the context switches which would occur each time a message was sent and control flow passed to another task.

Another alternative considered was to implement true messages which would be sent from one task to another. This scheme would probably contribute to better system performance than would the function-based method discussed above since messages

<table>
<thead>
<tr>
<th>Module Name</th>
<th>When it is executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module_NameInit</td>
<td>When the module is SPAWNed</td>
</tr>
<tr>
<td>Module_NameOn</td>
<td>When the module is turned ON</td>
</tr>
<tr>
<td>Module_NameCycle</td>
<td>Executed periodically at a set frequency whenever the module is ON</td>
</tr>
<tr>
<td>Module_NameOff</td>
<td>When the module is turned OFF</td>
</tr>
<tr>
<td>Module_NameError</td>
<td>When an error occurs</td>
</tr>
</tbody>
</table>

Table III.1. Standard Chimera Module Functions
would not cause the currently executing task to block unless a reply was needed. Chimera supports this scheme with Interprocessor Message Passing [9]; however, using this feature of Chimera would make the UTAP implementation dependent on the Chimera Operating System, thus violating one of the major philosophies behind UTAP.

Based on the discussion above, I decided to implement the function-based scheme. The ease of implementation and C language support independent of operating system provided enough advantage to choose this method.

III.3.2. The Interface Layer

The next problem to be faced was how to make the UTAP implementation completely operating system independent. Unfortunately, Chimera forces modules to be written with a predefined structure (discussed in Section III.2). Chimera schedules the Module_NameCycle function of each module to run at the appropriate time, while the UTAP architecture requires that a UTAP message start and stop each module. The approaches seem to be almost diametrically opposite from each other; I needed to find a way to bridge the two different architectural approaches.

My solution was the Chimera/UTAP Interface Layer, or CUIL (pronounced "cool"). Figure III.4 illustrates the CUIL. Essentially, the CUIL bridges the gap between Chimera and UTAP. The CUIL is a Chimera module which conforms to all of the format requirements of any other Chimera module. Using UTAP messages, which are function calls, it invokes and passes data to a UTAP-compliant module. The UTAP-compliant module receives messages and communicates to the CUIL and to other UTAP-compliant modules using UTAP messages. In other words, the UTAP-compliant module does not
know or care what other module is acting on the messages; all that matters is that the messages are responded to correctly.

The CUIL module handles all operating system-dependent tasks, such as communication with devices and the robot. When Chimera invokes the CUIL Module_Name_Init routine, any Chimera-specific tasks are completed, and then the UTAP module’s initialization routine is invoked through the use of a UTAP message.

![Diagram of Chimera/UTAP Interface Layer]

**Figure 11.4. Chimera/UTAP Interface Layer**

The CUIL scheme allows the UTAP-compliant modules to be completely operating system independent, and therefore portable. It is important to note that other operating systems may also require an interface layer to address specific requirements of the particular operating system. Thus, a major portion of this thesis effort was invested in the design and development of the interface layer.
III.3.3. State Information

As discussed in the above sections, Chimera stores all information in the Global State Variable Table, or GSVT. UTAP specifies a module called the Object Knowledgebase to be a repository for task information.

Since these two items seem to be a good match, one design approach was to allow a UTAP-compliant module to access the GSVT directly. All external modules would consider the Object Knowledgebase to be another UTAP module because of its external UTAP interface, but internally it would be dependent on the Chimera Operating System. Although the intent of UTAP is violated somewhat with this approach, it does provide a very efficient, easy to implement approach because Chimera automatically provides the mechanisms for concurrent access to the GSVT.

A second approach would be to not use the Chimera GSVT and to instead implement the Object Knowledgebase module with its own internal state table. Though this could be done fairly easily, the issue of concurrent access by multiple tasks could become a problem. Also, interprocessor communication in Chimera is handled through the use of the GSVT.

I chose to implement the first option. The main reason to use the GSVT was so that UTAP modules could be executed at the same time as non-UTAP modules. Any regular Chimera modules would require the GSVT, so it would have to be present and in use in order to execute these modules. Since it had to be there, the best choice was to use it and accept the portability limitations on the Object Knowledgebase.
III.3.4. Mapping Chimera Modules into UTAP Modules

The next phase of the design of the new architecture focused on mapping Chimera modules into the predefined UTAP modules shown in Figure III.1. The baseline system is described by Table III.2 and Table III.3. Table III.2 shows what Chimera module was mapped to each UTAP module. Table III.3 shows how internal features of the Chimera OS have the functionality of certain UTAP modules. This system was chosen because it is a simple telerobotic task.

<table>
<thead>
<tr>
<th>Chimera Module Name</th>
<th>Purpose</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Maps To UTAP Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>jtrackball</td>
<td>gets the position commands from the trackball and converts them to joint position commands</td>
<td>none</td>
<td>Q_REF (Reference joint positions)</td>
<td>Operator Input</td>
</tr>
<tr>
<td>PUMA_pidg</td>
<td>proportional, integral, derivative (PID) controller for the PUMA manipulator. This controller eliminates steady-state error in the commanded joint positions. (joint-level)</td>
<td>Q_REF (Reference joint positions) Q_MEZ (Measured joint positions) T_GRAV (gravity compensation torques)</td>
<td>Q_MEZ (Measured joint positions) Q_MEZ (Measured joint velocities)</td>
<td>Robot Servo Control</td>
</tr>
<tr>
<td>grav_comp</td>
<td>calculates the torques needed at each joint to compensate for gravity (joint-level)</td>
<td>Q_MEZ (Measured joint positions) T_GRAV (gravity compensation torques)</td>
<td>T_GRAV (gravity compensation torques)</td>
<td>Robot Servo Control</td>
</tr>
</tbody>
</table>

Table III.2. Baseline System Description

<table>
<thead>
<tr>
<th>Chimera Function</th>
<th>Maps To UTAP Module(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global State Variable Table</td>
<td>Object Knowledgebase</td>
</tr>
<tr>
<td>User Interface</td>
<td>Status and Graphical Display</td>
</tr>
<tr>
<td>Scheduler</td>
<td>Analysis and Diagnosis and Task Program Sequencing</td>
</tr>
</tbody>
</table>

Table III.3. Chimera Features That Map Directly To UTAP
III.4. Implementing the Changes

The process of implementing the architectural decisions took much longer than I expected. The Object Knowledgebase was picked to be the first module to be implemented as a UTAP-compliant module because it is needed by all other UTAP-compliant modules. The *jtrackball* module was implemented next because of its simplicity relative to the other modules. During coding of the first two modules, most of the problems were identified and overcome. These problems and their resolution are presented below.

III.4.1. General Problems in Implementation Phase

Most of the problems encountered were related to my initial unfamiliarity with the C programming language and the Chimera operating system. I was quickly able to learn about them, but some of the fine points had to be learned through trial and error. Also, figuring out how to use the gicc compiler and what makefiles had to be changed to recompile the system were other tasks which caused problems during this stage.

Pointers, which are used extensively throughout the code, were another source of problems. The first pointer-related problem was just ensuring that the correct pointers were being passed from one module to another. The second problem was much more insidious and caused a major delay in getting any working code. The system would lock up randomly, but always when I was using the gravity compensation UTAP module. In the module, I had declared a structure as follows:
typedef struct {
    int qmez_id; /* Object ID for Q_MEZ in Object Knowledgebase */
    int tgrav_id; /* Object ID for T_GRAV in Object Knowledgebase */
    float *Tgrav;
    float *Qmez;
} RSCI_Local_t;

static RSCI_Local_t *local;

This compiled and appeared to work correctly, but after much investigation, I eventually determined that it was the cause of the “random” lockups. Since I failed to allocate memory for a variable of the structure type before declaring a pointer to the structure type, I was getting a random pointer to somewhere in memory. When I wrote data to a member of the structure, I was writing to that random memory address. If I wrote to another module’s memory space or to the operating system’s space, then the system would crash. I solved this problem by allocating a variable of the structure type, declaring a pointer to the structure type, and then setting the pointer to point to the variable, as shown in the code fragment below.

/* create the local 'object' Data structure */
static RSCI_Local_t *local;
static RSCI_Local_t local_var;

... ...

void Module_Name(void)
{
    local = &local_var;
    ... more code ...
}

Once this change was applied to all of the modules I had written, the “random” lockup problem stopped occurring.

Another problem which took a great deal of time to solve involved the Chimera .rmod files. Since each of the Chimera/UTAP Interface Layer modules was implemented as a Chimera module, each had a .rmod file. The first line of the .rmod file is the
MODULE line. The name of the Chimera module, which should be the same as the filename (without the .c extension), is placed on this line. When the Chimera command "spawn module_name" is used to spawn a module, Chimera first looks at the file "module_name.rmod" to get the frequency and the name of the actual module to spawn. Since I was using an existing module as a basis for my new module, I copied the source code and the .rmod file. After changing the source code, however, I neglected to change the old module name to the new module name in the .rmod file. The result was that when I tried to spawn my module, Chimera actually spawned the old module. This caused a lot of problems because I thought my new module was working correctly for over a week when it actually did not work at all. I was able to catch this error because I added a print statement to my module and I noticed that it never printed out.

Another general problem was that no debugger was available to help determine what was causing all of the errors discussed above. Instead, kprintf statements were used to determine the state of the system at various points. Kprintf is a Chimera function which provides the same functionality as the C printf statement. Kprintf, however, does not cause the function invoking it to block and therefore lose control of the processor as does the printf statement.

III.4.2. Object Knowledgebase

When I designed my new system, I planned to make the Object Knowledgebase, or OK, a separate module. However, once I actually started implementing the module, I ran into a problem with Chimera that forced me to change the design. Because of the way Chimera handles the Global State Table, I could not find a good way of having a separate module write directly to the Global State Variable Table.
I spoke with Dr David Stewart, one of the creators of Chimera, to try to find a way to do so [8]. He suggested that I should not use the “Direct Write” function that is provided by Chimera because it was primarily included only for testing purposes. Without this function, no module is able to write directly to the state table. The \textit{Object Knowledgebase} module needs to be a static artifact as is the GSVT. However, if implemented as a Chimera module, it would be run cyclically, and so the data would only be copied to and from the GSVT at that same frequency. I was concerned that other modules might get incorrect data because of this situation. Therefore, I instead decided to implement the OK in a distributed fashion.

Instead of being one separate module, I have implemented the OK in each of the Chimera/UTAP Interface Layer, or CUIL, modules. Each CUIL module has functions to handle calls to the \textit{Object Knowledgebase}. I then slightly modified the standard UTAP messages used to send and get data from the Object Knowledgebase to take the new implementation into account. Instead of using the standard UTAP message \texttt{US_OK_ATTRIBUTE_QUERY}, I have changed the form to \texttt{US_OK_XXX_ATTRIBUTE_QUERY}, where XXX is the UTAP abbreviation for that module. For instance, the \textit{Operator Input} module would use the message \texttt{US_OK_OI_ATTRIBUTE_QUERY}. Although I have in effect added new messages to the UTAP specification, this seemed to be the best solution.

\textbf{III.4.3. Operator Input}

After solving all of the problems discussed above, this module was fairly easy to code. The CUIL module handles reading from the trackball through the serial port. The
gains and speed are read from the configuration .rmod file using Chimera system calls; these values are requested by the UTAP OI module via a UTAP message.

III.4.4. Robot Servo Control

This module was written in two parts. First, I made the Chimera puma_pidg module UTAP-compliant. Next, I implemented the Chimera grav_comp module. After both of these modules were tested independently, I combined them into the UTAP RSC module.

Whether gravity compensation is used or not is specified in the .rmod configuration file for the CUIL module. Also, the UTAP messages US_AXIS_SERVO_START_GRAVITY_COMPENSATION and US_AXIS_SERVO_STOP_GRAVITY_COMPENSATION can be issued to start or stop gravity compensation at any time while the system is running. Note that these messages can be issued by another module or the user can change the value directly in the configuration file.

III.5. Testing the Changes

I tested each UTAP module in isolation. By this I mean that only one UTAP module at a time was in use in the system. All other modules were the existing Chimera modules that had already been tested.

III.5.1. Simulation

I tested the Object Knowledgebase and the Operator Interface modules using simulation. I did this to ensure that they were working properly so that the robot would not be damaged in the case of an accident. The existing Chimera module psim_pidg was
used for the simulation. This module reads the commanded position and returns the updated measured position.

### III.5.2. Informal Testing

Once a module was completed and compiled correctly, it was tested informally. This informal testing involved using the trackball to move the robot or the simulated robot. I used `display_test.c`, another program that I wrote, to view various values from the GSVT for this testing. This testing was informal because there was no structure to the tests; they were simply used to gain confidence that the modules were actually working correctly before formally testing them.

### III.5.3. Formal Testing

I formally tested each module in the following manner:

1) I chose ten predefined positions and orientations.

2) The UTAP module under test was the only UTAP module spawned; all other modules were the existing chimera modules.

3) From the home position, the robot was commanded to each of the ten positions and orientations using the Chimera `trijgen` module, which generates a trajectory. Table III.4 lists some of the important predefined positions.

4) I compared the final position to the commanded position.

<table>
<thead>
<tr>
<th>Position Name</th>
<th>Joint 1</th>
<th>Joint 2</th>
<th>Joint 3</th>
<th>Joint 4</th>
<th>Joint 5</th>
<th>Joint 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>0.0</td>
<td>-1.5708</td>
<td>1.5708</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Data Initial</td>
<td>0.0</td>
<td>-2.36</td>
<td>2.36</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Data Final</td>
<td>1.5708</td>
<td>-1.5708</td>
<td>0.524</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Offboard</td>
<td>0.0</td>
<td>-1.5090</td>
<td>2.800</td>
<td>0.0</td>
<td>0.058</td>
<td>0.0</td>
</tr>
<tr>
<td>NBoard</td>
<td>0.0</td>
<td>-1.8090</td>
<td>0.3490</td>
<td>0.0</td>
<td>-0.078</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table III.4. Predefined Positions
This testing method ensured that each of the modules worked correctly when used alone. After I verified that each module was operating correctly, I spawned all of the available UTAP modules and repeated the testing procedure. This was done to ensure that the UTAP modules worked together without any system-level conflicts.

III.6. Measuring Performance

III.6.1. Missed Cycles

Once all of the UTAP modules were tested, performance data needed to be collected. Originally, I intended to use the number of missed cycles as a metric; however, UTAP did not impact the system performance as much as I had anticipated, so no missed cycles occurred in the baseline configuration using the existing gains. Chimera keeps track of missed cycles, so this data would have been easy to obtain and obtaining it would not have impacted the system performance at all.

III.6.2. Step Response

Since I could not use the missed cycle metric to quantify performance, I had to come up with a different measurement to determine how UTAP impacts performance. I decided to use the step response of the robot as a measure of the system performance because it shows the overall system performance. Step response shows how the robot responds over time to a commanded position. Instead of focusing only on the software performance, I was able to see the entire system performance by using the step response. Using an existing module called track, I collected data using the following procedure:

1) I spawned the Chimera modules track and trijgen and the UTAP module RSC.

   Note that RSC includes the grav_comp, puma_pidg, and Object
Knowledgebase functionality. Gravity compensation was turned on and the gains were set as shown in Table III.5.

2) I commanded the robot to the Data Initial position (See Table III.4).

3) I turned on the track module. This module starts to record data when the commanded and actual positions differ, so it did not immediately start recording data.

4) I commanded the robot to the Data Final position with a speed of 1.5 seconds. See Table III.4. The track module recorded data until the actual positions matched the commanded positions. Note that the end points were chosen so that the trajectory between them would excite the dynamics of the robot. Although joints 4, 5, and 6 are not commanded to change, these joints are affected by the dynamic forces and so the controller must compensate for these forces.

5) Steps 2, 3, and 4 were repeated two more times. The first time, the trajectory duration was 3 seconds; the second time it was 5 seconds. Thus, the trajectory was tested at slow, nominal, and fast speeds.

<table>
<thead>
<tr>
<th>Joint</th>
<th>Position Gain, $K_p$</th>
<th>Velocity Gain, $K_v$</th>
<th>Integral Gain, $K_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4000</td>
<td>80</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>11000</td>
<td>114</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3000</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>310</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>300</td>
<td>17</td>
<td>5</td>
</tr>
</tbody>
</table>

Table III.5. Gains Used for Testing and Data Acquisition
III.7. Summary

This chapter has described the UTAP specification, the pre-UTAP AFIT architecture, and the process by which the AFIT robotic system was made UTAP-compliant. In discussing the AFIT architecture, the Chimera Real-Time Operating System was explained. This chapter also presented the procedures by which the UTAP-compliant system was tested and how performance was measured. In the next chapter, I analyze the data collected using these procedures.
IV. Analysis

IV.1. Commanded versus Actual Position Error

From the data collected using the procedure described in Chapter III, I calculate and plotted the error between commanded and actual position of each joint for each trajectory. Each joint has three plots since each plot shows the error for the existing AFIT controller and for the UTAP-compliant controller for one of the three trajectories. The plots for the nominal trajectory are shown in Figures IV.1 through IV.6, while all of the plots can be found in Appendix D. As can be seen from the figures, each plot shows the error curves for both controllers. The error in radians is plotted against the cycle of the periodic task which recorded the data (Note: $1^\circ = 0.0175$ radians). The cycle can be equated to time because the task recorded data once each cycle and it was executed at a frequency of 50 Hz. The plot shown in Figure IV.1 shows that both controllers had error but converged to the steady-state condition of no error. Figures IV.2 through IV.6 show similar results.

Each of the plots shows a common trend. For the slow trajectory, the UTAP controller has a spiky curve for about the first 250 cycles; this type of curve also occurs for about the first 100 cycles during the nominal trajectory and for the first 50 cycles for the fast trajectory. This type of curve can be explained by noting that the gains used while recording this data have been tuned for the old AFIT controller and not for the new UTAP controller. Gain tuning is a process by which the controller gains are adjusted to optimum values for a particular system.
Figure IV.1. Error Plot for Joint 1 at Nominal Trajectory

Figure IV.2. Error Plot for Joint 2 at Nominal Trajectory
Joint 3 Error for Nominal Trajectory

Figure IV.3. Error Plot for Joint 3 at Nominal Trajectory

Joint 4 Error for Nominal Trajectory

Figure IV.4. Error Plot for Joint 4 at Nominal Trajectory
Joint 5 Error for Nominal Trajectory

![Joint 5 Error Plot](image1.png)

Figure IV.5. Error Plot for Joint 5 at Nominal Trajectory

Joint 6 Error for Nominal Trajectory

![Joint 6 Error Plot](image2.png)

Figure IV.6. Error Plot for Joint 6 at Nominal Trajectory
IV.2. Analysis

The plots in Appendix D show the error as a function of time, while Tables IV.1, IV.2, and IV.3 show the integral error which occurred during the 500 cycles of data collection. I calculated the integral error by summing the absolute value of the error measurement at each cycle. As can be seen in the tables, the new UTAP compliant controller had slightly less total error than the AFIT controller for joints 1 and 3 at each trajectory, but the UTAP controller had more error for the other joints at all trajectories. Also, in every case for joints 4, 5 and 6, the UTAP controller had much greater error than did the AFIT controller.

I calculated the percent difference of error using the equation

\[
\text{Percent Difference} = 100 \times \left( \frac{\text{UTAP Controller Error} - \text{AFIT Controller Error}}{\text{UTAP Controller Error}} \right)
\]

As Tables IV.1, IV.2, and IV.3 show, the UTAP controller had a much greater total error than did the AFIT controller. However, the joint 6 error accounts for much of this total error. If the joint 6 error is not included, the total percent difference of error drops to 14.07%, 17.41%, and 28.58%, respectively. If the error for joints 4, 5 and 6 are excluded, the UTAP controller actually has less total error than the AFIT controller. The percent differences of error for this case are -1.80%, -2.45%, and -0.97%, respectively.

<table>
<thead>
<tr>
<th></th>
<th>Joint 1</th>
<th>Joint 2</th>
<th>Joint 3</th>
<th>Joint 4</th>
<th>Joint 5</th>
<th>Joint 6</th>
<th>Total Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFIT Controller</td>
<td>1.8189</td>
<td>0.7853</td>
<td>1.0213</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>3.6255</td>
</tr>
<tr>
<td>UTAP Controller</td>
<td>1.7497</td>
<td>0.9547</td>
<td>0.8569</td>
<td>0.5317</td>
<td>0.1558</td>
<td>95.2280</td>
<td>99.4768</td>
</tr>
<tr>
<td>Percent Difference</td>
<td>-3.96</td>
<td>17.74</td>
<td>-19.16</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>96.36</td>
</tr>
</tbody>
</table>

Table IV.1. Integral Error for Slow Trajectory
### Table IV.2. Integral Error for Nominal Trajectory

<table>
<thead>
<tr>
<th>Joint 1</th>
<th>Joint 2</th>
<th>Joint 3</th>
<th>Joint 4</th>
<th>Joint 5</th>
<th>Joint 6</th>
<th>Total Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFIT Controller</td>
<td>2.3471</td>
<td>0.8405</td>
<td>1.2949</td>
<td>0.0115</td>
<td>0.0001</td>
<td>0.0377</td>
</tr>
<tr>
<td>UTAP Controller</td>
<td>2.2042</td>
<td>1.1034</td>
<td>1.0679</td>
<td>0.8692</td>
<td>0.2423</td>
<td>104.7391</td>
</tr>
<tr>
<td>Percent Difference</td>
<td>-6.48</td>
<td>23.83</td>
<td>-21.22</td>
<td>98.68</td>
<td>99.94</td>
<td>99.96</td>
</tr>
</tbody>
</table>

### Table IV.3. Integral Error for Fast Trajectory

<table>
<thead>
<tr>
<th>Joint 1</th>
<th>Joint 2</th>
<th>Joint 3</th>
<th>Joint 4</th>
<th>Joint 5</th>
<th>Joint 6</th>
<th>Total Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFIT Controller</td>
<td>2.9696</td>
<td>1.1653</td>
<td>1.6241</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>UTAP Controller</td>
<td>2.8886</td>
<td>1.3002</td>
<td>1.5148</td>
<td>1.6659</td>
<td>0.4419</td>
<td>1.3951</td>
</tr>
<tr>
<td>Percent Difference</td>
<td>-2.80</td>
<td>10.38</td>
<td>-7.22</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

### IV.3. Summary

The performance of the UTAP-compliant and the pre-UTAP architectures was presented and compared using the step response and the integral error of the step response. The collected data shows that the UTAP-compliant system had worse overall performance than the non UTAP-compliant system, but I believe this performance degradation can be improved with gain tuning.
V. Conclusions and Recommendations

V.1. Conclusions

The work performed during this research project leads to several conclusions about the UTAP architecture. The primary conclusion is that the UTAP architecture can be implemented and it will work using the Chimera operating system and the PUMA 560 manipulator. Since the architecture is designed to be system-independent, this conclusion can be generalized. Assuming that an interface layer similar to the Chimera/UTAP Interface Layer can be implemented on an arbitrary system, then the UTAP software architecture can also be implemented on that system.

Another conclusion is that the system performance is adversely affected by the UTAP architecture. Both the message-passing and the interface layer contribute to the degradation by adding overhead to the processor. However, the data collected is insufficient to determine how much of the degradation is due to message-passing and how much is due to the interface layer. Also, even though the error is increased, the commanded task was still performed in the time required. Thus, for a soft real-time system such as the AFIT robot, the difference in error may not matter as long as the task or application is still accomplished by a UTAP-compliant system in the time required.

UTAP is a good effort to try to standardize telerobotic system development in an open, portable manner using software engineering principles. Though the UTAP specification has some inconsistencies and it does not address all issues, it is a vehicle to achieve standardization in the robotics community much as the IBM PC Bus eventually
caused a standardization among PC components. Such standardization helps achieve the goals of lowering cost, improving reliability, and increased component availability.

In general, the implementation of a UTAP-compliant system is not trivial due to the generality of the UTAP specification. Because the specification is not yet finalized, the implementor must make many decisions which may make the code incompatible with another implementation. One example of this type of decision is the format of the configuration file.

V.2. Recommendations

V.2.1. Recommendations for Future Research

A follow-on study should analyze the code produced in this thesis project. The delay associated with the code should be analyzed analytically using a technique such as rate monotonic analysis. Also, more data should be collected so that the performance impacts can be identified more clearly.

In order to get more data on the difficulty of implementing UTAP and on how it affects performance, a project similar to this thesis effort should be conducted using a different operating system but the same hardware and the C language. Since AFIT will be receiving a copy of VxWorks from the vendor and it supports our current hardware, I recommend implementing UTAP using the VxWorks operating system. The main issues to be dealt with in this project would be: 1) the necessity and design of an interface layer for this OS; 2) what telerobotic task to implement; and 3) how to measure performance. The third issue is important because there is a need to separate the message-passing performance impact from the interface layer impact. An analytical technique such as rate
monotonic analysis may be useful in this regard. Also, there will not be any existing working code to use as a benchmark to test the UTAP modules against.

To get even more data, a similar experiment should be performed using a different robotic system. AFIT already has the Adept robot, which uses a proprietary OS and language. The same issues mentioned above would apply to this project. A project of this type would show the language-, operating system-, and robotic system-independence of the UTAP specification.

Implementing a UTAP compliant system using a programming language other than C would prove beneficial because it would show that the UTAP specification is truly language-independent. I would recommend using Ada if possible. Again, all of the above comments apply. In addition, an operating system which supports Ada would have to be used.

Any of the projects discussed above should also strive to implement more of the features of the UTAP specification that were not implemented in this thesis project. Implementing a system-level configuration file to describe what components are present and to show the relationships among modules would allow the system to be more powerful and more easily reconfigurable.

Another feature that should be implemented is to support posting of messages that each module will respond to. This feature will allow the UTAP implementation to be more stable when confronted with a UTAP module which passes a non-implemented message.

Finally, any new UTAP implementation should implement more messages so that a clear picture of the performance impact of the message-passing may not be seen. In other
words, the more message-passing that occurs, the better we can observe the impact on the system from these messages.

V.2.2. Recommendations to Improve the UTAP Specification

The specification should clearly define what UTAP compliance means and come up with a meaningful way of measuring it. Without such a definition, it will be hard to compare different implementations of UTAP and to determine what a claim of “UTAP-compliant” means.

The module responsibilities need to be defined more clearly and more precisely. If the responsibilities are better defined, there will be less variance in how modules implemented by different people perform the assigned tasks.

Message meanings and parameters also need to be more clearly defined. For many messages, the implementor has to decide the function of the message simply by looking at the message name and the parameter list. Similarly, some of the parameters have unclear names and functions. Also, listing the intended purpose or function of each message may prevent an implementor from adding a new message to perform the same task as an old message that the implementor did not understand.

The configuration file formats and the how-to-pass part of the interface need to be defined explicitly. If these features are left undefined, then different implementors may choose different formats and then UTAP modules will no longer be transportable from one system to another.

Finally, the UTAP document [6] is “C-centric”, but the specification is intended to be language-independent. Thus, I recommend modifying the examples and the messages
into a different format. This will remove the suggestion that C should be used to implement UTAP.

V.3. Summary

This thesis project has investigated the UTAP software architecture by implementing a UTAP-compliant system and measuring the performance. The results of this thesis will assist the Air Force in determining whether the UTAP specification should be adopted as an Air Force standard. This thesis effort has shown that the UTAP architecture is valid but that it lacks maturity and its implementation on an existing system is not trivial. This thesis also discusses issues that an implementor may face when a system is changed to be compliant with the UTAP specification. Figure V.1 summarizes the conclusions of this thesis. Figure V.2 summarizes my recommendations for future research, and Figure V.3 summarizes my recommendations for the UTAP specification.

<table>
<thead>
<tr>
<th>Conclusion</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UTAP can be implemented on an arbitrary system (with the use of an interface layer)</td>
</tr>
<tr>
<td>2</td>
<td>System performance is adversely affected by the UTAP architecture</td>
</tr>
<tr>
<td>3</td>
<td>UTAP is a good effort to try to standardize telerobotic system development in an open, portable manner</td>
</tr>
<tr>
<td>4</td>
<td>Implementation of a UTAP-compliant system is not trivial due to the generality of the UTAP specification</td>
</tr>
</tbody>
</table>

Figure V.1. Summary of Conclusions
### Research Recommendation

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Further analyze code produced in this thesis project</td>
</tr>
<tr>
<td>2</td>
<td>Implement UTAP using a different operating system</td>
</tr>
<tr>
<td>3</td>
<td>Implement UTAP using a different robotic system</td>
</tr>
<tr>
<td>4</td>
<td>Implement UTAP using a different programming language</td>
</tr>
<tr>
<td>5</td>
<td>Implement more of the features from the UTAP specification</td>
</tr>
<tr>
<td>6</td>
<td>Implement more of the messages defined by the UTAP specification</td>
</tr>
</tbody>
</table>

*Figure V.2. Summary of Research Recommendations*

### Recommendation for UTAP Document

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Clearly define UTAP-compliance and how to measure it</td>
</tr>
<tr>
<td>2</td>
<td>Define module responsibilities more clearly</td>
</tr>
<tr>
<td>3</td>
<td>Define message and message parameter meanings more clearly</td>
</tr>
<tr>
<td>4</td>
<td>Define the configuration file formats</td>
</tr>
<tr>
<td>5</td>
<td>Define how data will be passed between modules (how-to-pass)</td>
</tr>
<tr>
<td>6</td>
<td>Remove perception that C language is required by reformatting examples and messages</td>
</tr>
</tbody>
</table>

*Figure V.3. Summary of Recommendations for UTAP Specification*
Bibliography


APPENDIX A

Chimera/UTAP System User’s Manual

Note: User input and system prompts are shown in a different font than the rest of the text. User input is also shown in boldface. Text in italics represents a placeholder for some other text (i.e., a variable name).

Steps:

1. Log in to Gepetto at the system prompt. Gepetto is the only workstation that can be used to run the PUMA robot.

2. Change to the ~/chimera/bin directory by using the following command at the UNIX prompt:

   ```
   gepetto:@~/chimera/bin> cd chimera/bin
   ```

3. Ensure that the VMEbus is powered on.

4. Start the Chimera system by typing the following command at the UNIX prompt:

   ```
   gepetto:@~/chimera/bin> chim
   ```

4. Calibrate the PUMA robot by typing the following at the Chimera prompt:

   ```
   CHIM:control> ex calibrate
   ```

   This command will execute the calibrate command that is stored in the ~/chimera/bin subdirectory.

5. Start the main system, tmain. This is accomplished by running a script which contains the following lines:

   ```
   attach control
   attach crusher
   kill all
   select control
   download tmain
   ```
The script is executed by typing the following at the Chimera prompt:

CHIM:control> <Go

6. Once the script has completed, the tmain program is downloaded on the real-time processing units (RTPUs). To start the program, type the following command at the Chimera prompt:

CHIM:control> go all

This command will start up the SBS system and will result in the SBS prompt.

7. Spawn any necessary modules and then turn them on. The script UTAP_Example will spawn the following modules: int_RSC, int_jtrackball, and trjgen. The int_PUMA_pidg module will be turned on. The script contains the following commands:

spawn crusher int_RSC
spawn control int_jtrackball
spawn control trjgen
on int_RSC

The script is executed by typing the following at the SBS prompt:

SBS tmain<none> <UTAP_example

8. Either of the trjgen or int_jtrackball modules can be turned on to cause the robot to change position. DO NOT turn both modules on at the same time as they both attempt to write values that will cause motion. Unpredictable motion may result if this occurs, and the robot may be damaged because of this.

To turn on the trajectory generator module trjgen, type the following at the SBS prompt:

SBS tmain<some-module> on trjgen j1_val j2_val j3_val j4_val j5_val j6_val duration speed

or use a script. Similarly, to turn on the trackball module int_jtrackball, type:

SBS tmain<some-module> on int_jtrackball
Some scripts exist which will use the trjjgen module. A few of these scripts are shown below:

- `<Offboard - on trjjgen 0.0 -1.5090 2.8000 0.0 0.058 0.0 5.0 3`
- `<Board - on trjjgen 0.0 -1.3090 2.8000 0.0 0.058 0.0 5.0 3`
- `<NBoard - on trjjgen 0.0 -1.8090 0.3490 0.0 -0.078 0.0 7.0 3`
- `<Home - on trjjgen 0.0 -1.5708 1.5708 0.0 0.0 0.0 5.0 3`

9. The trjjgen module will automatically turn off when the error between commanded and actual position is within a certain value. The int_jtrackball and int_PUMA_pidg modules, as well as any other modules which may have been used during the Chimera session, must be turned off manually by using the following command at the SBS prompt:

```
SBS tmain<some-module> off module-name
```

10. When the session is finished, each module must be turned off as in step 9, then each module must be killed by typing the following command at the SBS prompt:

```
SBS tmain<some-module> kill module-name
```

11. Once all modules are killed, exit the SBS system by typing the following at the SBS prompt:

```
SBS tmain<none> quit
```

12. Exit Chimera by typing the following at the Chimera prompt:

```
CHIM:control> quit
```

13. Logout of the system by typing the following at the Unix prompt:

```
gepetto:@-/chimera/bin> logout
```

Miscellaneous Chimera Commands:
To see what tasks exist and what state they are in, as well as information on missed cycles, execution frequency, etc., type the following command at the SBS prompt:

```
SBS tmain<some-module> status
```

To view the Global State Table, type the following command at the SBS prompt:

```
SBS tmain<some-module> display all
```
For a list of available modules, type the following command at the SBS prompt:

```
SBS tmain<some-module> mod
```

For a list of commands available in Chimera, type the following command at the SBS prompt:

```
SBS tmain<some-module> ?
```

For help with a particular command, type the following command at the SBS prompt:

```
SBS tmain<some-module> help command-name
```
APPENDIX B

Chimera/UTAP System Programmer's Manual

Note: User input and system prompts are shown in a different font than the rest of the text. User input is also shown in boldface. Text in italics represents a placeholder for some other text (i.e., a variable name).

Steps:

1. Log in to Gepetto or any other workstation on the Hawkeye network.

2. If any Chimera modules need to be edited, then change to the ~/chimera/src/module directory by using the following command at the UNIX prompt:

   gepetto:@~/chimera/bin> cd ~/chimera/src/module

   Edit any Chimera modules that need to be changed by using any text editor. The Makefile in this directory needs to be updated to include any new Chimera modules that are to be included. The Makefile will be used to compile the final system, so only modules that are referenced in the Makefile will be compiled.

3. If any changes to the .rmod file, which is the Chimera configuration file, need to be made, then change to the ~/chimera/module directory by using the following command at the UNIX prompt:

   gepetto:@~/chimera/bin> cd ~/chimera/module

   Edit the .rmod files that need to be changed by using any text editor. Make sure that the "MODULE" line in the .rmod file references the correct module name or else problems may occur.

4. If any UTAP modules need to be edited, then change to the ~/chimera/src/libutap directory by using the following command at the UNIX prompt:

   gepetto:@~/chimera/bin> cd ~/chimera/src/libutap

   Edit any UTAP modules that need to be changed by using any text editor. The Makefile in this directory needs to be updated to include any new UTAP modules that are to be included. The Makefile will be used to compile the final system, so only modules that are referenced in the Makefile will be compiled.
5. The final Makefile which links in all necessary components is located in the
~/chimera/src/main directory. Change to this directory by using the following command at
the UNIX prompt:

gepetto: @~/chimera/bin> cd ~/chimera/src/main

Ensure that all Chimera and UTAP modules that are needed are referenced in this
Makefile. If a module is missing, then Chimera will generate an error message when the
missing module is spawned.

6. If no new modules are being added to the system (i.e., only changes to existing
modules are being made), then none of the Makefiles will need to be updated.

7. Change to the ~/chimera/src directory and then make the system by using the
following commands at the UNIX prompt:

gepetto: @~/chimera/bin> cd ~/chimera/src
gepetto: @~/chimera/bin> make

Any compilation or linking errors must be corrected. If there are no errors, then the gicc
compiler will be used to compile and link the system. The tmain program is the final
product of this step.
APPENDIX C

UTAP Messages

Note: Bold and Italicized messages have been implemented in this Thesis Project.

**GENERIC (42)**
*US* STARTUP
*US* SHUTDOWN
*US* RESET
*US* ENABLE
*US* DISABLE
*US* ESTOP
*US* START
*US* STOP
*US* ABORT
*US* HALT
*US* INIT
*US* HOLD
*US* PAUSE
*US* RESUME
*US* ZERO
*US* BEGIN SINGLE_STEP
*US* NEXT SINGLE_STEP
*US* CLEAR SINGLE_STEP
*US* BEGIN BLOCK
*US* END BLOCK
*US* BEGIN PLAN
*US* END PLAN
*US* USE PLAN
*US* BEGIN MACRO
*US* END MACRO
*US* USE MACRO
*US* BEGIN EVENT
*US* END EVENT
*US* MARK BREAKPOINT
*US* MARK EVENT
*US* GET_SELECTION_ID
*US* POST_SELECTION_ID
*US* USE_SELECTION
*US* USE AXIS MASK
*US* USE_EXT_ALGORITHM
*US* LOAD_EXT_PARAMETER
*US* GET_EXT_DATA_VALUE
*US* POST_EXT_DATA_VALUE
*US* SET_EXT_DATA_VALUE
*US* LOAD_STATUS_TYPE
*US* LOAD_STATUS_PERIOD
*US* GENERIC_STATUS_REPORT
*US* AXIS_SERVO_USE_ABS_VELOCITY_MODE
*US* AXIS_SERVO_USE_REL_VELOCITY_MODE
*US* AXIS_SERVO_USE_PID
*US* AXIS_SERVO_USE_FEEDFORWARD_TORQUE
*US* AXIS_SERVO_USE_CURRENT
*US* AXIS_SERVO_USE_VOLTAGE
*US* AXIS_SERVO_USE_STIFFNESS
*US* AXIS_SERVO_USE_COMPLIANCE
*US* AXIS_SERVO_USE_IMPEDANCE
*US* AXIS_SERVO_START_GRAVITY_COMPENSATION
*US* AXIS_SERVO_STOP_GRAVITY_COMPENSATION
*US* AXIS_SERVO_LOAD_DOF
*S* AXIS_SERVO_LOAD_CYCLE_TIME
*S* AXIS_SERVO_LOAD_PID_GAIN
*S* AXIS_SERVO_LOAD_JOINT_LIMIT
*S* AXIS_SERVO_LOAD_VELOCITY_LIMIT
*S* AXIS_SERVO_LOAD_GAIN_LIMIT
*US* AXIS_SERVO_HOME
*US* AXIS_SERVO_SET_BREAKS
*US* AXIS_SERVO_CLEAR_BREAKS
*US* AXIS_SERVO_SET_TORQUE
*US* AXIS_SERVO_SET_CURRENT
*US* AXIS_SERVO_SET_VOLTAGE
*US* AXIS_SERVO_SET_POSITION
*US* AXIS_SERVO_SET_VELOCITY
*US* AXIS_SERVO_SET_ACCELERATION
*US* AXIS_SERVO_SET_FORCES
*US* AXIS_SERVO_JOG
*US* AXIS_SERVO_JOG_STOP

**TOOL (14)**
*US* SPINDLE_RETRACT_TRAVERSE
*US* SPINDLE_LOAD_SPEED
*US* SPINDLE_START_TURNING
*US* SPINDLE_STOP_TURNING
*US* SPINDLE_RETRACT
*US* SPINDLE_ORIENTATION
*US* SPINDLE_LOCK_Z
*US* SPINDLE_USE_FORCE
*US* SPINDLE_USE_NO_FORCE
*US* FLOW_START_MIST
*US* FLOW_STOP_MIST
*US* FLOW_START_FLOOD
*US* FLOW_STOP_FLOOD
*US* FLOW_LOAD_PARAMETERS

**SENSOR (45)**
*US* START_TRANSFORM
*US* STOP_TRANSFORM
*US* START_FILTER
*US* STOP_FILTER
*US* SENSOR_USE_MEASUREMENT_UNITS
*US* SENSOR_LOAD_SAMPLING_SPEED
*US* SENSOR_LOAD_FREQUENCY
*US* SENSOR_LOAD_TRANSFORM
*US* SENSOR_LOAD_FILTER
*US* SENSOR_GET_READING

**ERRORS (9)**
*US* ERROR_COMMAND_NOT_IMPLEMENTED
*US* ERROR_COMMAND_ENTRY
*US* ERROR_DUPLICATE_NAME
*US* ERROR_BAD_DATA
*US* ERROR_NO_DATA_AVAILABLE
*US* ERROR_SAFETY_VIOLATION
*US* ERROR_LIMIT_EXCEEDED
*US* ERROR_OVER_SPECIFIED
*US* ERROR_UNDER_SPECIFIED

**AXIS SERVO (34)**
*US* AXIS_SERVO_USE_ANGLE_UNITS
*US* AXIS_SERVO_USE_RADIAN_UNITS
*US* AXIS_SERVO_USE_ABS_POSITION_MODE
*US* AXIS_SERVO_USE_REL_POSITION_MODE
US_TDS_SELECT_OPERATION
US_TDS_SELECT_OPMODE
US_TDS_LOAD_SELECTIONS
US_TDS_LOAD_REFERENCE_UNITS
US_TDS_LOAD_RATE_DEFAULTS
US_TDS_LOAD_ORIGIN
US_TDS_LOAD_SENSING_DEFAULTS

TASK_KNOWLEDGE (4)
US_TK_DEFINE_FRAMEWORK
US_TK_MACRO_CREATE
US_TK_MACRO_DELETE
US_TK_MACRO_MODIFY

PARENT_TASK_PROGRAM_SEQUENCING (7)
US_TPS_SELECT_AGENT
US_TPS_SELECT_TOOL
US_TPS_SELECT_SENSOR
US_TPS_INTERP_RUN_PLAN
US_TPS_INTERP_HALT_PLAN
US_TPS_INPUT_REQUEST
US_TPS_OUTPUT_ENABLE_SUBSYSTEM

TASK_PROGRAM_SEQUENCING (10)
US_TPS_FREE_SPACE_MOTION
US_TPS_GUARDED_MOTION
US_TPS_CONTACT_MOTION
US_TPS_SET_SUPERVISORY_MODE
US_TPS_SELECT_FEATURE
US_TPS_SELECT_MATERIAL
US_LOAD_OBSTACLE
US_LOAD_PATTERN
US_TPS_MARK_EVENT
US_TPS_ENABLE

OPERATOR_INTERFACE (9)
US_BEGIN_FRAMEWORK
US_END_FRAMEWORK
US_CREATE_FRAMEWORK
US_DELETE_FRAMEWORK
US_ADD_SYMBOLIC_ITEM
US_DELETE_SYMBOLIC_ITEM
US_ADD_SYMBOLIC_ITEM_ATTR
US_DELETE_SYMBOLIC_ITEM_ATTR
US_SET_SYMBOLIC_ITEM_ATTR

OBJECT_MODELING (3)
US_OM_CREATE
US_OM_DELETE
US_OM_MODIFY

OBJECT_CALIBRATION (4)
US_OC_SET_CALIB
US_OC_GET_CALIB
US_OC_SET_ATTR
US_OC_GET_ATTR

OBJECT_KNOWLEDGE (9)
US_OK_RECORD
US_OK_PLAYBACK
US_OK_CREATE_OBJ
US_OK_DELETE_OBJ
US_OK_MODIFY
*US_OK_MODIFY_ATTRIBUTE
*US_OK_ATTRIBUTE_QUERY
*US_OK_OUTPUT_REGISTERED_OBJ_ID

US_OK_ATTRIBUTE_RESPONSE

TRAJECTORY_DESCRIPTION (15)
US_TRD_OPEN
US_TRD_ERASE
US_TRD_RECORD
US_TRD_RECORD_ON
US_TRD_RECORD_OFF
US_TRD_FIND
US_TRD_NEXT
US_TRD_PREVIOUS
US_TRD_DELETE
US_TRD_NAME_ITEM
US_TRD_DELETE_ITEM
US_TRD_SET_JOINT_MODE
US_TRD_SET_CARTESIAN_MODE
US_TRD_MODIFY
US_TRD_ADD_ELEMENT

ANALYSIS_DIAGNOSIS_SYSTEM (1)
USADS_COLLISION_DETECTED

UTAP_DATA_DEFS (34)
US_POST_ID
US_GET_OBJECT_ID
US_USE_OBJECT
US_GET_FEATURE
US_USE_FEATURE
US_GET_VALUE
US_POST_VALUE
US_GET_LIST
US_POST_LIST
US_ATTRIBUTE_POST_RESPONSE
US_ATTRIBUTE_GET_TIME
*US_ATTRIBUTE_GET_POSITION
US_ATTRIBUTE_GET_ORIENTATION
US_ATTRIBUTE_GET_POSE
US_ATTRIBUTE_GET_VELOCITY
US_ATTRIBUTE_GET_ACCELERATION
US_ATTRIBUTE_GET_JERK
US_ATTRIBUTE_GET_FORCE
US_ATTRIBUTE_GET_TORQUE
US_ATTRIBUTE_GET_MASS
US_ATTRIBUTE_GET_TEMPERATURE
US_ATTRIBUTE_GET_PRESSURE
US_ATTRIBUTE_GET_VISCOSITY
US_ATTRIBUTE_GET_LUMINANCE
US_ATTRIBUTE_GET_HUMIDITY
US_ATTRIBUTE_GET_FLOW
US_ATTRIBUTE_GET_HARDNESS
US_ATTRIBUTE_GET_ROUGHNESS
US_ATTRIBUTE_GET_GEOMETRY
US_ATTRIBUTE_GET_TOPOLOGY
US_ATTRIBUTE_GET_SHAPE
US_ATTRIBUTE_GET_PATTERN
US_ATTRIBUTE_GET_MATERIAL
US_ATTRIBUTE_GET_KINEMATICS

MESSAGES ADDED FOR THIS PROJECT
(NOT IN UTAP SPECIFICATION)
*US_RSC_LOAD_TORQUES
*US_RSC_CHECK_ROBOT_POWER
*US_ERROR_ROBOT_POWER

58
APPENDIX D

Step Response Error Plots
Joint 1 Error for Nominal Trajectory

Figure D.1. Joint 1 Error for Nominal Trajectory
Figure D.2. Joint 2 Error for Nominal Trajectory
Figure D.3. Joint 3 Error for Nominal Trajectory
Figure D.4. Joint 4 Error for Nominal Trajectory
Figure D.5. Joint 5 Error for Nominal Trajectory
Figure D.6. Joint 6 Error for Nominal Trajectory

Joint 6 Error for Nominal Trajectory
Figure D.7. Joint 1 Error for Fast Trajectory
Figure D.8. Joint 2 Error for Fast Trajectory
Figure D.9. Joint 3 Error for Fast Trajectory
Figure D.10. Joint 4 Error for Fast Trajectory
Figure D.11. Joint 5 Error for Fast Trajectory
Joint 6 Error for Fast Trajectory

Figure D.12. Joint 6 Error for Fast Trajectory
Figure D.13. Joint 1 Error for Slow Trajectory
Figure D.14. Joint 2 Error for Slow Trajectory
Joint 3 Error for Slow Trajectory

Figure D.15. Joint 3 Error for Slow Trajectory
Figure D.16. Joint 4 Error for Slow Trajectory
Joint 5 Error for Slow Trajectory

Figure D.17. Joint 5 Error for Slow Trajectory
Figure D.18. Joint 6 Error for Slow Trajectory
APPENDIX E

UTAP and Interface Module Source Code
Display Test Module
(used to test other modules)

/* ***************************************************************/
/* display_test.c */
/* */
/* created by Kevin P Anchor 09-13-95 */
/* Air Force Institute of Technology */
/* */
/* modified: */
/* $ date $ $ initials $ $ comments $ */
/* */
/* reviewed by $ Some name here $ $ date $ */
/* */
/* ***************************************************************/

This module periodically displays the values of vars in the
state table to the screen.
It is used for testing purposes only.
State variable table:
        INCONST: none
        OUTFONST: none
        INVAB: Q_REF, T_GRAV
        OUTVAR: none

Special notes:

/* ***************************************************************/
/* include files */
/* ***************************************************************/
#include <chimera.h>
#include <sbs.h>

/* ***************************************************************/
/* module 'Local_t' definition as required by Chimera */
/* ***************************************************************/
typedef struct
{ float *Gref;
  float *Tgrav;
} display_testLocal_t;
Interface Module for jTrackball

/* ******************************************************* */
/* Module: int_jtrackball.c */
/* created by: Kevin Anchor 13 Aug 95 */
/* Air Force Institute of Technology */
/* This module is based on jtrackball.c written */
/* by Capt Tom Deeter */
/* modified: */
/* ******************************************************* */

/* This module acts as an interface between the Chimera RTOS and */
/* the Unified Telerobotic Architecture Project (UTAP) compliant */
/* module utap_jtrackball.c. */
/* */
/* INPUT: Q_REF - reference joint positions */
/* OUTPUT: Q_REF - reference joint positions */
/* Special notes: */
/* */
/* ******************************************************* */

#include <chimera.h>
#include <sbs.h>
#include <iod.h>

/* ******************************************************* */
/* macro definitions */
/* ******************************************************* */
#define NBBYTES 12  /* read 12 bytes of data from trackball */
#define UTAP_ERROR_FLAG */
#define UTAP_NO_ERROR 0
#define UTAP_ERROR_DISABLE_OI 1

/* ******************************************************* */
/* internal function declarations */
/* ******************************************************* */

static int Clean_port(IOD *iod);  /*Function to clean serial port and send handshake*/

/* ******************************************************* */
/* External function declarations */
/* ******************************************************* */
extern int US_INIT_OI(void);
extern int US_START_OI(void);

/* ******************************************************* */
/* global vars available to all functions in this module */
/* ******************************************************* */

/* Used to transfer values to/from UTAP module */
static float utap_Ref; */
/* the value of the IOD port in use by Chimera */
static IOD *iod_var;
static float gain[6];  /* values of the rmad GAINS */
static float speed;  /* selected speed of the movement */
/* set during a UTAP error condition, values are defined above as macros*/
static int UTAP_ERROR_FLAG;

/* ******************************************************* */
/* module 'Local_t' definition as required by Chimera */
/* ******************************************************* */

typedef struct {
  float *Qref;
  int *NoDoF;
  const char *iod_name;  /* Number of degrees of freedom */
  IOD *iod;  /* iod interface variable */
  int sioPort;  /* which port (from rmad file) */
  char *iodevname[MAXIODEVLEN];  /* which io device (from rmad file) */
  iodIOParam_t sloparam;  /* serial port parameters: page 169 */
  int _jtrackballLocal_t;
} int_jtrackballLocal_t;

/* ******************************************************* */
/* module initialization as required by Chimera */
/* ******************************************************* */
SBS_MODULE(int_jtrackball);

/* ******************************************************* */
/* Chimera Interface Modules */
/* ******************************************************* */

/* ******************************************************* */
/* init_jtrackballInit Initialize the module. */
/* ******************************************************* */

int int_jtrackballInit(configInfo_t *configInfo, local_t *local, sbsTask_t *stack);

82
sboTask_t *stack;

iod_var = local->iod;

/* Make sure the error flag is cleared */
UTAP_ERROR_FLAG = UTAP_NO_ERROR;

US_INIT_OK(); /* Call UTAP Init Module */
return (int) local;

} /* *****************************************************************

/* int jtrackballReinit Re-Initialize the module.
*/
/* ******************************************************************/

int _jtrackballReinit(local, stack)
int jtrackballLocal_t *local;
sboTask_t *stack;

Clean_port(local->iod);
return I_OK;

} /* ******************************************************************

/* int _jtrackball Re-start up the module.
*/
/* ******************************************************************/

int _jtrackball(local, stack)
int jtrackballLocal_t *local;
sboTask_t *stack;

kprintf("Starting the robot control in joint 1 mode at speed 2.11fps", speed);

return I_OK;

} /* ******************************************************************

/* int jtrackballCycle Process module information.*/
/* ******************************************************************/

int _jtrackballCycle(local, stack)
int jtrackballLocal_t *local;
sboTask_t *stack;

US_START_OK(); /* Call UTAP Module */

if (UTAP_ERROR_FLAG == UTAP_NO_ERROR)
return I_OK; /* normal exit */
else if (UTAP_ERROR_FLAG == UTAP_ERROR_DISABLE_USER)

/* the trackball sends the number three times so we must clean
*/
port twice so button is erased*/
Clean_port(iod_var);
printf("****UTAP: Operator Interface Module Disabled\n");
/* Clear error flag so it won't cause a problem when
the module is restarted */
UTAP_ERROR_FLAG = UTAP_NO_ERROR;
sbsMsgSend(stask, SBS_OFF); /* send finish signal */
return SBS_OFF;
} else
{
printf("Unknown error flag was raised in int_jtrackballCycle\n", UTAP_ERROR_FLAG);
printf("Disabling UTAP OI Module (just in case)...");
/* the trackball sends the number three times so we must clean
port twice so button is erased*/
Clean_port(iod_var);
Clean_port(iod_var);
/* Clear error flag so it won't cause a problem when
the module is restarted */
UTAP_ERROR_FLAG = UTAP_NO_ERROR;
sbsMsgSend(stask, SBS_OFF); /* send finish signal */
return SBS_OFF;
}

/* ***********************************************/
/* int_jtrackballOff Stop the module. */
/* ***********************************************/
int int_jtrackballOff(local, stask)
int jtrackballLocal_t * local;
sbsTask_t *stask;
{kprintf("int_jtrackball: OFF\n");
 return I_OK;
}

/* ***********************************************/
/* int_jtrackballKill Clean up after the module. */
/* ***********************************************/
int int_jtrackballKill(local, stask)
int jtrackballLocal_t *local;
sbsTask_t *stask;
{kprintf("int_jtrackball: FINISHED\n");
 iodClose(local->iod);
 return I_OK;
}

/* ***********************************************/
/* int_jtrackballError Attempt automatic error recovery. */
/* ***********************************************/
int int_jtrackballError(local, stask, mptr, ermsg, errcode)

int jtrackballLocal_t *local;
sbsTask_t *stask;

errModule_t *mptr;
char *errmsg;
int errcode;
{
/* Return after not correcting error. */
return SBS_ERROR;
}

/* ***********************************************/
/* int_jtrackballClear Clear error state of the module. */
/* ***********************************************/
int int_jtrackballClear(local, stask, mptr, ermsg, errcode)
int jtrackballLocal_t *local;
sbsTask_t *stask;

errModule_t *mptr;
char *errmsg;
int errcode;
{
/* Return after not clearing error. */
return SBS_ERROR;
}

sbsNewError(stask, "Clear not defined, still in error state", errcode);
return SBS_ERROR;
}

/* ***********************************************/
/* int_jtrackballGet Set module parameters. */
/* ***********************************************/
int int_jtrackballGet(local, stask)
int jtrackballLocal_t *local;
sbsTask_t *stask;
{ return I_OK;
}

/* ***********************************************/
/* int_jtrackballSet Get module parameters. */
/* ***********************************************/
int int_jtrackballSet(local, stask)
```c
int jtrackballLocal_t *local;
sbTask_t *task;
)
return I_OK;
}

/**
 * int jtrackballSync For modules which synchronize to something other than the clock.
 */
/**
 * int jtrackballSync(local, stask)
 * int jtrackballLocal_t *local;
 * sbTask_t *task;
 * return I_OK;
 */

/**
 * Local function that will read all remaining bytes then send a */
 * signal of hex OS to handshake */
*/
static int Clean_port(iod)
 100 *iod;
 1 char rqtst = 5;
 1 int nb = 1;
 1 char data;
while(nb > 0) /* flush the sio buffers one byte at a time */
 1 nb = iodRead(iod, &data, 1);
 1 iodWrite(iod, &rqtst, 1); /* send packet of data to start*/
 1 delay(.0055); /* hand shaking */
 1 delay(nb); /* wait for data to show in buffers */
return nb;
}

/**
 * int jtrackballSync(local, stask)
 * int jtrackballLocal_t *local;
 * sbTask_t *task;
 * return I_OK;
 */

int US_PI_READ_DATA()

byte * US_PI_READ_DATA(type, channel, bytes_read)
char *type;
char *channel;
int *bytes_read;
/* This module returns a pointer to the array which contains the data
 that is read from the trackball. */

byte *buf[12];
int I /* for debugging code */

/* Read data */
*bytes_read = iodRead(iod_var, buf, nbytes);
Clean_port(iod_var); /* clean port and send handshake */
/* for debugging
printf("--US_PI_READ_DATA: Values read from trackball are:");
for (i=4;i<10;i++)
printf("%d ", bytes_read[i]);
printf("%d ", bytes_read[10]); */
return Ibuf;

/**
 * int jtrackballSync(local, stask)
 * int jtrackballLocal_t *local;
 * sbTask_t *task;
 * return I_OK;
 */

int US_DISABLE_OI()

void US_DISABLE_OI (void)
{
  UTAP_ERROR_FLAG = UTAP_ERROR_DISABLE_OI;
}

/**
 * int jtrackballSync(local, stask)
 * int jtrackballLocal_t *local;
 * sbTask_t *task;
 * return I_OK;
 */

#define US_GET_EXT_DATA_VALUE (req_data)
char * US_GET_EXT_DATA_VALUE (req_data)
char *req_data;

if (strcmp(req_data, "GAINS") == 0)
printf("US_GET_EXT_DATA_VALUE: Returned Gains \n");
return gain;
if (strcmp(req_data, "SPEED") == 0)
printf("US_GET_EXT_DATA_VALUE: Returned Speed \n");
return speed;

printf("US_GET_EXT_DATA_VALUE: ERROR, Nothing returned for requested \"%s\" \n", req_data);
}

int US_OK_OI_OUTPUT_REGISTERED_OBJ_ID (name)
char *name;
{
    if (strcmp(name, "Q_REF") == 0)
        return 1; /* 1st Object ID for this module is 1 */
    else
        printf("US_OK_OF_OUTPUT_REGISTERED_OBJ_ID: ERROR, Unregistered Object \n\n"%s" requested \n", name);

void * US_OK_OF_ATTRIBUTE_QUERY (obj_id)
int   *obj_id;
{
    if (obj_id == 1)
        /* return pointer to Qref's current values */
        return utap_Qref;
    else
        printf("US_OK_OF_ATTRIBUTE_QUERY: ERROR - unused Object ID Requested \n\n");  
}

void US_OK_OF_MODIFY_ATTRIBUTE (obj_id, pointer_to_value)
int   *obj_id;
void *pointer_to_value;
{
    /* This module will do nothing since the UMAP module points to the same 
data that the Interface module points to. Thus, the data is already 
updated in the local state table and will be updated in the Global 
State Table at the conclusion of this cycle */
}
UTAP Operator Interface Module

/**
 * Module: UTAP_OI.c
 * 
 * created by: Kevin Anchors 13 Aug 95
 * 
 * modified: 
 */

#ifdef
This program is based on the jtrackball.c module written by
Tom Deeter at the Air Force Institute of Technology. It is
intended to be compliant with the Unified Telerobotic
Architecture Project (UTAP) Version 2.

It operates the Dim 6 trackball as an input device to chimera

INPUT: Q_REF - reference joint positions
OUTPUT: Q_REF - reference joint positions
Special notes:

*******************************************************************************/

/ * include files */
*******************************************************************************/

/* chimera.h is included only because it defines a byte */
#include <chimera.h>

/ * local data structure definition */
*******************************************************************************/

typedef struct {
    int qref_id; /* Object ID for Q_REF in Object Knowledgebase */
    float * qref; /* current value of QRef */
    int joint; /* array [6], but defined as a ptr for compatibility */
    float * gain; /* which joint is active */
    float * gain; /* values of the cmd GAINS */
    float * gain; /* array [6], but defined as a ptr for compatibility */
    float speed; /* selected speed of the movement */
    float data[6]; /* data from the track ball */
} OI_Local_t;

/* create the local 'object' Data structure */
static OI_Local_t *local;

static OI_Local_t local_var;

/* External function declarations */
*******************************************************************************/

extern void US_DISABLE_OI (void);
extern byte * US_PIO_READ_DATA (char *, char *, int *);
extern void * US_GET_EXT_DATA_VALUE (char *);
extern int US_OK_OF_OUTPUT_REGISTERED_OBJ_ID (char *);
extern void * US_OK_OF_ATTRIBUTE_QUERY (int);
extern void * US_OK_OF_MODIFY_ATTRIBUTE (int, void *);

*******************************************************************************/

/ * UTAP functions */
*******************************************************************************/

/ * US_INIT_OI Initialize the module. */
*******************************************************************************/

int US_INIT_OI (void)
{
    float *temp_speed;
    int i;

    local = &local_var;

    /* This data needs to be read from a config file */
    / * A global module will read these values from the Chimera .RMOD files */
    /* get the gain values from the global configuration file reader */
    / * for our purposes, this file reader will read the Chimera .RMOD files */
    local->gain = {float *} US_GET_EXT_DATA_VALUE("GAINS");
    print("-- US_INIT_OI: Got Gains = ");
    for (i=0;i<6;i++)
    {
        printf("%f ",local->gain[i]);
    }
    printf("\n");
    temp_speed = {float *} US_GET_EXT_DATA_VALUE("SPEED");
    local->speed = temp_speed;
    print("-- US_INIT_OI: Got Speed = \f \n ", local->speed);
    / * load local->data with zeros */
    for (i=0;i<12;i++)
        local->data[i]=0.0;
/* get the attribute ID for Qref from the Object Knowledgebase (OK). This 
* ID will be used throughout the module. Note that the OK has been 
* implemented in a distributed fashion rather than as a single module 
* because of constraints of the Chimera OS. */

printf("-- US_INIT_OI: Setting Object ID for Q_REF\n");
local->qref_id = US_OK_OI_OUTPUT_REGISTERED_OBJ_ID("Q_REF");
printf("-- US_INIT_OI: Got Object ID for Q_REF = \%d\n",local->qref_id);
local->joint = 0;
printf("-- US_INIT_OI: Returning\n");
return 0;

/* *************************************************************** */
/* US_START_OI Process module information. */
/* *************************************************************** */

int US_START_OI (void)
{ /* declare needed local variables for this function */
byte *pio_buffer; /* holds the incoming data */
int i, button, /* Holds the value of the selected button from 
the trackball */
*num_bytes_read; /* the number bytes read from the trackball */
char *cptr; /* temp pointer used in calculation */

/* Update local data from the Object Knowledgebase. Requests to OK 
must use the Object ID, qref_id is passed by value since it is a 
scalar and qref is passed by address (automatically) since it is an 
array of float */
local->qref = (float *)US_OK_OI_ATTRIBUTE_QUERY(local->qref_id);

/* Read raw data from trackball */
pio_buffer = USPIO_READ_DATA("RAW", "TRACKBALL", num_bytes_read);
/* The trackball returns 12 bytes of data to the pio_buffer array, 
but the only values of interest are bytes 4-9 and byte 10 */
/* format is USPIO_READ_DATA(type, channel, number_of_bytes_transferred)*/
if (*num_bytes_read > 0) /* data has been returned, so convert it to 
useful data */
{
/* convert the raw data into usable data */
cptr=pio_buffer;
for (i=0;i<6;i++)
local->data[i]=((float)cptr[i+4] / 128.0) * local->gain[1];
/* get the button value, which is in byte 10 of the returned data */
button= (int)pio_buffer[10];

/* for debugging: */
printf("UTAP OI: values are: ");
printf("\%f",local->data[1]);
printf("\%f",button); */
/* take action as required for buttons */
/* the buttons 1 thru 6 select the robot joint */
if ((button > 0) && (button <= 6))
{ local->joint = button - 1;
print("New axis = joint \%d\n", button);
}
/* button 7 or 8 stops trackball module */
if (button == 7 || button == 8)
{ US_DISABLE_OI();
}
/* update the local value of Qref for the selected joint */
local->qref[local->joint] = local->data[1]*local->speed;
/* Update Object Knowledgebase. Qref is passed by address since 
it is an array */
US_OK_OI_MODIFY_ATTRIBUTE(local->qref_id, local->qref);
return 0;
}
Interface Module for the Robot Servo Control
(PIDG controller and Gravity Compensation Modules)

/* *******************************************************************/
/* Module: int_RSC.c */
/* created by: Kevin Anchor 28 Sep 95 */
/* Air Force Institute of Technology */
/* modified: */
/* *******************************************************************/

This program is based on the puma pidg.c module and the
gridity compensation program gray_comp.c
created by Wayne F. Carriker 05-12-93
Carnegie Mellon University
modified by Wayne F. Carriker 06-02-93
modified for new SAI and IOD code
modified by Matthew W. Gertz 11-15-93
updated to Chimera 3.2
panic button now invokes error
modified by Tom Deeter 08-17-94
Added the joint out of range
checks
It is intended to be compliant with the Unified Telerobotic
Architecture Project (UTAP) Version 2 module utap_RSC.

combined puma 560 interface module and independent
joint PID controller with feed-forward gravity
compensation
State Variable table:
INCONST: none
OUCNST: NDOF - number of degrees of freedom
INVAR: Q_REF - reference joint positions
Q_DOT_REF - reference joint velocities
T_GRAV - gravity compensation torques
OUTVAR: Q_MEI - measured joint positions
Q_MEI - measured joint velocities

/* Special notes: */
/* This module does not supply correct DH values */
/* *******************************************************************/

#include <chimera.h>
#include <stdio.h>
#include <math.h>
#include <io.h>
#include <stdlib.h>

#define DUMMY CODE 1
#define UTAP_NO_ERROR 0
define UTAP_ERROR_DISABLE_RSC 1
define UTAP_ERROR_JOINT_OUT_OF_RANGE 2
#define UTAP_ERROR_ROBOT_NO_POWER 3

/* *******************************************************************/
/* Internal function declarations */
/* *******************************************************************/
float *US_ATTRIBUTE_GET_POSITION (void);

/* *******************************************************************/
/* External function declarations */
/* *******************************************************************/
extern int US_INIT_RSC (void);
extern int US_START_RSC (void);

/* *******************************************************************/
/* Global vars available to all functions in this module */
/* *******************************************************************/
/* Used to transfer values to/from UTAP module */
static float *utap_Q_ref;
static float *utap_Qd_ref;
static float *utap_T_grav;
static float *utap_Q_mei;
static float *utap_Qd_mei;
static float utap_task_freq;
static float gray_comp_freq;

89
static int use_gray_comp;
static double kpgains[6], klgains[6], kvgains[6];
static double *kpgain = kpgains, *klgain = klgains, *kvgain = kvgains;
static SAI
static float utap_joint[6];
static int
/* ***************************************************************/
/* module 'local_t' definition as required by Chimera */
/* ***************************************************************/
int RSCLocal_t;
#endif
/* ***************************************************************/
/* module initialization as required by Chimera */
/* ***************************************************************/

SBS_MODULE(int RSC);
/* ***************************************************************/
/* Chimera Interface Modules */
/* ***************************************************************/
/* ***************************************************************/
/* int RSCInit Initialize the module. */
/* ***************************************************************/
int int RSCInit(info, local, stack)
    configinfo_t *info;
    int RSCLocal_t *local;
    sbTask_t *stack;
{ sbSvar_t
    *svar = stack->svar;
    char
    info[i], *ninfo;
    unsigned int
csl;
    float
    *dh;
/* All Chimera-specific initialization is handled in this module. The
UTAP init module is then called to perform UTAP initialization */
/* Get pointers to state variables. */
local->Qd_ref = svarTranslateValue(svar->variable, "Q_Ref", float);
local->Qd_grav = svarTranslateValue(svar->variable, "T_Grav", float);
local->Qd_mez = svarTranslateValue(svar->variable, "Q_Mez", float);
local->Qd_ref = local->Qd_ref;
local->T_grav = local->T_grav;
/* Check if gravity compensation should be used */
/* If GRAV COMP == 0, then no gravity compensation will be used.*/
if (local->puma = NULL) {
    errInvoke(stask->errmod, "puma initialization failed", DUMMY_CODE);
    kprintf("int RSCInit: Puma is initialized\n");
    /* Ensure that the robot has been calibrated. */
saiStatus(local->puma, SAI_CALIB, &status);
    if (status != 0) {
        errInvoke(stask->errmod, "Puma is not calibrated", DUMMY_CODE);
        kprintf("int RSCInit: Puma is initialized\n");
    /* Make the local puma var available to all functions in module */
    puma = local->puma;
    /* Set the OUTCONSTAT */
    *
    ndof = svarTranslateValue(svar->variable, "NDOF", int);
    *ndof = 6;
    dh = svarTranslateValue(svar->variable, "DH", float);
    for (i = 0; i < 24; ++i)
        dh[i] = 0.0;
    /* allow UTAP module to know the task frequency */
    utap_task_freq = stask->freq;
    /* clear error flag */
UTAP_RSC_ERROR_FLAG = UTAP_NO_ERROR;
US_INIT_RSC();  /* Call UTAP Init Module */
/* Return from initialization. */
return (int) local;
}

int RSCLocalInit(local, task)
int RSCLocal_t *local;
sbsTask_t *task;
{
    return I_OK;
}

/* ***********************************************
* int RSCOn
* ***********************************************
*
int int RSCOn(local, task)
int RSCLocal_t *local;
sbsTask_t *task;
{
    int i, j, power;
    float *joint_ptr, joint[6];
    sbsSvar_t *svar = stask->svar;
    svarVar_t *svarQ_ref, *svarQd_ref;

    /* Read the measured position. */
    saiRead(local->puma, SAI_Q, joint);
    for (i = 0; i < 6; ++i)
        local->Q_Mez[i] = joint[i];

    /* Set measured velocity to zero to start. */
    for (i = 0; i < 6; ++i)
        local->Qd_Mez[i] = 0.0;

    /* Do an explicit state variable table write so that the PUMA will */
    /* will come up "self stable" ((Q_REF = Q_Mez) & (Q"_REF = Q"_MEZ)). */
    svarQ_ref = svarTranslate(svar->variant, "Q_REF");
    svarQd_ref = svarTranslate(svar->variant, "Q"_REF");
    for (i = 0; i < 6; ++i)
        local->Q_ref[i] = joint[i];
    local->Qd_ref[i] = 0.0;
    svarWrite(svarQ_ref);
    svarWrite(svarQd_ref);

    /* Turn on the PUMA in software and then wait for the hardware. */
    printf("\n\nPower up robot now...\n\n");
    saiControl(local->puma, SAI_POWER, (pointer)1);
    power = 0;
    while ((power = saiStatus(local->puma, SAI_POWER, &power)) == 1)
    {
        return I_OK;
    }

    /* ***********************************************
* int RSCCycle
* ***********************************************
*
int int RSCCycle(local, task)
int RSCLocal_t *local;
sbsTask_t *task;
{
    US_START_RSC();  /* Call UTAP Module */

    if (UTAP_RSC_ERROR_FLAG == UTAP_NO_ERROR)
        return I_OK;  /* normal exit */
    else if (UTAP_RSC_ERROR_FLAG == UTAP_ERROR_Robot_NO_POWER)
        return SBS_OFF;
    else if (UTAP_RSC_ERROR_FLAG == UTAP_ERROR_JOINT_OUT_OF_RANGE)
        return SBS_OFF;
    else
    {
        kprintf("Unknown error flag %i raised in int_RSCCycle\n",
                 UTAP_RSC_ERROR_FLAGS);
        kprintf("Turning robot off (just in case)...\n");
        return SBS_OFF;
    }

    /* ***********************************************
* int RSCOff
* ***********************************************
*
int int RSCOff(local, task)
int RSCLocal_t *local;
sbsTask_t *task;
{
/* Disable power in hardware. */
saiControl(local->puma, SAI_POWER, (pointer) 0); kprintf("Robot powered down.\n");

/* Indicate that the module is off and return. */
kprintf("int_RSC: OFF\n");
return I_OK;
}

/* *****************************/
/* int_RSCKill       */
int int_RSCKill(local, stask)
int_RSCLocal_t  *local;  *stask;
{ /* Close the external device (the PUMA). */
saiFinish(local->puma);

/* Indicate that the module is finished and return. */
kprintf("int_RSC: FINISHED\n");
return I_OK;
}

/* *****************************/
/* int_RSCError     Attempt automatic error recovery. */
/* *****************************/
int int_RSCError(local, stask, mptr, errmsg, errcode)
int_RSCLocal_t  *local;  *stask;
sbsTask_t      *mptr;
char           *errmsg;
int            errcode;
{ /* Return after not correcting error. */
    return SBS_ERROR;
}

/* *****************************/
/*******************************************************************************
/* US_RSC_GET_EXT_DATA_VALUE
*******************************************************************************

void *US_RSC_GET_EXT_DATA_VALUE(req_data) char *req_data;
    
    if (strcmp(req_data, "KPGAINS") == 0)
        return kpgain;
    
    if (strcmp(req_data, "KVGAＩNS") == 0)
        return kvgain;
    
    if (strcmp(req_data, "KIGAINS") == 0)
        return kigain;
    
    if (strcmp(req_data, "USE_GRAV_COMP") == 0)
        return suse_grav_comp;
    
    if (strcmp(req_data, "GRAV_COMP_FREQ") == 0)
        return sgrav_comp_freq;
    
    if (strcmp(req_data, "FREQ") == 0)
        return sgrav_task_freq;
    
    kprintf("US_RSC_GET_EXT_DATA_VALUE: ERROR, Nothing returned for requested
        \"%s\" \n", req_data);
    
/*******************************************************************************
/* US_OK_RSC_OUTPUT_REGISTERED_OBJ_ID
*******************************************************************************

int US_OK_RSC_OUTPUT_REGISTERED_OBJ_ID(name) char *name;
    
    if (strcmp(name, "Q_REF") == 0)
        return 1; /* 1st Object ID for this module is 1 */
    
    if (strcmp(name, "QREF") == 0)
        return 2; /* 2nd Object ID for this module is 2 */
            
if (strcmp(name, "T_GRAV") == 0)
    return 3; /* 3rd Object ID for this module is */
if (strcmp(name, "Q\_MEZ") == 0)
    return 4; /* 4th Object ID for this module is */
if (strcmp(name, "Q\_MEZ") == 0)
    return 5; /* 5th Object ID for this module is */

kprintf("US_OK_RSC_OUTPUT_REGISTERED_OBJ_ID: ERROR, Unregistered
Object \"%s\" requested \n", name);
    
/*******************************************************************************
/* US_OK_RSC_ATTRIBUTE_QUERY
*******************************************************************************

void *US_OK_RSC_ATTRIBUTE_QUERY(obj_id)
    
    int obj_id;
    
    if (obj_id == 1)
        { /* return pointer to Q\_ref's current values */
        return utap_q_ref;
    
    if (obj_id == 2)
        { /* return pointer to Q\^\_ref's current values */
        return utap_qd_ref;
    
    if (obj_id == 3)
        { /* return pointer to T\_grav's current values */
        return utap_t_grav;
    
    if (obj_id == 4)
        { /* return pointer to Q\_mez's current values */
        return utap_q mez;
    
    if (obj_id == 5)
        { /* return pointer to Q\^\_mez's current values */
        return utap_qd mez;
    
    else
        kprintf("US_OK_CI_ATTRIBUTE_QUERY: ERROR - unused Object ID
Requested \n\n");

93
void US_OK_RSC_MODIFYATTRIBUTE (obj_id, pointer_to_value)
{
    int *obj_id;
    void *pointer_to_value;

    /* This module will do nothing since the UTap module points to the same
data that the interface module points to. Thus, the data is already
updated in the local state table and will be updated in the Global
State Table at the conclusion of this cycle */
}

float *US_ATTRIBUTE_GET_POSITION (void)
{
    float *joint_ptr;
    int i;

    joint_ptr = utap_joint;
    saiRead(puma, SAI_O, joint_ptr);

    /*
     * printf("US_ATTRIBUTE_GET_POSITION: Read joints ");
     * for (i=0; i<6; i++)
     *   printf(" |",joint_ptr[i]);
     * printf("\n\n");
     */

    return joint_ptr;
}

void US_ERROR_LIMIT_EXCEEDED (error_msg)
{
    char *error_msg;

    UTAP_RSC_ERROR_FLAG = UTAP_ERROR_J0INT_OUT_OF_RANGE;
    kprintf("UTAP_ERROR: %s",error_msg);
}

void US_RSC_LOAD_TORQUES (torque_ptr)
{
    float *torque_ptr;
    int i;

    /* printf("US_RSC_LOAD_TORQUES: Writing Torques ");
     * for (i=0; i<6; i++)
     *   printf(" |",torque_ptr[i]);
     * printf("\n\n");
     */

    saiWrite(puma, SAI_T, torque_ptr);
}

void US_ERROR_ROBOT_POWER (error_msg)
{
    char *error_msg;

    utap_rsc_error_flag = utap_error_robot_no_power;
    kprintf("UTAP_ERROR: %s",error_msg);
}
UTAP Robot Servo Control Module
(FIDG controller and Gravity Compensation Modules)

/* ***********************************************************************/
/* Module: utap_fsc.c */
/* created by: Kevin Anchor 20 Sep 95 */
/* Air Force Institute of Technology */
/* modified: */
/* *************************************************************************/

This program is based on the puma_pidg.c module

created by Wayne F. Carriker 05-12-93
Carnegie Mellon University
modified by Wayne F. Carriker 06-02-93
adapted for new SAI and IGO code
modified by Matthew W. Gertz 11-15-93
updated to Chimesa 3.2
panic button now invokes error
modified by Tom Deeter 08-17-94
Added the joint out of range checks

It is intended to be compliant with the Unified Telerobotics
Architecture Project (UTAP) Version 2.

="/**************************************************************************/

/ \combined PUMA 560 interface module and independent
/ \joint FIDG controller with feed-forward gravity
/ \ compensation torques

State variable table:
INCONST: none
OUTCONST: NDOF - number of degrees of freedom
INVAR: Q_REP - reference joint positions
Q"_REF - reference joint velocities
T_LGRAV - gravity compensation torques
OUTVAR: Q_MEZ - measured joint positions
Q"_MEZ - measured joint velocities

="/**************************************************************************/

/* Special notes: */
/* This module does not supply correct DH values */
/* *************************************************************************/

#include <math.h>
#include "puma.h"

/* *************************************************************************/

/* macro definitions */
/* *************************************************************************/

/* for FIDG controller */
#define MASKA 0x0f
#define MASKB 0x03
#define AVGA 5
#define INVAGA .2
#define AVGB 3
#define INVAGB ,33333333333333
#define DORRY_CODE 1
#define J10 -2.7925268 /* -160 degrees in rads */
#define J11 2.7925268 /* 160 degrees in rads */
#define J20 -3.9269825 /* -225 degrees in rads */
#define J21 3.9269825 /* 225 degrees in rads */
#define J30 0.6981317 /* 40 degrees in rads */
#define J31 -0.6981317 /* -40 degrees in rads */
#define J40 4.0491639 /* 232 degrees in rads */
#define J41 -4.0491639 /* -232 degrees in rads */
#define J50 1.9198622 /* 110 degrees in rads */
#define J51 -1.9198622 /* -110 degrees in rads */
#define J60 2.9670597 /* 170 degrees in rads */
#define J61 -2.9670597 /* -170 degrees in rads */
#define J70 1.7453293 /* 100 degrees in rads */
#define J71 -1.7453293 /* -100 degrees in rads */
#define J6O -4.712389 /* -270 degrees in rads */
#define J61 4.712389 /* 270 degrees in rads */

/* for gravity compensation */
extern unsigned * DORRY;
#define VGRAV (unsigned t; if ((_DORRY != 0)
(_VWHERE((0x00000001), _VALUE((0x00000014), t=*_DORRY) rff ((t & 0x00000000) != 0)
abort(1));)
#define rot26(s,c,I,O) o[0] = I[0] * c - I[1] * s; 
 o[I] = I[0] * s + I[1] * c; 
 o[2] = I[2];
#define icrot26(s,c,I,O) o[0] = I[0] * c + I[1] * s; 
 o[I] = I[0] * s - I[1] * c; 
 o[2] = I[2];
#define rot35(s,c,I,O) o[0] = I[0] * c + I[1] * s; 
 o[I] = I[0] * s - I[1] * c; 
 o[2] = I[2];
#define icrot35(s,c,I,O) o[0] = I[0] * c + I[1] * s; 
 o[I] = I[0] * s - I[1] * c; 
 o[2] = I[2];

95
#define rot14(s, c, i, o)  o[2] = i[0] * s + i[1] * c;  
    o[1] = i[0] * c - i[1] * s;  
    o[2] = i[1] * c + i[0] * s;  
    o[2] = -i[0] * i[1];  
#define rot14(c, s, i, o)  o[0] = i[0] * s + i[1] * c;  
    o[1] = i[0] * c - i[1] * s;  
    o[2] = i[1] * c + i[0] * s;  
    o[2] = -i[0] * i[1];  

#define add2vec(i, i2, o)  o[0] = i[0] * i[0] + i[2][0];  
    o[1] = i[1] * i[1] + i[2][1];  
    o[2] = i[2][1] * i[2][2];  
    o[0] = i[2][1] * i[2][2] - i[1][1] * i[2][2];  
    o[0] = s[0][0] * s[0][1];  
    o[1] = s[0][0] * s[0][2];  
    o[2] = s[0][2] * s[0][2];  

extern void *US_RSC_GET_EXT_DATA_VALUE (char *);  
extern int US_OK_RSC_OUTPUT_REGISTERED_OBJ_ID (char *);  
extern void *US_OK_RSC_ATTRIBUTE_QUERY (int);  
extern void US_OK_RSC_MODIFYATTRIBUTE (int, void *);  
extern float *US_ATTRIBUTE_GETPOSITION (void);  
extern int US_RSC_CHECK_DROBOT_POWER (void);  
extern void US_ERROR_LIMIT_EXCEEDED (char *);  
extern void US_RSC_LOAD_TORQUE (float *);  
extern void US_ERROR собОBOT_POWER (char *);  

/* **********************************************************/  
/* UTP functions */  
/* **********************************************************/  

void US_START_GRAV_COMP (void);  
void US_END_GRAV_COMP (void);  

/* **********************************************************/  
/* US_INIT_RSC */  
/* **********************************************************/  

void US_INIT_RSC (void)  
{  
    int i, j, k;  
    local = &local_var;  
    /* get the attribute IDs for needed vars from the Object */  
    Knowledgebase (OK). These IDs will be used throughout the module. */  
    Note that the OK has been implemented in a distributed fashion rather than as a single module because of constraints of the Chimera OS. */  
    local->Q_ref_id = US_OK_RSC_OUTPUT_REGISTERED_OBJ_ID("Q_REF");  
    local->Q_grav_id = US_OK_RSC_OUTPUT_REGISTERED_OBJ_ID("Q_GRAV");  
    local->Q_mez_id = US_OK_RSC_OUTPUT_REGISTERED_OBJ_ID("Q_MEZ");  
    /* Get values from config file (which is the .rmd file in our case) */  
    local->kp = (double *)US_RSC_GET_EXT_DATA_VALUE("KPGAINS");  
    local->ki = (double *)US_RSC_GET_EXT_DATA_VALUE("KIGAINS");  
    local->kv = (double *)US_RSC_GET_EXT_DATA_VALUE("KVGAINS");  
    local->grav_com = (int)US_RSC_GET_EXT_DATA_VALUE("US Hôm GRAV COM");  
    local->Grav_comp_freq = *(float)US_RSC_GET_EXT_DATA_VALUE("GRAV_COMP_FREQ");  
    local->Task_freq = *(float)US_RSC_GET_EXT_DATA_VALUE("TFREQ");  
    printf("US_INIT_RSC: KP gains are: ");  
}  

*/
for (i = 0; i < 6; ++i)
    printf("%f", local->kp[i]);
printf("\n");
printf("US_INIT_RSC: KV gains are: ");
for (i = 0; i < 6; ++i)
    printf("%f", local->kv[i]);
printf("\n");
printf("US_INIT_RSC: KI gains are: ");
for (i = 0; i < 6; ++i)
    printf("%f", local->ki[i]);
printf("\n");
printf("US_INIT_RSC: Task frequency is %f Hz\n", local->task_freq);
if (local->use_grav_comp == 1)
    printf("US_INIT_RSC: Using Gravity Compensation\n");
else
    printf("US_INIT_RSC: Not using Gravity Compensation\n");

/* Initialize integral control to zero. */

for (k = 0; k < 6; ++k)
    local->integral[k] = 0.0;

/* Zero position storage to start. */

for (i = 0; i < 3; ++i)
    for (j = 0; j < 16; ++j)
        if (j < 4)
            local->posw[i][j] = 0.0;
        else
            local->posm[i][j] = 0.0;

/* Reset all the velocity calculation values. */

local->indexa = AVG; local->indexb = AVG;
local->indexx = 0;
local->indexb = 0;
local->ncycle = 0;
local->itimebaseA = INAVGA * local->task_freq;
local->itimebaseB = INAVGB * local->task_freq;
/* was: local->itimebaseA = INAVGA * task_freq;
local->itimebaseB = INAVGB * task_freq; */

//************************************
/* US_START_RSC */
// Process module information. */

void US_START_RSC (void)
{
    static short counter = 0;
    int flag, i, power, val;
    float *joint, torque[6];
    double *pm[3], *pw[3];
    double cosine[6], sine[6], temp;

    /* Get VARS from the Object Knowledgebase */
    local->Q_ref = (float *) US_OK_RSC_ATTRIBUTE_QUERY(local->Q_ref_id);
    local->Qd_ref = (float *) US_OK_RSC_ATTRIBUTE_QUERY(local->Qd_ref_id);
    local->Q_grav = (float *) US_OK_RSC_ATTRIBUTE_QUERY(local->Q_grav_id);
    local->Qd_grav = (float *) US_OK_RSC_ATTRIBUTE_QUERY(local->Qd_grav_id);
    local->Q_mez = (float *) US_OK_RSC_ATTRIBUTE_QUERY(local->Q_mez_id);
    local->Qd_mez = (float *) US_OK_RSC_ATTRIBUTE_QUERY(local->Qd_mez_id);

    /* Read measured positions from the hardware. */
    joint = US_ATTRIBUTES_GET_POSITION();
    for (i = 0; i < 6; ++i)
        local->Q_mez[i] = joint[i];

    /* Compute "measured" velocity. */
    for (i = 0; i < 3; ++i)
        for (j = 0; j < 16; ++j)
            if (j < 4)
                pm[i][j] = local->posw[i][j];
            else
                local->posm[i][j] = 0.0;

    for (i = 0; i < 3; ++i)
        for (j = 0; j < 16; ++j)
            if (j < 4)
                pw[i][j] = local->posm[i][j];
            else
                local->posw[i][j] = 0.0;

    if (local->ncycle >= AVG)
        for (i = 0; i < 3; ++i)
            local->Qd_mez[i] = (float) ((pm[i][local->indexa] -
                pm[i][local->indexb]) * local->itimebaseA);
    for (i = 0; i < 3; ++i)
        local->Qd_mez[i] = (float) ((pw[i][local->indexb] -
            pw[i][local->indexa]) * local->itimebaseB);
    local->ncycle++;
    local->indexa = ++(local->indexa) & MASKA;
    local->indexb = ++(local->indexb) & MASKB;
local->indexb = ++(local->indexb) & MASKB;
local->jindexb = ++(local->jindexb) & MASKB;

/* Check if qref values are out of range */
flag=0;
if ((local->Q_ref[0]<J1LO || local->Q_ref[0]>J1HI) && flag=1;
if (flag=0) {
    printf("Joint angle %d out of range: puma_pidg shutdown\n",flag);
    US_ERROR_LIMIT_EXCEEDED("Out-Of-Range");
    return;
}

/* Perform independent joint PID control with gravity compensation. */
/* Perform gravity compensation at the rate specified in the config file, if it is turned on either in the config file or by another UTAP message */
if ((counter % (int)local->grav_comp_freq) == 0) /* Is it time? */
    if (local->use_grav_comp == 1) /* Is gravity compensation on? */
        /* Precompute sines and cosines. Only compute values for joints two through five because we will ignore joints one and six for speed. (Joint one needs no gravity compensation, and joint six needs very little compensation. */
        for (i = 1; i < 5; ++i) {
            temp = (double)local->q_mez[i];
            cosine[i] = cos(temp);
            sine[i] = sin(temp);
        }
        /* Call the gravity compensation routine */
        gravcomp(sine, cosine, local->T_grav);
*/

/* Compute position error. */
for (i = 0; i < 6; ++i) {
    pe[i] = ((double)local->Q_ref[i]) - ((double)local->Q_mez[i]);
}

/* Compute integral of position error. */
for (i = 0; i < 6; ++i) {
    local->integral[i] += pe[i];
}

/* Compute velocity errors. */
for (i = 0; i < 6; ++i) {
    ve[i] = ((double)local->Qd_ref[i]) - ((double)local->Qd_mez[i]);
}

/* Compute torques (including feed-forward torques). */
for (i = 0; i < 6; ++i) {
    torque_ptr[i] = (float)(local->ki[i] * local->integral[i] +
        local->kx[i] * pe[i] +
        local->kv[i] * ve[i] +
        (double)(local->T_grav[i]));
}

/* Write torques to the hardware */
US_RSC_LOAD_TORQUES(torque_ptr);

/* Every 256 cycles check to see if the power is still on. */
counter++;
if ((counter % 256) == 0) {
    power = US_RSC_CHECK_ROBOT_POWER();
    if (!power) {
        US_ERROR_ROBOT_POWER("The Puma has lost power.");
    }
}

/* Update OUTVARS in Object Knowledgebase */
US_OK_RSC_MODIFY_ATTRIBUTE(local->T_grav_id, local->T_grav);
US_OK_RSC_MODIFY_ATTRIBUTE(local->Q_grav_id, local->Q_grav);
US_OK_RSC_MODIFY_ATTRIBUTE(local->Q_mez_id, local->Q_mez);
US_OK_RSC_MODIFY_ATTRIBUTE(local->Qd_mez_id, local->Qd_mez);
static double q0[3] = { 0.0, 0.0, 0.0};
static double q2[3] = { 0.068-PUMA560_A2, 0.006, 0.0934-0.016};
static double q3[3] = { PUMA560_A3, 0.014, 0.070};
static double q4[3] = { 0.0, 0.019, 0.0};
static double q5[3] = { 0.0, 0.0};
static double q6[3] = { 0.0, 0.0, 0.0};

/* The following masses and centers of gravity are from a paper by Armstrong, Khatib, & Burdick. */
static double m2 = 23.0;
static double m3 = 4.80;
static double m4 = 0.20;
static double m5 = 0.34;
static double m6 = 1.95; /* include estimation of gripper & FTS */
static double a1[3] = { 0.0, -9.9, 0.0}; /* fixed base acceleration */
double a2[3], a3[3], a4[3], a5[3], a6[3];
double f2[3], f3[3], f4[3], f5[3], f6[3];
double n2[3], n3[3], n4[3], n5[3], n6[3];
double c2, c3, c4, c5, c6, s2, s3, s4, s5;

/* Make local copies of sines and cosines. */
c2 = *(+c);  s2 = *(+s);
c3 = *(+c);  s3 = *(+s);
c4 = *(+c);  s4 = *(+s);
c5 = *(+c);  s5 = *(+s);

/* Perform the forward recursion. */
/* Obtain a1. */
irot26(s2, c2, a1, a2);
irot35(s3, c3, c2, a3);
irot14(s4, c4, a3, a4);
irot35(s5, c5, a4, a5);

/* Perform the backward recursion. */
/* Obtain F6, M6, f6, n6, t[5]. */
scaVarmult(m6, a5, f6);
xprod(q6, f6, fullt);
/ * rotations.
 */

scalarmult(m5, a5, F5);
add2vecs(f6, F5, F5);
xprod(q6, F5, tempa);
add2vecs(fullt, tempa, n5);
rot35(a5, c5, n5, fullt);
t[4] = fullt[2];

/* Obtain t[3].
 */

scalarmult(m4, a4, F4);
rot35(a5, c5, f5, tempa);
add2vecs(tempa, F4, F4);
xprod(q4, F4, tempa);
add2vecs(fullt, tempa, tempb);
xprod(q4, F4, tempa);
add2vecs(tempb, tempa, n4);
rot14(a4, c4, n4, fullt);
t[3] = fullt[2];

/* Obtain t[2].
 */

scalarmult(m3, a3, F3);
rot14(a4, c4, f4, tempa);
add2vecs(tempa, F3, F3);
xprod(q3, F3, tempa);
add2vecs(fullt, tempa, tempb);
xprod(q3, F3, tempa);
add2vecs(tempb, tempa, n3);
rot35(a3, c3, n3, fullt);
t[2] = fullt[2];

/* Obtain t[1].
 */

scalarmult(m2, a2, F2);
rot35(a3, c3, f3, tempa);
add2vecs(tempa, F2, F2);
xprod(q2, F2, tempa);
add2vecs(fullt, tempa, tempb);
xprod(q2, F2, tempa);
add2vecs(tempb, tempa, n2);
rot26(a2, c2, n2, fullt);
t[1] = fullt[2];}
Object Knowledgebase
(This is the attempt that was made to implement the OK as a separate module. See Chapter 3 for a discussion of why this code was not used)
/* ***************************************************/
/* Module: utap_OK.c */
/* created by: Kevin Anchor 7 Sep 93 */
/* Air Force Institute of Technology */
/* */
/* modified: */
/* */
/* */
/* This module implements the UTAP Object Knowledgebase module. */
/* It uses the Chimera global state table, so it is operating */
/* system (specifically, Chimera) dependent. Interfaces to both */
/* Chimera and UTAP are provided in this one module. */
/* */
/* Note that the Object Knowledgebase was not implemented using */
/* this code. It is provided only to show the initial approach. */
/* See Chapter 3 of the Thesis for a discussion of this subject. */
/* ***************************************************/

#include <chimera.h>
#include <sba.h>

/*==================================================================*/
/* Define */
/*==================================================================*/
#define UTAP_OK 1
#define UTAP_OK_OFF 0

/*==================================================================*/
/* External function declarations */
/*==================================================================*/

/*==================================================================*/
/* Local_T definition as required by Chimera */
/*==================================================================*/

/*==================================================================*/
/* Can the position in the structure be used as the ID? Yes, that is */
/* how it is handled in this module. The Chimera index is used as the */
/* ID */
/*==================================================================*/

/*==================================================================*/
/* How are new attributes added 'on the fly' during runtime? For our */
/* system, all data needed must be known ahead of time */
/*==================================================================*/

/* local data for the OK module */
static sbsSvar_t *svar;
static int utap_OK_status;

typedef struct {
  int dummy;
} utap_OKLocal_t;

/*==================================================================*/
/* Module initialization as required by Chimera */
/*==================================================================*/

SBS_MODULE(utap_OK);

/*==================================================================*/
/* Initialize the module. */
/*==================================================================*/

int utap_OKInit(config_t *config, local_t *local, stack_t *stack)
{
  svar = &stack->svar;
kprintf("utap_OKInit: svar = %p \n", svar);
kprintf("utap_OKInit: stack->svar = %p \n", &stack->svar);
  utap_OK_status = UTAP_OK_OFF;
  return (int) local;
}

/*==================================================================*/
/* Re-Initialize the module. */
/*==================================================================*/

int utap_OKReinit(local, stack)
{
  utap_OKLocal_t *local;
  sbsTask_t *task;
  
  return 1;
}

/*==================================================================*/
/* Start up the module. */
/*==================================================================*/

101
int upat_OKOn(local, stack)
upat_OKLocal_t *local; *task;

sbsTask_t *task;

printf("Object Knowledgebase: ON\n");

upat_OK_status = UPAT_OK_ON;
kprintf("\nupat_OKOn: status is \d \n\", upat_OK_status);

return I_OK;

int upat_OKCycle(local, stack)
upat_OKLocal_t *local; *task;

sbsTask_t *task;

return I_OK;

int upat_OKOff(local, stack)
upat_OKLocal_t *local; *task;

sbsTask_t *task;

kprintf("upat_OK: OFF\n");

upat_OK_status = UPAT_OK_OFF;

return I_OK;

int upat_OKKill(local, stack)
upat_OKLocal_t *local; *task;

sbsTask_t *task;

kprintf("upat_OK: FINISHED\n");

upat_OK_status = UPAT_OK_OFF;

return I_OK;

int upat_OKError(local, stack, mptr, errmsg, errcode)
upat_OKLocal_t *local; *task;

sbsTask_t *task; *mptr;

char *errmsg;

int errcode;

/* Return after not correcting error. */

return SBS_ERROR;

int upat_OKClear(local, stack, mptr, errmsg, errcode)
upat_OKLocal_t *local; *task;

sbsTask_t *task; *mptr;

char *errmsg;

int errcode;

/* Return after not clearing error. */

sbsNewError(task, "Clear not defined, still in error state", errcode);

return SBS_ERROR;

int upat_OKSet(local, stack)
upat_OKLocal_t *local; *task;

sbsTask_t *task;

/* Set module parameters. */

int upat_OKGet(local, stack)
upat_OKLocal_t *local; *task;

sbsTask_t *task;

/* Get module parameters. */

int upat_OKSet(local, stack)
upat_OKLocal_t *local; *task;

sbsTask_t *task;

/* */
int utap_OKSync(local, stask)
  utap_OKLocal_t *local;  /*local;  
  stask_t   *stask;
  return I_OK;
}

int US_OK_OUTPUT_REGISTERED_OBJ_ID(name)
char *name;
  /* returns the index value of the requested object. This value
represents the position of the object in the State Variable
Table. I_ERROR is returned if the specified object does not
exist in the Table or if the module is not turned on. */
kprintf("utap_OK: Getting Requested Object ID...");
kprintf("utap_OK: svar address = ", svar);
  return svarIndex(svarTranslate(svar, name));

void US_OK_MODIFY_ATTRIBUTE (obj_id, pointer_to_value)
int   obj_id;
void *pointer_to_value;
  svarWriteDirectLx(svar, obj_id, pointer_to_value);
VITA

1st Lt Kevin Anchor was born on 6 Apr 1969 in Aberdeen, MD. He graduated from Westside High School in Augusta, GA, in 1987 and enrolled at Clemson University in Clemson, SC, in the fall of 1987. He received a Bachelor of Science degree in Electrical Engineering on December 12, 1991, and he received his commission in the United States Air Force that same day.

His first assignment was at Wright-Patterson Air Force Base, OH, as a staff engineer in the Avionics Directorate of the Aeronautical Systems Center. He then moved to the B-2 System Program Office, also at Wright-Patterson AFB, as the Responsible Engineer for several of the B-2's avionics systems. In June of 1994, he entered the School of Engineering at the Air Force Institute of Technology.

Permanent Address: 8805 Gardner's Mill Road
Augusta, GA 30909
**Title:** Design and Evaluation of Standard Telerobotic Control Software

**Abstract:**
This thesis represents the first implementation of a proposed Air Force standard telerobotic control architecture. This architecture was developed by the NASA Jet Propulsion Laboratory and the National Institute of Standards and Technology under contract to the Air Force Materiel Command Robotics and Automation Center of Excellence (RACE) as the Unified Telerobotics Architecture Project (UTAP).

The AFIT Robotics and Automation Applications Group (RAAG) Lab B facility computational structure was redesigned to be compliant with the UTAP architecture. This thesis shows that the UTAP specification is implementable. However, if the underlying operating system does not support generic message passing, an interface layer must be implemented to access operating system functions.

The UTAP compliant controller implemented the robot servo control, object knowledge base, and user interface components of the specification. The controller performed adequately although there was degradation in the performance as evidenced by increased error during trajectories. We believe this error can be reduced by re-tuning the controller gains.

Further study of the UTAP specification is recommended: additional functions such as external sensor readings should be added; implementation of the specification on different operating systems and robot platforms will prove the transportability of the specification.
### GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Agency Use Only (Leave blank).</strong></td>
</tr>
<tr>
<td>2</td>
<td><strong>Report Date.</strong> Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.</td>
</tr>
<tr>
<td>3</td>
<td><strong>Type of Report and Dates Covered.</strong> State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).</td>
</tr>
<tr>
<td>4</td>
<td><strong>Title and Subtitle.</strong> A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.</td>
</tr>
<tr>
<td>5</td>
<td><strong>Funding Numbers.</strong> To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:</td>
</tr>
</tbody>
</table>
|         | - C - Contract  
|         | - G - Grant     
|         | - PE - Program 
|         | - TA - Task    
|         | - WU - Work Unit Element Accession No.                                                          |
| 6       | **Author(s).** Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s). |
| 7       | **Performing Organization Name(s) and Address(es).** Self-explanatory.                         |
| 8       | **Performing Organization Report Number.** Enter the unique alphanumeric report number(s) assigned by the organization performing the report. |
| 9       | **Sponsoring/Monitoring Agency Name(s) and Address(es).** Self-explanatory.                   |
| 10      | **Sponsoring/Monitoring Agency Report Number. (If known)**                                    |
| 11      | **Supplementary Notes.** Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report. |
| 12a     | **Distribution/Availability Statement.** Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR). |
|         | **DOD -** See DoDD 5230.24, "Distribution Statements on Technical Documents."
|         | **DOE -** See authorities.                                                                      |
|         | **NASA -** See Handbook NHB 2200.2.                                                             |
|         | **NTIS -** Leave blank.                                                                        |
| 12b     | **Distribution Code.**                                                                          |
|         | **DOD -** Leave blank.                                                                          |
|         | **DOE -** Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports. |
|         | **NASA -** Leave blank.                                                                        |
|         | **NTIS -** Leave blank.                                                                         |
| 13      | **Abstract.** Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report. |
| 14      | **Subject Terms.** Keywords or phrases identifying major subjects in the report.               |
| 15      | **Number of Pages.** Enter the total number of pages.                                          |
| 16      | **Price Code.** Enter appropriate price code (NTIS only).                                      |
| 17-19   | **Security Classifications.** Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page. |
| 20      | **Limitation of Abstract.** This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited. |