12-1996

Analysis and Design of Standard Telerobotic Control Software

Matthew L. June

AFIT/GCS/ENG/96D-12

ANALYSIS AND DESIGN OF STANDARD TELEROBOTIC CONTROL SOFTWARE

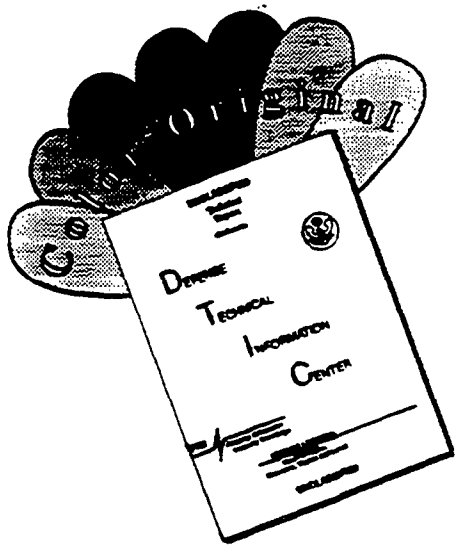
THESIS

Matthew L. June, Capt, USAF

AFIT/GCS/ENG/96D-12

Approved for public release; distribution unlimited

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

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.

AFIT/GCS/ENG/96D-12

ANALYSIS AND DESIGN OF STANDARD TELETROBOTIC 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 Computer Science

Matthew L. June, B.S.C.S.

Capt, USAF

December 1996

ACKNOWLEDGMENTS

First and foremost, I would like to thank my wife, Cindy. Thoughout my career, she has provided unflinching support and encouragement. She has patiently endured numerous moves, many TDYs, and two degree programs. During all of these stressful times, she provided strength and motivation to me and our children. Without her, my accomplishments would be few. I would also like to thank Maj Mark Kanko. His classroom presence is exceptional and his advocacy of the software engineering field is commendable. Additionally, his Christian support during a family member's illness meant a great deal to me.

Finally, I would like to thank Maj Dean Schneider for recruiting me for this research. His excitement about the robotics field is contagious and made the research fun. His knowledge of the robotics field is vast and made the research possible.

Table of Contents

List of Figures

List of Tables

ABSTRACT

The Robotics and Automation Center for Excellence (RACE) at Kelly Air Force Base, Texas, has defined an open telerobotics control architecture. This architecture, called the Unified Telerobotic Architecture Project (UTAP), is a proposed standard for all Air Force telerobotic systems. Implementation of UTAP will reduce the cost of robotic applications by increasing software modularity, portability, and reusability. This thesis continued the effort to prove the feasibility of UTAP.

In December, 1995, 1st Lt Anchor implemented a portion of the UTAP specification on a PUMA robot. The UTAP-compliant controller exhibited some degradation in the system performance. However, the performance degradation was not fully measured. This thesis extended the measurements of Anchor's implementation.

Additionally, a portion of the UTAP specification was implemented on an Adept 550 manipulator and the performance effects were measured. The implementation included portions of the generic, robot/axis servo control, tool control, sensor control, programmable IO, subsystem task level control, task description and supervision, parent task program sequencer, task program sequencer, and object knowledge modules.

Performance measurements of the Adept implementation indicated that, although performance was adversely affected, the degradation was caused by the interface between the UTAP-compliant application and the non-UTAP-compliant operating system. There was little difference between the complaint and non-compliant applications.

Successful implementation of the UTAP specification on the PUMA and Adept manipulators proves that UTAP is a feasible telerobotic architecture. However, further

study of the specification is recommended. Specifically, the development of a UTAP-compliant operating system should be continued.

ANALYSIS AND DESIGN OF STANDARD TELEROBOTIC CONTROL SOFTWARE

I. Introduction

The Robot Industries Association (RIA) 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" [12]. A telerobotic system couples the robot's abilities with a human operator's abilities to think and react. There are three basic types of telerobotic systems based on the amount and type of interaction between the robot and the human. The first type of telerobotic system is "operator controlled." With systems of this type, the operator has complete, real-time control of the robot's movements. The second type is "operator supervised." In this case, the robot obeys a control program and the operator does not have direct, real-time control. However, the operator can stop the robot or change the program as required. The final type is "shared control." This is a combination of the other types. The operator exerts some real-time control over the robot while the control program manages the remaining aspects of the task [12].

In each of these systems, the operator provides some sort of input to the system and receives some type of output or feedback. A telerobotic control architecture defines the methodology behind providing the input and receiving the output. Currently, each telerobotic system has its own control architecture.

Because each system has its own unique control architecture, it is very difficult to port a task, or function, from one system to operate on another system. Therefore, there is very little reuse of software in the telerobotic field. This results in expensive one-of-a-kind installations. To lower costs and improve efficiency and productivity, telerobotic systems must be modular, portable, and easily reconfigurable. A standard telerobotic control architecture would provide a framework for developing systems that meet these objectives.

Because the Air Force desires the use of telerobotic systems for many critical tasks including aircraft painting and paint stripping, surface cleaning, and sealing and desealing of fuel tanks, the Robotics and Automation Center for Excellence (RACE) at Kelly Air Force Base, Texas, has defined an open telerobotics control architecture. This architecture, called the Unified Telerobotic Architecture Project (UTAP), is a proposed standard for all Air Force telerobotic systems. UTAP supports a modular development that ensures plug-and-play functionality. The standardized interface and modularity of UTAP mean that a telerobot can be switched from one task to another simply by switching modules. The modularity of UTAP also increases the potential reuse of modules on new tasks.

I.1. Problem

Before I conducted my research, UTAP had only been implemented on one Air Force system. This was accomplished under a previous AFIT thesis effort conducted by 1st Lt Kevin Anchor. He implemented a portion of UTAP on the PUMA robot in the

AFIT Robotics Laboratory [3]. However, the effects of his implementation on the performance of the robot were not completely measured.

In addition to measuring the performance of Anchor's UTAP implementation, other systems needed to be retrofitted to the UTAP specification before UTAP could be considered feasible. Therefore, I implemented the UTAP specification on the Adept 550 robot in the robotics laboratory and measured the performance effects of the implementation.

I.2. Summary of Current Knowledge

Anchor's thesis work involved implementing the UTAP specification on the PUMA 560 manipulator using the Chimera 3.2 real-time operating system. Anchor measured the performance effects of UTAP by commanding the robot through a series of motions at various speeds and recording the difference between the commanded and the actual positions. Measurements were taken on the system both before and after implementing UTAP. He then plotted the amount of error between commanded and actual robot position and performed statistical analysis on the data [3]. While this method gave a general idea of the performance issues associated with his implementation, several additional steps needed to be completed before a thorough understanding was obtained. These steps included controller gain tuning and modifying the trajectory generation module to allow step response testing. They are defined in later sections.

In addition to Anchor's work, the Advanced Cybernetics Group, Inc. (ACG) developed some applications that embodied some of the concepts of the information

module portion of the UTAP specification [5]. ACG's code provided a foundation for the implementation of UTAP on the Adept 550.

I.3. Scope

This research effort was divided into two separate tasks; performance testing of Anchor's UTAP implementation and implementation of UTAP on the Adept robot. The performance testing of Anchor's UTAP implementation was conducted first. I only tested the UTAP functionality implemented by Anchor and did not implement any further UTAP functionality on the PUMA manipulator. Second, portions of the following modules were implemented on the Adept 550 manipulator using the V+ operating system:

- generic
- robot/axis servo control
- tool control
- sensor control
- programmable IO
- subsystem task level control
- task description and supervision
- parent task program sequencer
- task program sequencer
- object knowledge

This task also included performance testing.

I.4. Approach/Methodology

As stated earlier, my research consisted of two major tasks; performance testing of Anchor's UTAP implementation and implementation of UTAP on the Adept system.

I.4.1 Performance Testing of Anchor's UTAP Implementation

Performance testing was conducted using the following general steps:

a. Using the non-UTAP-compliant controller, command the robot through a series of motions at varying speeds while recording the commanded and actual joint position values.

b. Tune the controller gains for optimal performance of Anchor's UTAP implementation.

c. Using Anchor's UTAP-compliant controller, command the robot through a series of motions at varying speeds while recording the commanded and actual joint position values.

d. Modify the Chimera trajectory generation module to allow step response testing

e. Using Anchor's UTAP-compliant architecture and the modified trajectory generation module, command the robot to perform a step motion while recording the commanded and actual joint position values.

f. Analyze the data.

I.4.2 Implementation of UTAP on the Adept Manipulator

Implementation of the UTAP specification on the Adept manipulator was conducted using the follow steps:

a. Determine an appropriate application to convert to UTAP compliance.

b. Analyze the application to determine which program statements require modification.

c. Convert the program statements to UTAP messages.

d. Develop an interface that implements the UTAP messages in the V+ programming language.

e. Develop a test routine that measures the running time of the application.

f. Measure the performance of the non-UTAP and UTAP-compliant versions of the application by executing the test routine with the robot speed set to 25, 50, 75, and 100 percent of maximum.

g. Collect metrics on source lines of code and number of procedure calls for both versions of the application.

h. Analyze the data.

I.5. Materials and Equipment

All necessary equipment and software were available in the AFIT Robotics Laboratory. The software and equipment required to test Anchor's UTAP implementation included the following:

PUMA 560 manipulator
VMEbus hardware associated with the PUMA
Sun Sparc 2 workstation
Chimera 3.2 real-time operating system
SunOS 4.1.3 operating system
UTAP software developed by Anchor

The software and equipment required to implement the UTAP specification on the Adept robot included the following:

Adept 550 manipulator with MV-19 VME controller
VMEbus hardware associated with the Adept
V+ 11.1 operating system and programming environment
ACG UTAP information module V+ code

I.6. Thesis Organization

This thesis report is divided into five chapters. Chapter I is an introduction to the topic and contains background information. A literature review of current robotic architecture work and associated information is presented in Chapter II. Chapter III contains the methods and procedures used to measure the performance of Anchor's UTAP implementation, as well as an evaluation of the research results. Chapter IV describes the implementation of UTAP on the Adept system and presents an analysis of the results. Finally, Chapter V contains the conclusions drawn from this research and makes recommendations for the UTAP specification and future research.

II. Literature Review

In this literature review, I present an initial examination of the current work in the field of robotic architectures. I have limited the examinations to work which supports the Air Force's need for an open telerobotic architectural standard. The review describes the Unified Telerobotic Architecture Project (UTAP) specification, which is being considered by the RACE to become the Air Force architecture standard for robotic systems. 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 to measure the performance of the converted system.

The review begins with the UTAP specification. The specification is summarized and its purpose and goals are presented. Then, it discusses research covered by a previous AFIT thesis on the UTAP specification and describes the remaining research. This is followed by an investigation of a commercial application of the UTAP specification. Finally, the review presents a comparison of the UTAP effort to other standardization efforts.

II.1. Unified Telerobotic Architecture Project Specification

The Unified Telerobotic Architecture Project is described in [9]. The UTAP specification describes a standard interface environment that promotes the development of modular, portable, and reusable robotic applications. To help avoid point solutions to specific applications, the UTAP architecture accommodates different types of robotic manipulators with different degrees of freedom. It also accommodates different part

materials and geometrys. Additionally, it provides a facility to upgrade or change equipment, sensors, and feedback mechanisms as technology advances.

The UTAP architecture is composed of hardware and software modules. Each module has defined inputs, outputs, and responsibilities. All data inside a module is self-contained and hidden from other modules. The Object Knowledge module stores data that is needed by multiple modules. Data and control flow are passed between modules through the use of pre-defined messages. Thus, the UTAP specification defines the modules and interfaces that make up the system.

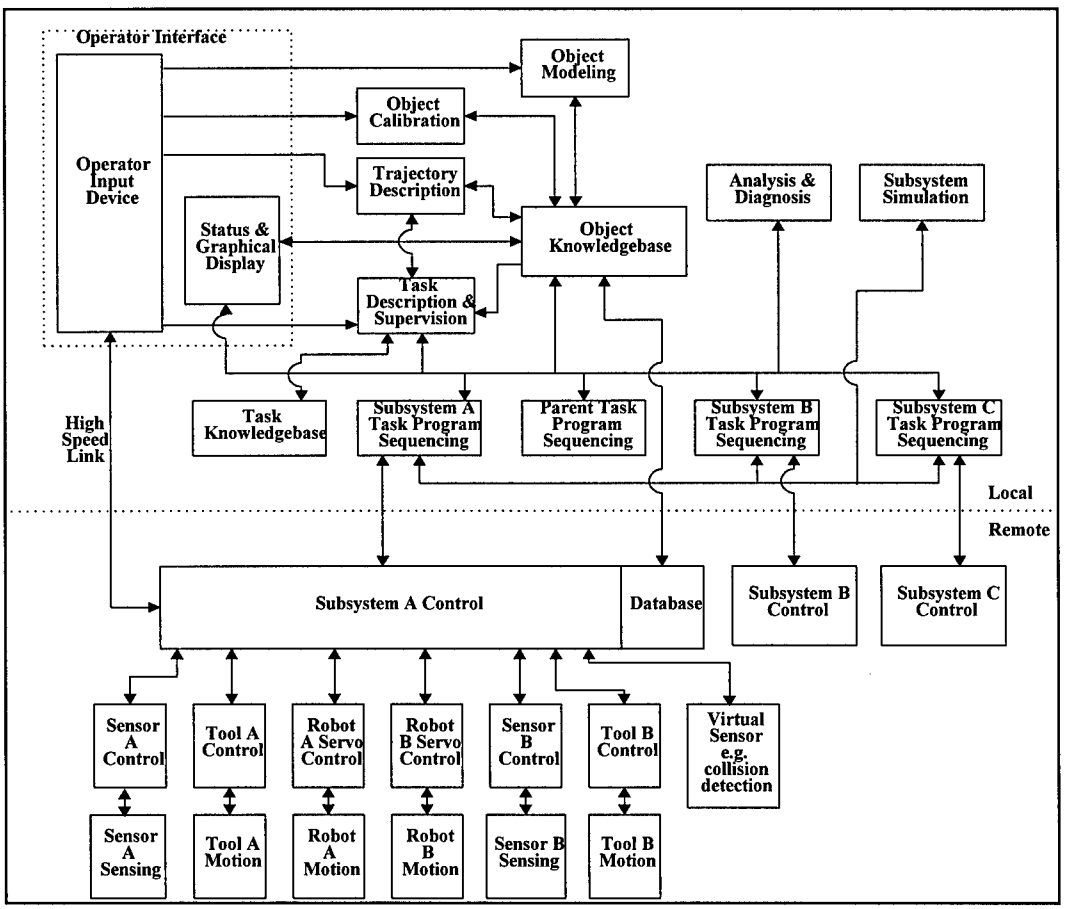


Figure II.1. UTAP Architecture [3]

Figure II.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 direction communication can occur. Communication can only occur between connected modules.

For any particular application, not all of the modules need to exist. Likewise, some modules may have multiple instances. The local and remote controllers each have a configuration file that specifies which modules compose the system. For example, if the Object Modeling module is not needed, then the corresponding configuration file would indicate the absence of that module from the system. Furthermore, not all modules will accommodate every interface message. Therefore, each module maintains a service profile that describes the set of UTAP messages and data posting capabilities supported by the module.

The UTAP specification provides a foundation for a standard telerobotic architecture. However, there are some minor naming inconsistencies in the specification. For example, the UTAP specification gives three different names for the SGD module; "Status Graphics and Displays", "Status and Graphical Display", and "Status Graphics Displays." It is even abbreviated as SDG in some places. The different names can imply different functions for the module. Inconsistencies of this nature also appear in other areas of the specification. For example, twenty different modules are named throughout the document. However, messages are only defined for seventeen modules. Additionally, some areas of the specification require more explanation or clarification. The UTAP messages are an example of this. The meaning and purpose behind each

message should be described. Without defining the meaning of each message, an individual implementing a UTAP-compliant application may not know which message to use to accomplish a particular task. Chapter VI further addresses these issues with the UTAP specification and presents my recommendations.

II.2. Previous AFIT Research

Anchor's research [3] represents the first implementation of UTAP. His research primarily involved redesigning the AFIT Robotics and Automation Applications Group (RAAG) Lab B PUMA manipulator to be compliant with the UTAP architecture. Anchor's thesis shows 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.

Anchor's UTAP-compliant controller implemented portions of the robot servo control, object knowledge, and operator interface modules of the specification (see Figure II.1). The controller performed adequately; although, there was degradation in the performance as evidenced by increased position error during trajectories. Additionally, Anchor pointed out the "spikiness" of the error plots of the UTAP controller when compared to the plots of the non-UTAP controller (see Figure III.3). He indicated that the "spikiness" was due to the lack of controller gain tuning. This is further addressed in the next chapters.

Anchor conducted his performance measurements without tuning the gains of his UTAP-compliant controller. Gain tuning will optimize the performance of the controller and decrease the amount of error. Furthermore, Anchor obtained his measurements by

using the Chimera trajectory generation module to move the robot in a step response and the Chimera track module to record the commanded and actual joint angles. The trajectory generation module computes the trajectory needed to move the robot from the current position to the commanded position. It accomplishes this by dividing the difference between the positions into very small increments and moving the robot through each increment. Error compensation takes place at each increment. Thus, the trajectory generation module does not provide a true step response motion.

The motion command of the trajectory generation module hides some of the performance degradation of the UTAP-compliant controller. To avoid this, the trajectory generation module must be modified to command the robot directly from the initial position to the final position without the incremental movement. This is a true step response and provides more accurate data on the performance of the UTAP-compliant controller.

II.3. Commercial Work Associated with UTAP

Using the UTAP specification as a guide, Advanced Cybernetics Group, Inc. (ACG) has implemented several robotic application building blocks that are compatible with the UTAP philosophy [5]. ACG built their product on the commercially available Adept V+ robotic programming language. In addition to describing the application's implementation, ACG gives examples of commercial and Air Force sites that are using their product to perform tasks with telerobotic systems.

ACG begins to show that the philosophy of the UTAP document is sound and commercially available products can be used to implement the UTAP specification.

ACG's design of their building blocks conforms to the UTAP goals of modularity and reusability. They have also adopted a module naming convention similar to the UTAP message naming convention. However, the ACG work focuses on the sensor control portion of the UTAP specification. The remaining portions of UTAP, such as task level control, programmable IO, etc., are not implemented.

ACG has shown that a commercially-available product can be used to implement modules that are generic enough to be used in several different robotic applications. This confirmed that modularity and reusability are obtainable goals. However, ACG did not closely match their naming conventions and message parameters with the UTAP specification. This, coupled with the fact that ACG focused primarily on the sensor control module while I intended to implement a broader portion of the UTAP specification, prevented me from simply building upon their V+ code as I had originally intended.

II.4. Other Standardization Efforts

To understand the amount of time and effort required to develop a standard with the amount of complexity inherent in a telerobotic architecture, we can compare the UTAP effort to the effort expended developing the Ada programming language. The benefits of a standardized architecture can be seen by looking at the benefits obtained through the Portable Operating System Interface (POSIX) standard.

II.4.1. Ada

In 1974, the Department of Defense (DoD) initiated the common high order language program to define the requirements for a common language for programming

large scale and real-time systems [6]. After extensive reviews by the Services, industrial organizations, universities, and foreign military departments, the final requirements were published in the Steelman specification. This phase in the development of Ada lasted for four years.

Implementation of the requirements began in 1978 and culminated in 1983 with the publication of ANSI/MIL-STD-1815A. This five-year development consisted of three phases. The design team received numerous evaluation reports at the end of the first and second phases. During the third phase, they received nine hundred language issue reports and test and evaluation reports from fifteen different countries. The design team also conducted several intense week-long design reviews attended by dozens of industry experts.

From the above paragraphs we see that the Ada project took nine years and thousands of man-hours to reach the first release. In contrast, UTAP began in 1992 as an engineering study of Air Logistic Center (ALC) remanufacturing processes conducted by NASA's Jet Propulsion Laboratory (JPL) [11] and the UTAP working group consisted of ten individuals. Certainly, UTAP does not have the same publicity that Ada has; and it definitely does not have the same support and funding as Ada. Therefore, we cannot expect to apply the same amount of effort to UTAP. On the other hand, the comparison shows that exceptional progress has already been made on the UTAP specification considering the level of effort expended. Likewise, the comparison shows that it is reasonable to expect that a considerable amount of work remains.

II.4.2. Portable Operating System Interface (POSIX)

The POSIX System Interface Standard is the central open system standard related to the historical UNIX timesharing system and is an application program interface standard for basic operating system functions [10]. The POSIX working group had the following four main goals when developing the standard:

1. Promote application source code portability
2. Specify an interface, rather than an implementation
3. Consider all major historical implementations
4. Keep the interface minimal

By achieving these goals, the working group has ensured that applications developed for POSIX-compliant systems can be ported to new POSIX-compliant systems with minimal effort. This saves time and money by providing users with the capability to upgrade and expand their systems. By specifying the interface but not the implementation, the working group ensured that users can take advantage of new technology. In other words, as long as the interface remains the same, an old implementation can be replaced by a better one.

When examining the goals of the UTAP specification, we find a direct correspondence to the goals of the POSIX working group. It is reasonable to expect UTAP will benefit from the same advantages as those experienced by POSIX compliant systems, provided the UTAP goals are obtained.

II.5. Summary

A standard interface environment for telerobotic applications will reduce the cost of telerobotic systems by increasing module reuse and portability and by decreasing module development time and costs. The UTAP specification defines a standard that

attempts to meet these goals. Anchor's research showed the feasibility of implementing the UTAP specification on one particular platform. However, there are still performance issues to be investigated and UTAP must be implemented and tested on additional platforms. ACG's work proves that robotic applications can be developed by re-using generic modules written in the V+ language. Finally, by comparing the UTAP effort to other standardization efforts, we see that the process of developing and implementing a large standard of this nature is difficult and time consuming but the rewards are worth the effort.

III. Performance Measurement of Anchor's UTAP Implementation

This chapter describes the methodology behind the performance measurement of Anchor's UTAP implementation. It presents the steps followed to accomplish the measurement and justifies the choices made while conducting the measurement. It then provides an analysis of the data obtained the from the measurements.

III.1. Methodology

This section is divided into three parts. The first part contains preliminary information about Chimera and Anchor's UTAP controller that is needed to understand the methods. The second part contains the steps taken to prepare for the performance measurement. Finally, the third part explains the actual performance measurement.

III.1.1. Preliminary Information

Anchor implemented his UTAP-compliant controller under the Chimera Real-Time Operating System (OS). Chimera provides the interface between the operator and the manipulator hardware. It converts the program commands (movement commands etc.) to digital signals and sends them to the manipulator hardware. Chimera supports application development in the C and C++ programming languages. It is the Chimera OS that provides the functions to use and program the manipulator.

Throughout his research, Anchor used several predefined manipulator positions. To maintain continuity with his performance tests, I used the same positions. Table III.1 lists the relevant positions. The values are in radians and are measured from the baseframe of the respective joint.

In many of the steps described below, the term "spawned" is used. Under Chimera, spawning a module loads the module into memory, but does not begin execution of the module. The "on" command begins the execution of the module, or activates it. It is through the "on" command that operator-input parameters, if required, are passed to the module.

Position Name	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
Home	0.0	-1.5708	1.5708	0.0	0.0	0.0
Data Initial	0.0	-2.36	2.36	0.0	0.0	0.0
Data Final	1.5708	-1.5708	0.524	0.0	0.0	0.0

Table III.1. Predefined Positions (radians)

Several Chimera modules are also mentioned in this chapter. What follows is a brief description of their purpose and the results of their execution.

The Chimera **puma_pidg** module is the proportional integral derivative (PID) controller module used on the AFIT PUMA 560 manipulator under the Chimera operating system. Anchor modeled his UTAP-compliant controller after the **puma_pidg** module and the Chimera gravity compensation (**grav_comp**) module.

The Chimera **trjjgen** module is the trajectory generator and accepts as input the desired joint angles (measured in radians from the baseframe of each joint) and a movement duration (measured in seconds). From these values, the module computes a trajectory (as described in section II.2.) that will move the manipulator to the desired joint angles.

The Chimera **track** module records the manipulator's commanded and actual joint angles during movement. When the track module is activated, it records the data at a rate of fifty samples per second. The data is written to a file specified at activation.

Anchor's **int_RSC** module was the interface module between his UTAP-compliant controller (the **UTAP_RSC** module) and the Chimera operating system functions. Figure III.1 shows the relationship between Anchor's interface and UTAP modules and the Chimera OS modules. In Anchor's modules, RSC stands for Robot Servo Control.

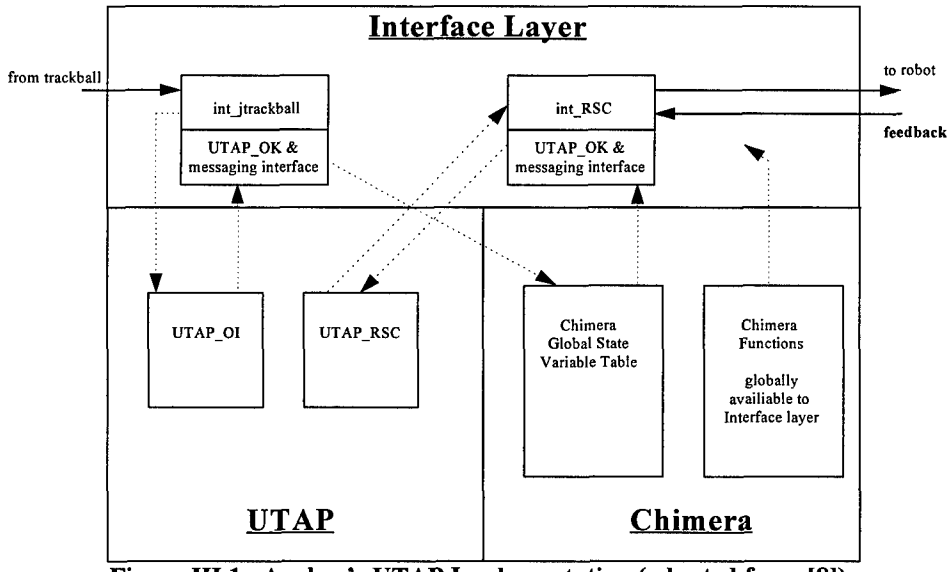


Figure III.1. Anchor's UTAP Implementation (adapted from [8])

To start the UTAP-compliant controller, the user spawns and activates the **int_RSC** module. The **int_RSC** module accesses the **UTAP_RSC** functions that include the PID controller and gravity compensation functionality.

III.1.2. Preparatory Steps

Before I could begin the performance measurements, I needed to tune the gains of Anchor's controller and establish a performance baseline. These steps are described and justified in the next two sections.

III.1.2.1. Gain Tuning

To accurately measure the performance limits of a system, the system must be operating at its optimal level. In section IV.3 of [3], Anchor suggested that the performance of his UTAP-compliant system could be improved by controller gain tuning. Therefore, his system required gain tuning before the performance could be measured.

In [7], Klafter describes the basic steps required to gain tune a PID controller as follows:

1. With $K_i = K_d = 0$, adjust K_p until the system step response is either critically (or slightly under-) damped.
2. For the value of K_p just found, increase K_i until the steady-state error is either zero or has reached an "acceptable" value. (Normally, $K_i > K_p$.)
3. Increase K_d until the system step response is again either critically or slightly underdamped.

Each joint of the manipulator has its own values for K_p, K_i, and K_d and must be tuned separately. Under Anchor's UTAP implementation, all three values for each of the six joints are stored in the configuration file named *int_rsc.rmod.*

III.1.2.1.1. Joint One Gain Tuning

Joint one was gain tuned using Klafter's procedure without deviation. I determined when the joint was critically damped, slightly under-damped, or had an acceptable steady-state error by plotting the joint position data at each iteration of the procedure. The Chimera **track** module was used to record the data while the joint moved through a 90.0 degree arc.

III.1.2.1.2. Results from Gain Tuning Joint One

During the tuning of joint one, I quickly learned that it was going to be a very time consuming process. Each adjustment of one of the values meant that each module

must be unloaded, the value changed, the modules re-spawned, the movement started, the data extracted from the **trjjgen** output, the data plotted, and the next change determined. Occasionally, the system would give an error message stating that it was not calibrated. When this occurred, I had to kill all the active modules, calibrate the manipulator, and re-load all the modules. Since this was a time consuming process, I decided to combine steps and save time.

III.1.2.1.3. Joint Two Gain Tuning

For joint two, I began at the home position with the original gains. A -90.0 degree movement was commanded and the position data from the track module was plotted. Then, based on the results from joint one, the movement was commanded with an increased K_p and again with a decreased K_p, leaving K_i and K_d at the original values. By analyzing the resulting plots of these three movements, it could be determined that joint two was already critically damped with the original K_p value.

Next, the K_i value was slightly increased and the K_d value was slightly decreased from the original values. This decision was also based on the results of tuning joint one. After commanding the -90.0 degree movement, the results were analyzed. The trajectory was slightly improved. I continued this process until I was satisfied with the joint two trajectory. The modified procedure saved two days.

III.1.2.1.4. Joint Three Through Six Gain Tuning

The joint two procedure was used on joints three through six with similar time savings. Table III.2 shows the gain values before and after tuning. Appendix D contains the trajectory plots for all six joints.

Joint	K_p		K_i		K_d	
	Before	After	Before	After	Before	After
1	4000	4000	5	20	80	20
2	11000	11000	5	20	114	10
3	3000	3000	5	20	25	25
4	500	500	5	5	7	7
5	310	310	5	5	7	7
6	300	300	5	5	17	17

Table III.2. Gain Values

III.1.2.2. Obtaining a Baseline

Besides tuning the controller gains, another step was required. Before the performance of Anchor's UTAP implementation could be measured, a performance baseline of the original system had to be obtained. With this baseline, a comparison could be made between the original system and Anchor's UTAP-compliant system. For the comparison to be valid, the baseline had to be obtained by executing the Chimera functions that matched the functions implemented in the UTAP controller. Anchor modeled his UTAP system after the Chimera **puma_pidg** and **grav_comp** modules. Therefore, the baseline was obtained using those Chimera modules.

The steps used to obtain the baseline were the same steps Anchor used to obtain his performance measurements. Additionally, data was collected from the same three movement durations used by Anchor. The fast movement lasted 1.5 seconds, the nominal movement lasted 3 seconds, and the slow movement lasted 5 seconds. The measurements were taken during the movement of the manipulator from the data initial position to the data final position. These positions were the same as those used by Anchor and are shown in Table III.1. For each of the three motion durations, I activated the **puma_pidg**,

grav_comp, and **trjjgen** modules, giving the **trjjgen** module the data initial position. When the manipulator reached the data initial position, the **track** module was activated. Then, the **trjjgen** module was activated with the data final position and the appropriate duration. Upon completion of the movement, the data file created by the **trjjgen** module contained the commanded and actual joint position value samples. Since this was accomplished for each of the motion durations, three files were produced containing data. With this data, the error between commanded and actual joint positions could be calculated. This error data constituted the baseline performance.

III.1.3. Measurement of the Performance

This section of the chapter is divided into three parts. This first part discusses the pseudo-step response performance measurements. These are the same type of measurements as conducted by Anchor. The second part describes my attempt at a modified step response performance measurement. The step response issue was first addressed in section II.2 and is further addressed below. The final part discusses missed cycle testing.

III.1.3.1. Pseudo-Step Response

As discussed in section II.2, the **trjjgen** module does not provide a true step response motion. However, the measurements taken by Anchor did give a good general idea of the performance of his UTAP controller. Also, I wanted to see how much the gain tuning affected the performance. Therefore, I conducted the performance measurement of the gain tuned system using Anchor's method (which is also the same

method used to obtain the baseline). Section III.2.2 contains an analysis of the data received during these measurements and Appendix E contains the position error plots.

III.1.3.2. Modified Step Response

The first task under this portion of the research was to modify the Chimera **trjjgen** module to perform a true step response motion. The modification involved removing the loop in the **trjjgenCycle** function that divided the difference between the current position and the desired position into small increments and moved the joint through the increments. Instead, the function now moves the joint directly from the current position to the desired position. Appendix F contains the **trjjgenCycle** function source code both before and after modification.

The next task involved using the modified trjjgen module to collect the step response data. I attempted to follow the same data collecting procedure as I had used with the pseudo-step response function. However, any attempt to command movement resulted in severe oscillations of joint four, five, or six. Once the oscillations started, the emergency kill switch had to be pressed to stop them. Therefore, use of the modified step response for performance testing was not feasible. Section III.2.3 discusses these oscillations and their probable cause.

III.1.3.3. Missed Cycle Testing

I had originally planned to conduct a third performance test on Anchor's controller. Under this test, the execution frequency of the original controller would have been slowed to the point were it almost missed cycles but did not. I would have then attempted to execute the UTAP controller at the same frequency. If execution was

possible and the controller was stable, the number of missed cycles would have been counted and evaluated. However, contrary to Anchor's belief, the UTAP controller was already missing cycles at the current operating frequency. The missed cycles, coupled with the oscillations, prevented the conduction of this test.

III.2. Analysis

This section is divided into three parts corresponding to the gain tuning, pseudo-step response testing, and step response testing of Anchor's controller. In each part, the data is presented and explained.

III.2.1. Gain Tuning

Figure III.2 shows the final trajectory plot obtained while tuning joint one. The graph shows the joint one trajectories for before and after gain tuning. As can be seen, there is a significant improvement in the post-gain tuned trajectory over the pre-gain tuned trajectory. The post-gain tuning trajectory reaches steady state much sooner and has less over-shoot.

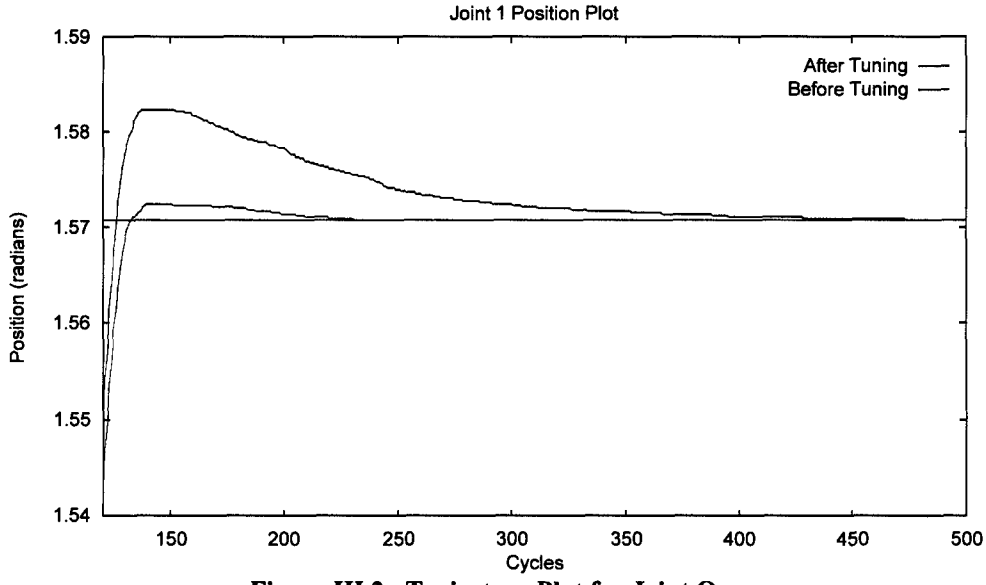


Figure III.2. Trajectory Plot for Joint One

Joints two and three showed similar improvements. As shown in Table III.2, the gains for joints four, five, and six did not require changing. Therefore, the position plots show only a single trajectory. Appendix D contains the trajectory plots for all six joints.

III.2.2. Pseudo-Step Response

From the data obtained by the baseline measurements, I calculated the error between the commanded and actual positions of each joint for each trajectory on the non-UTAP controller. The data obtained during the pseudo-step response measurement was then used to calculate the position error of each joint for each trajectory on the gain-tuned UTAP controller. These error values, along with the values obtain by Anchor for the non-gain tuned UTAP controller, were then plotted. Each plot shows the original, non-gain tuned UTAP, and gain tuned UTAP controller's error for a particular joint at one of the three trajectory durations. The error plot for the nominal trajectory of joint one is shown in Figure III.3 and Appendix E contains all the error plots.

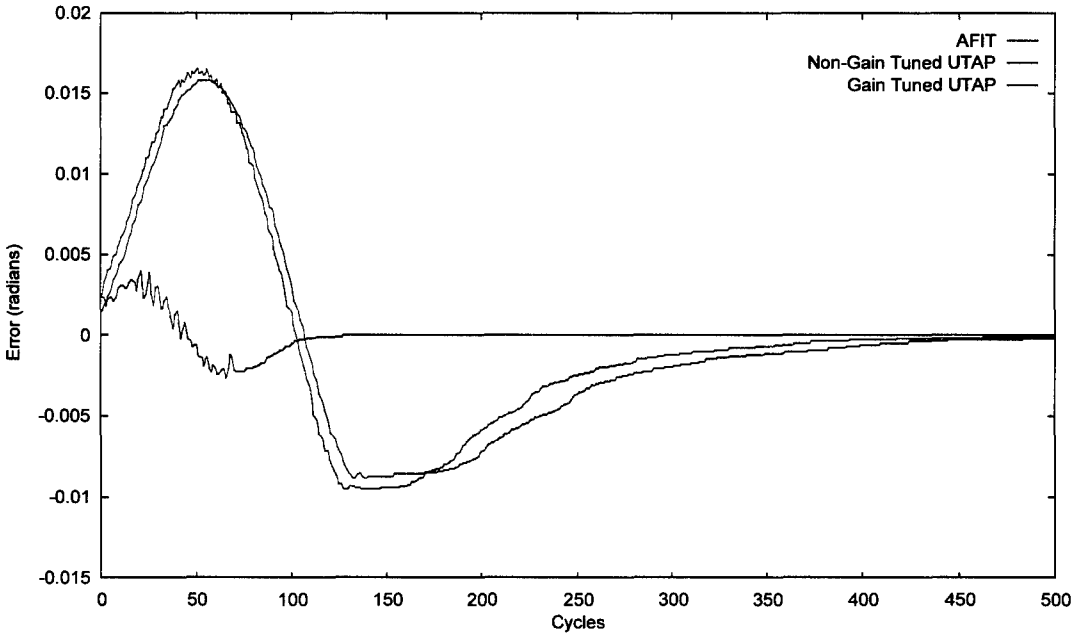


Figure III.3. Joint 1 Error for Nominal Trajectory

The error values are in radians and are plotted against the cycle of the periodic task that recorded the data. The cycle can be equated to time since the task recorded the data once each cycle and it was executed at a frequency of 50 Hz.

Tables III.3, III.4, and III.5 show the integral error exhibited by each controller. I calculated these values by summing the absolute value of the error measurement at each cycle. The percent difference between the original controller and the pre-gain tuned controller was calculated using the following equation:

$$PercentDifference = 100 \times \left(\frac{PreGainTunedError - OriginalError}{PreGainTunedError} \right)$$

The percent difference between the original controller and the post-gain tuned controller was calculated using the following equation:

$$PercentDifference = 100 \times \left(\frac{PostGainTunedError - OriginalError}{PostGainTunedError} \right)$$

Likewise, the percent difference between the pre-gain tuned controller and the post-gain tuned controller was calculated with this equation:

$$PercentDifference = 100 \times \left(\frac{PostGainTunedError - PreGainTunedError}{PostGainTunedError} \right)$$

	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6	Total
Original Error:	1.9273	0.8392	1.1085	0	0	0	3.8751
Pre-gain Tuning Error:	1.7497	0.9547	0.8569	0.5317	0.1558	95.228	99.4768
Post-gain Tuning Error:	0.1347	0.1208	0.1442	0.1124	0.0658	0.2828	0.8607
Original vs Pre-gain Tuning Percent Difference	-10.15	12.09	-29.37	100.00	100.00	100.00	96.10
Original vs Post-gain Tuning Percent Difference	-1,330.35	-594.80	-668.84	100.00	100.00	100.00	-350.22
Pre-gain Tuning vs Post-gain Tuning Percent Difference	-1,198.53	-690.38	-494.31	-373.16	-136.86	-33,566.96	-11,457.39

Table III.3. Integral Error for Slow Trajectory

	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6	Total
Original Error:	2.3087	0.9821	1.2041	0	0	0	4.4948
Pre-gain Tuning Error:	2.2042	1.1034	1.0679	0.8692	0.2423	104.7391	110.2261
Post-gain Tuning Error:	0.2007	0.1221	0.2247	0.1673	0.1010	0.4785	1.2943
Original vs Pre-gain Tuning Percent Difference	-4.74	11.00	-12.75	100.00	100.00	100.00	95.92
Original vs Post-gain Tuning Percent Difference	-1,050.42	-704.57	-435.78	100.00	100.00	100.00	-247.26
Pre-gain Tuning vs Post-gain Tuning Percent Difference	-998.35	-803.99	-375.19	-419.45	-139.91	-21,786.81	-8,415.96

Table III.4. Integral Error for Nominal Trajectory

	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6	Total
Original Error:	2.9561	1.1813	1.5658	0	0	0	5.7032
Pre-gain Tuning Error:	2.8886	1.3002	1.5148	1.6659	0.4419	1.3951	9.2065
Post-gain Tuning Error:	0.2666	0.1379	0.3385	0.3019	0.1756	0.9034	2.1239
Original vs Pre-gain Tuning Percent Difference	-2.34	9.15	-3.37	100.00	100.00	100.00	38.05
Original vs Post-gain Tuning Percent Difference	-1,008.98	-756.42	-362.54	100.00	100.00	100.00	-168.53
Pre-gain Tuning vs Post-gain Tuning Percent Difference	-983.66	-842.65	-347.47	-451.81	-151.68	-54.43	-333.48

Table III.5. Integral Error for Fast Trajectory

The plots and the tables show that, in all cases, by gain tuning the UTAP controller, the error was reduced from that exhibited by the non-gain tuned UTAP controller. In fact, the error values for all trajectories of joints one, two, and three of the post-gain tuned controller are less than the values for the original controller. Additionally, the total error values for all trajectories of the post-gain tuned controller are less than the values for the original controller. If only the total error and percent difference numbers are considered, it would appear that implementing a gain-tuned UTAP controller enhanced the system performance. However, there is no evidence to suggest that the original controller was gain-tuned for optimal performance. Normal

wear of the system would require that the gains be periodically tuned. I believe that the original controller requires gain tuning. If gain-tuning of the original controller was accomplished, I would expect it to perform better than the gain-tuned UTAP controller. the gain-tuned original controller versus the post-gain tuned UTAP controller error values would be similar to the original versus pre-gain tuned values.

III.2.3. Step-Response

As stated in Section III.1.3.2, the step response evoked severe oscillations in joints four, five, or six. Movement of joints one, two, or three usually caused joint four to oscillate. Movement of joints four, five, or six would cause that particular joint to oscillate. The oscillations occurred when the magnitude of a commanded joint movement exceeded a certain threshold. The threshold varied for each joint. The larger joints, joints one, two, and three, could sometimes be moved up to five degrees without oscillations. Joints four, five, and six could rarely be moved more than one degree.

While investigating the oscillations, I found that they could be induced with both the original and the modified trajectory generation modules under the UTAP-compliant controller. However, they could not be induced with either module under the original controller. Eventually, the problem was isolated to the controller gains.

When the K_p and K_d values were reduced and the K_i value was set to zero for joints four, five, and six, the oscillations ceased to occur. This indicated a problem in the gain tuning procedures. When conducting both Klafter's and the modified gain tuning procedures, the **trjjgen** module was used to move the joint. In both cases the joint movement lasted 1.5 seconds. The movements were not large enough or fast enough to

induce the oscillations. Additionally, for the same reasons a step response was needed during the performance measurements, a step response movement should have been used when gain tuning the controller. Therefore, a better method for gain tuning the controller must be developed.

When experimenting with the joint four gains, I found that the K_p and K_d gains could be set to 125 and 3.5 respectively and the K_i gain could be set to 1.5 without the oscillations occurring. When K_i was set above 1.5 with these K_p and K_d values, evidence of the oscillations began. Therefore, implementing UTAP limited the range of gain values that could be used without causing oscillations. By limiting the gain values, it is possible that the joint could not be tuned to optimal performance.

III.3. Summary

This chapter has presented the methods used to obtain the performance measurements of Anchor's UTAP implementation. It described gain tuning the controller and obtaining a measurement baseline. This was followed by a description of the measurement of the two types of step response. Some other tests were discussed that proved to be not feasible.

The chapter then moved on to analyze the data. It indicated that performance was greatly enhanced by gain tuning the controller. The performance was enhanced so much that it appeared to perform better than the original controller. This suggested that the original controller would also benefit from gain tuning. Gain tuning the original controller would likely return the performance ratio between the original controller and the UTAP controller back to that observed by Anchor.

The step response testing performed on the UTAP controller identified a problem with the UTAP controller gains. The procedures used to gain tune the controller did not take into consideration the method used to move the manipulator, the size of the movement, or the speed of the movement.

IV. Implementation of UTAP on the Adept 550 Manipulator

This chapter presents the methods used to implement the UTAP specification on the Adept 550 manipulator. It describes the methods involved with the preparation of the application, development of the interface layer, testing, and performance measurement. Next, the chapter presents an analysis of the performance data.

IV.1. Methodology

This section of the chapter describes the implementation of the UTAP specification on the Adept 550 manipulator. The overall approach to the task was to find an existing application written for the Adept system, convert it to UTAP compliance, and develop an interface layer that would allow the UTAP-compliant application to run on the Adept system. These steps are described below and are followed by a description of the methods involved with the performance measurement of the implementation.

IV.1.1. Determination of the Appropriate Application

There were three major criteria to consider in choosing the application. First, the application must incorporate a variety of functions such as robot control, use of the manual control pendant (MCP), operator input, and disk file IO. This ensured that a respectable portion of the UTAP specification would be implemented. Second, the application must not be too large and complex or there would not have been time to finish the project. Third, AFIT must have legal rights to use and alter the application.

Previously, ACG had provided AFIT with numerous applications that demonstrated various capabilities of the Adept system. After reviewing these applications, I found that they were either too complex (several thousand lines of code) or

too specialized (only involved robot control and output to the screen). I also reviewed two projects from the Robotic Fundamentals course. These were not complex enough for this purpose.

The application eventually chosen was one I had written during a V+ programming course given by Adept Technologies. The primary function of the application is to move parts from one pallet to another. Additional functions include MCP control of the application, screen display of location information, screen display of system information, and writing location information to an audit trail file. I also added two force control functions to the application. The first simulates attempting to insert a part into a recess. If the part will not slide into the recess smoothly, it is returned to the pallet and another part is tried. The second force control function slowly lowers a part to the work surface. It uses force sensing to determine when the part is sitting on the work surface. It then releases the part and moves to a safe position. Appendix G contains the original source code for the application.

IV.1.2. Conversion of the Application to UTAP Compliance

In order to begin the conversion, I needed to decide how to implement the UTAP messages. Anchor had to choose between using Chimera's Interprocessor Message Passing or implementing them as subroutine calls [3]. The choice turned out to be very easy since V+ does not support any type of interprocess messages. Therefore, the only option was to implement the UTAP messages as V+ subroutine calls. A small problem did arise from this. V+ allows a subroutine to have up to 256 characters in its name.

However, V+ only recognizes the first 15. Therefore, UTAP messages with long names could be interpreted as the same message. For example, the UTAP messages US_AXIS_ SERVO_USE_ABS_POSITION_MODE and US_AXIS_SERVO_USE_REL_ POSITION_MODE would be interpreted by V+ as the same message, US_AXIS_ SERVO_U. During a telephone conversation, John Michaloski of the Intelligent Systems division of NIST and I decided that the best alternate would be to abbreviate the message names. Therefore, "US_" was dropped from the front of all the messages implemented. Most messages also have a three letter abbreviation of the module name following the "US_." This was standardized to a two letter abbreviation as shown in table IV.1. Additionally, other words were abbreviated as needed. For example, the UTAP message US_TLC_START_FINE_MOTION became tl_str_fine_mot. Consistency was maintained in all the abbreviations made. Appendix C contains a list of all the messages defined in the UTAP specification.

UTAP MODULE NAME	UTAP ABBREVIATION	V+ ABBREVIATION
Analysis & Diagnostics	ADS	AD
Object Calibration	OC	OC
Operator Input Devices	OI	OI
Object Knowledge	OK	OK
Object Modeling	OM	OM
Parent Task Program Sequencer	PTPS	PS
Robot/Axis Servo Control	RSC (or Axis Servo)	AS
Sensor Control	SC (or Sensor)	SC
Status Graphics & Display	SGD	SG
Subsystem Simulators	SS	SS
Tool Control	TC	TC
Trajectory Description	TD	TR
Task Description & Supervision	TDS	TD
Task Knowledge	TK	TK
Task Program Sequencer	TPS	TS
Subsystem Task Level Control	TLC	TL
Virtual Sensor	VS	VS

Table IV.1. UTAP Module Name Abbreviations

Since the focus of the UTAP specification is on the interfaces, conversion of the application to UTAP compliance was approached with the concept that anytime the application communicated with the operating system, the operator, or between processes, the communication must be in the form of a UTAP message. The major difficulty was determining which UTAP message should be associated with each V+ command or system call used in the application. This difficulty arose from the fact that the UTAP specification does not provide the meaning or usage of the messages it imposes. Table IV.2 shows the UTAP messages that were implemented.

UTAP MESSAGE	V+ FUNCTION NAME	V+ COMMAND
AXIS_SERVO		
US_AXIS_SERVO_SET_POSITION	as_set_position	MOVE
US_AXIS_SERVO_SET_VELOCITY	as_set_velocity	SPEED
GENERIC		
US_HOLD	hold	WAIT
US_INIT_OK	init_ok	GLOBAL
US_GET_EXT_DATA_VALUE	get_ext_data	PROMPT
US_POST_EXT_DATA_VALUE	post_ext_data	TYPE
OBJECT KNOWLEDGE		
US_OK_ATTRIBUTE_QUERY	ok_attrib_query	SPEED, SWITCH, CONFIG,...
PROGRAMMABLE_IO		
US_PIO_BIT_READ	pi_bit_read	SIG, PENDANT
US_PIO_BIT_SET	pi_bit_set	KEYMODE
US_PIO_DISABLE	pi_disable	FCLOSE, DETACH
US_PIO_ENABLE	pi_enable	ATTACH
US_PIO_SET_MODE	pi_set_mode	FOPENA, FOPENR, FOPENW
PARENT TASK PROGRAM SEQUENCER		
US_PTPS_SELECT_AGENT	ps_select_agent	EXECUTE
SENSOR		
US_SENSOR_GET_ATTRIBUTES_READING	sc_get_att_read	FORCE.READ, FORCE.OFFSET
US_SENSOR_GET_READING	sc_get_reading	LATCH, LATCHED, FORCE.READ
TASK DESCRIPTION		
US_TDS_EXECUTE_PROGRAM	td_exec_prog	CALL
TASK LEVEL CONTROL		
US_TLC_START_FINE_MOTION	tl_str_fine_mot	FINE

Table IV.2. UTAP Messages Implemented

The table gives the UTAP message name, the V+ function name, and the V+ command to which the message equates. Additionally, there were several functions that did not have corresponding UTAP messages. In these cases, messages were created to fill the needed function. Table IV.3 lists these messages. My main concern while converting the application code to UTAP compliance was to maintain the exact functionality of the original application. This would aid the comparison between the performance measurements of the original system and the UTAP-compliant system. More importantly, this would aid in proving the feasibility of UTAP. If an operational system, such as the F-15 paint booth, were being converted to UTAP compliance, we certainly could not change the functionality of the system. Appendix H contains the source code for the UTAP-compliant application.

UTAP MESSAGE	V+ FUNCTION NAME	V+ COMMAND
GENERIC		
US_GET_EXT_LOCATION_DATA	get_ext_loc_dat	SET
US_SET_EXT_LOCATION_DATA	set_ext_loc_dat	SET
SENSOR		
US_FT_SENSOR_DISABLE	ft_sc_disable	FORCE.MODE
US_FT_SENSOR_ENABLE	ft_sc_enable	FORCE.MODE
US_FT_SENSOR_MODE_SELECT	ft_sc_mode_sel	FORCE.MODE
TOOL CONTROL		
US_GRIPPER_CLOSE	gripper_close	SIGNAL 1
US_GRIPPER_OPEN	gripper_open	SIGNAL -1

Table IV.3. Additional Messages Created

IV.1.3. Development of the Interface Layer

Once the application had been re-written to be UTAP-compliant, a program layer needed to be written that accepted UTAP messages and translated them into V+

commands. The interface layer consists of a set of V+ program files and each program file corresponds to a UTAP module. For example, the program file pi.pg contains the V+ code that implements the functionality of the UTAP programmable input/output module. The following sections discuss general issues common to all the modules and the specific issues related to each module. Throughout this chapter I will refer to message names. When doing so, the V+ function name will be used, not the UTAP message name. This will make it easier for the reader to locate the message within the interface layer V+ programming code (Appendix I).

IV.1.3.1. General Programming Issues

While developing the interface layer, I incorporated programming techniques that ensured the easy expansion of the layer in the event that additional UTAP messages need to be implemented. For example, one of the parameters of the GET_EXT_DATA message is "channel." For the application selected, the keyboard is the only channel that inputs data via the GET_EXT_DATA message. However, within the code for the GET_EXT_DATA message, I used a case statement based on the channel to the select the operations that need to be completed. The keyboard section is the only one implemented, but adding the MCP, file, or other channel would be much easier under this design.

Several new messages were added to those defined by the UTAP specification. When this was done, the naming conventions used throughout the UTAP specification were adhered to.

IV.1.3.2. Axis Servo Module

Two UTAP messages were implemented from the axis servo module message list. Both messages were direct mappings to V+ commands. Of special interest was the parameter to the AS_SET_POSITION message. The UTAP specification defines the parameter to this message as a pointer to a double precision number. Instead, it was implemented as a V+ transformation location variable indicating a desired manipulator position. This was an example of the UTAP message name and parameters not matching the desired use of the message. However, a more suitable message could not be found, so this one was chosen. See Appendix B for a definition of the different V+ location variable types.

IV.1.3.3. Generic Module

This module implements a portion of the generic messages defined by the UTAP specification. The get_ext_data message was used to get operator input from the keyboard and the post_ext_data message was used to send output to the screen, MCP, or file. It is possible that the Operator Interface module should contain messages that serve the input and output functions implemented here. However, the UTAP specification did not provide such messages in the OI module, but instead placed these messages in the UTAP Data Definitions module. Two messages were added to this module that were not defined by the UTAP specification: get_ext_loc_dat and set_ext_loc_dat. These messages are used to store and retrieve V+ transformation location variables. The UTAP specification does not have messages of this type with the appropriate parameters. Therefore, I was forced to create the new messages.

IV.1.3.4. Object Knowledge Module

The purpose of the Object Knowledge module is to store and maintain specific knowledge of the various objects forming the entire system. This includes manipulator information such as model number as well as information about the part being worked on such as shape, mass, or hardness. For the purposes of this research, only the portions of the Object Knowledge module needed for the application were implemented. Therefore, the messages implemented in this module are the ones used to access system information.

IV.1.3.5. Programmable Input/Output Module

The messages in this module were used to enable, disable, and set the mode of the disk access and MCP. Also, they are used to read data from and write data to the MCP, signals, and system timers.

IV.1.3.6. Parent Task Program Sequencer Module

A typical robotic application will have several different processes controlling different aspects of the overall task. The palletizing application chosen uses separate processes to control the manipulator, manage the MCP, display the coordinate information, and display system information. The Parent Task Program Sequencer module manages the processes' execution sequence. Since it was the appropriate location for this type of message, the ps_select_agent message was used to start the execution of additional processes.

IV.1.3.7. Sensor Module

The UTAP specification is lacking in force/torque sensor messages. Therefore, I was required to create three new messages in this module. The messages are used to

enable, disable, and select the mode of a force/torque sensor. Existing UTAP messages were used to read the values from the sensor.

IV.1.3.8. Tool Control Module

The UTAP specification focuses primarily on aircraft painting and stripping operations. Thus, tool control messages are only defined for spindle and flow type tools, such as sanders and paint guns. For this reason, I was required to define two new messages for gripper operation: gripper_open and gripper_close. These messages are simple mappings to V+ gripper commands.

IV.1.3.9. Task Description Module

The Task Description module maintains information about what the task needs to accomplish. As a part of that, the module determines which subroutines should be used and when. The td_exec_prog message was part of this module and mapped directly to the V+ "call" command for initiating subroutines.

IV.1.3.10. Task Level Control Module

The Task Level Control module manages the events within each process. It controls positioning modes, motion types, feed rates, etc. The tl_str_fine_mot message was a direct mapping to the V+ FINE command.

IV.1.4. Testing

Testing occurred throughout the development of the interface layer. Each UTAP message was individually tested to ensure appropriate functionality. As stated earlier, an important part of my effort was to make sure that the UTAP-compliant version of the application produced the exact same results as the original version. With this in mind, I

wrote test drivers for each implemented message. The test driver would execute a V+ command or set of commands and then execute the UTAP message corresponding to the commands. I would then compare the output of each to ensure they matched. I used input values such that all categories of outcome were experienced.

Once all the messages had been tested separately, I began combining them to ensure they functioned together as expected. Again, I used test drivers to test the range of input and output possibilities.

After the individual tests were complete, I loaded the application and the interface layer and executed the application. I tested each function of the application to ensure it performed as expected and the output matched that of the original application.

IV.1.5. Performance Measurement

Performance of the implementation was measured in four different ways. First, the number of source lines of code (SLOC) in the original and the UTAP-compliant versions of the application were compared. V+ is an interpretive language. This means that a program's source code is not compiled into some form of machine language. Instead, each line of the program is parsed as execution occurs. The V+ system pre-processes one line ahead of the current execution. Source code optimization could not be accomplished by the system due to the interpretive nature of the language. Therefore, the SLOC number can provide meaningful insight into the complexity of a program.

The second measurement conducted was a calculation of the architectural design complexity of the source code using Card and Agresti's method [4] and provided a measure of the software quality. The method calculates the complexity of a design by

computing and summing two values. The first value defines the structural complexity of the code. It is derived from the relationships between the modules of a system. The relationships are quantified by squaring the number of modules called by each module (called the fanout or f_i) and dividing the sum of the squares by the number of modules (n) to normalize the value. Thus, the formula for structural complexity is: $S = \frac{\sum f_i}{n}$. The second value is the local complexity. This value indicates the amount of complexity within each module and is measured based on the amount of work performed within the module. The workload is calculated based on the number of input and output variables associated with the module (v_i) and the amount of work passed to other modules. The amount of work passed is quantitized by the same fanout value (f_i) used in the calculation of S. The whole value is normalized over the number of modules (n). Thus, the formula for local complexity is: $L = \frac{\sum \frac{v_i}{f_i + 1}}{n}$. These two values are added to give the total complexity of the code: $C = S + L$. The important thing to remember is that complexity measurement is simply one way to gage the quality of a software package. High complexity within a software package indicates that the software is susceptible to more design errors and will be more difficult to maintain.

The third measure of performance I gathered was the amount of system random access memory (RAM) needed by the original and UTAP-compliant versions of the application. Robotic systems have a finite amount of RAM. If UTAP compliance

required too much RAM, then the feasibility of the specification (on Adept systems) would be in question.

Finally, the fourth and most important measurement involved timing the two versions of the application. V+ has built-in timers that can be used in programs. One of these timers was used to measure the execution time of each of the two versions of the application. To ensure that operator response time was not involved in the times, the timer was initialized just after the last operator input was entered. The timer's value was then displayed when execution finished.

Since the execution time varied depending on the starting pallet, I timed both versions with both pallets as starting points. Both versions were timed at 25, 50, 75, and 100 percent of maximum speed. Five measurements were collected for each scenario.

The amount of time a program takes is a much more concrete figure than the number of source lines of code or complexity values involved with a program. This figure gives a vivid view of the amount of overhead added to the application by UTAP compliance.

IV.2. Analysis

This section presents the data obtained while measuring the performance of the Adept 550 UTAP Implementation. The number of source lines of code is the first data presented. Next, I address the Card and Agresti complexity metrics. Third, the amount of random access memory used is presented. Finally, the execution time data is discussed.

IV.2.1. Source Lines of Code

There were 434 total source lines of code for the non-UTAP version of the application. The UTAP version of the application had 769 lines of code. Although the number of code lines almost doubled, we must consider that the number for the UTAP version includes 303 lines of code that comprise the UTAP interface layer. Any new operating system that is developed to be UTAP-compliant will not require an interface layer. Therefore, if the interface layer is considered as part of the operating system, then the UTAP version only had 466 source lines of code. This is only 32 lines, or 7 percent, more that the original version.

IV.2.2. Complexity Measures

Following the method for computing the complexity measures as described in Chapter III, we fine that the original application had a structural complexity of 5.78 and a local complexity of 1.65. This equals a total design complexity of 7.43. When applied to the interface layer and the UTAP application combined, the method yields a structural complexity of 10.88, a local complexity of 1.49, and a total design complexity of 12.37. These values indicate that the UTAP version has a much more complex design than the original version. Once again, we can consider the interface layer part of the operating system. When this is done, the UTAP application has a structural complexity of 5.78, a local complexity of 2.00, and a total design complexity of 7.78. Comparing these values to those of the original application, we see that the structural complexity is the same for both. The additional complexity of the UTAP application lies in the local complexity. The structural complexity is the same because the calls to the UTAP interface are

considered operating system calls. Operating system calls are not considered when computing fanout. Therefore, the original and the UTAP applications have the same fanout and the structural complexities are the same. The local complexity is higher for the UTAP application because it uses more global variables than the original application. Several of these global variables are used to indicate the channel name parameter in a UTAP message. In a UTAP-compliant operating system, the channel names would not be considered global variables. If we did not count these as global variables when computing the local complexity of the UTAP application, the local complexity value would be much closer to that of the original application. The similarity in the complexity measures of the original and the UTAP applications indicates that developing and maintaining a UTAP-compliant application will not be any more difficult than implementing a non-UTAP-compliant application.

IV.2.3. Random Access Memory

The original application required 29.297 kilobytes of random access memory (RAM). The UTAP application and interface layer combined required 41.815 kilobytes of RAM. Again, we can view the interface layer as part of the operating system. The UTAP application alone required 30.332 kilobytes of RAM. Comparing this to the amount of RAM required by the original application, we find there is only a 1.035 kilobyte increase. This equates to only a 3.5 percent increase over the original application.

IV.2.4. Execution Time

The execution times for the original and the UTAP applications are given in Table IV.4. The 1 → 2 column indicates moving the parts from pallet one to pallet two. The 2 → 1 column indicates moving the parts from pallet two to pallet one. All times are in seconds.

		Speed:	100				Speed:	75	
	Original		UTAP			Original		UTAP	
	1 → 2	2 → 1	1 → 2	2 → 1		1 → 2	2 → 1	1 → 2	2 → 1
	81.049	85.330	91.081	95.514		91.978	98.330	101.936	108.500
	81.199	85.877	91.249	95.339		91.928	98.674	101.969	108.591
	81.043	84.878	91.288	95.849		91.890	98.698	101.869	108.457
	80.923	84.857	91.316	95.697		91.649	98.551	101.861	108.764
	81.150	85.024	91.352	95.631		91.853	98.646	101.697	108.563
Ave:	81.073	85.193	91.257	95.606	Ave:	91.860	98.580	101.866	108.575
		Speed:	50%				Speed:	25%	
	Original		UTAP			Original		UTAP	
	1 → 2	2 → 1	1 → 2	2 → 1		1 → 2	2 → 1	1 → 2	2 → 1
	113.625	125.905	123.968	135.033		191.686	212.138	199.038	219.406
	114.199	125.749	124.082	134.824		191.749	212.133	198.994	219.449
	114.559	125.820	124.334	134.804		191.666	212.138	199.031	219.488
	114.413	125.751	124.090	135.155		191.689	212.159	199.013	219.650
	114.816	125.689	123.971	135.242		191.712	212.132	199.055	219.546
Ave:	114.322	125.783	124.089	135.012	Ave:	191.700	212.140	199.026	219.508

Table IV.4. Execution Times (seconds)

Table IV.5 shows the differences in execution times between the original and the UTAP-compliant application. The values were obtained by subtracting the execution time of the original application from the execution time of the UTAP application. From this table, it

can be seen that, at all speeds, the original application runs faster than the UTAP application.

	Percentage of Maximum Speed			
	100%	75%	50%	25%
1 → 2	10.184	10.007	9.767	7.326
2 → 1	10.413	9.995	9.229	7.368
Average	10.299	10.001	9.498	7.347

Table IV.5. Execution Time Differences (seconds)

This was expected because of the increased processing required by the additional lines of code in the UTAP application. It is also noticeable that, as the speed decreases, the time difference between the original and the UTAP applications decreases. The implementation of UTAP on the Adept system did not affect the time required to move the manipulator from one position to another. The added time required by the UTAP application is a result of processing the additional lines of code. V+ continues to process program lines concurrent with robot motion as long as the lines do not require robot position information, sensor information, or cause another robot motion. Because of this, slower robot speeds allow more program processing during robot movement. Therefore, the smaller difference between execution times of the original and the UTAP applications during slower robot movement is explained by the increased processing time available during the robot movements.

Counting the interface layer, the UTAP application had 335 more lines of code than the original application. At maximum speed, the UTAP application took approximately 10.3 seconds longer to execute than the original application. Using these values, we can determine that each additional line of code adds 30.7 milliseconds to the

execution time. At 25 percent of maximum speed, the UTAP application took approximately 7.3 seconds longer. At this speed, each additional line of code adds 21.8 milliseconds to the execution time. Even if we add 1000 lines of code when we convert an application to UTAP compliance, we only add 31 seconds to the execution time. Of course, this assumes we are using V+, we have the same ratio of amount of movement to lines of code, and we use the larger value above to calculate the additional time.

IV.3. Summary

This chapter presented the methods used to implement the UTAP specification on the Adept manipulator. It presented the application chosen to convert to UTAP compliance and some of the difficulties faced during the conversion. The interface layer and how it was developed was then described. The chapter then gave a discussion of the test procedures and the performance measurements of the system.

The data obtained while measuring the Adept UTAP performance was then presented. It showed that the system performance was degraded by the implementation of UTAP. When comparing the original system to the UTAP application and the interface layer, performance degradation seemed high. However, the interface layer would not exist in a purely UTAP-compliant operating system. Thus, the original application can be compared to just the UTAP application and the interface layer viewed as part of the operating system. Under these circumstances, the UTAP application showed very little performance degradation. This indicates that retrofitting an existing robot system to UTAP would likely be unfeasible due to the performance issues.

However, a new robotic system design that incorporated UTAP would not have the performance degradation and would be feasible.

V. Conclusions and Recommendations

V.1. Conclusions

Several conclusions can be made from this thesis effort. First and foremost, implementing UTAP on two vastly different systems has shown that UTAP is a feasible telerobotic control architecture. Both systems were made UTAP-compliant for specific applications and the applications continued to performed all original tasks.

As expected, retrofitting both the Chimera system and the Adept system to UTAP compliance adversely affected the system performance. Unfortunately, difficulties with the Chimera system hindered the quantification of the performance degradation. Gain tuning Anchor's UTAP-compliant controller greatly improved it's performance. However, the data indicated that the original controller also needed gain tuning. The performance ratio between it and the UTAP controller would likely be returned back to that observed by Anchor by gain tuning the original controller.

Implementing UTAP on the Adept system has shown that retrofitting a system adds large amounts of overhead in the form of the interface layer. However, a system designed for UTAP compliance from the start will not have that overhead. Therefore, such a system should be able to perform as well as a non-compliant system.

The data obtained from the complexity measurements, source lines of code calculations, and RAM measurements of the Adept implementation indicates that developing and maintaining a UTAP-compliant application will not be any more difficult than implementing a non-UTAP-compliant application.

UTAP begins the telerobotic control architecture standardization effort. However, it has a long way to go. But, the effort has to start somewhere and UTAP has made a good start. It addresses the same goals as other successful standardization efforts and provides the building blocks to achieve those goals. With a continued effort to remove the inconsistencies in the UTAP specification and to provide the meaning behind each message, UTAP can make the next step towards standardizing the robotics community.

V.2. Recommendations

V.2.1. Improvements to the UTAP Specification

Determining the purpose of each UTAP message and which message to use for a particular task was the most difficult part of this thesis effort. The UTAP specification states that "the intent and meaning of UTAP API messages *should* be apparent from the message name" [9]. For most of the messages, this is true. However, finding the appropriate message for even basic tasks was sometimes difficult. For example, to move joint one of the manipulator, should the US_AXIS_SERVO_SET_POSITION message or the US_TLC_START_AUTOMATIC_MOTION message be used? What about the US_TLC_START_MANUAL_MOTION message? The UTAP specification should clearly define every message. The purpose of the message should be explicitly stated and guidelines presented for the use of the message. Additionally, the parameters associated with each message should be completely described, including whether they are input or output parameters and, where possible, the expected range of values.

In addition to defining the messages, the UTAP specification should define the purpose of each UTAP module. I had expected the Operator Interface module to contain

messages related to the input and output of information between the system and the operator. However, the specification does not define messages for these tasks under this module. This may not be the purpose of this module. The specification does not make the purpose of this or other modules clear. Clarifying each module's purpose would aid the understanding of the messages contained within the module. Implementors would also be less likely incorporate into a module the wrong functionality.

Focusing on "what to pass" rather than "how to pass" allows the specification to remain generalized. However, an implementation depends on the decision of "how to pass." A UTAP-compliant application based on passing information and control flow via function calls will not run on a system that uses message passing. The UTAP specification must specify one or the other method or it must state that UTAP-compliant systems must accommodate applications using either method.

Anchor correctly states that "the UTAP document is 'C-centric', but the specification is intended to be language-independent" [3]. The examples and message definitions should be modified as Anchor suggests. I recommend using some form of pseudo-code.

V.2.2. Future Research

A valid method for gain tuning the Chimera/PUMA system should be developed. The method should then be used to tune both the UTAP-compliant and the non-UTAP-compliant controllers. Once both controllers are tuned, the performance of the UTAP-compliant controller could be more accurately quantified.

Capt Lemley's research efforts toward designing a purely UTAP-compliant operating system (OS) should be continued [8]. The OS should be developed for a hardware platform that has an existing non-UTAP robotic control OS. Once the UTAP-compliant OS is in place, applications could be written with identical functionality for both the UTAP-compliant OS and the original OS. Neither application would require an interface layer since they both were written for their native OS. Comparing the performance of the two applications would define the impact on the system levied by UTAP conformance.

Finally, the Adept UTAP implementation should be expanded to include more of the messages defined by the UTAP specification. This would further cement the feasibility of UTAP and provide more data on the performance issues.

V.3. Summary

Table V.1 provides a summary of the conclusions of this thesis. Table V.2 summarizes my recommendations for improvements to the UTAP specification and Table V.3 summarizes my recommendations for future research. This thesis effort investigated the feasibility of the Unified Telerobotic Architecture by implementing the UTAP specification on an existing robotic system and measuring the performance of this and a previous implementation. This effort has shown that the architecture is feasible but requires additional clarification. This thesis has also shown that retrofitting existing systems exacts a heavy performance toll on the system, but designing a new system to be UTAP-compliant is an appropriate activity to pursue. The results of this thesis will aid in

making the decision to continue pursuing the UTAP specification as an Air Force standard.

	Conclusions
1	Gain tuning improved the performance of Anchor's UTAP controller
2	The original controller needs be have the gains tuned
3	Tuning the original controller would return the performance ratio between it and Anchor's back to ratio observed by Anchor
4	UTAP is a feasible telerobotic control architecture
5	Retrofitting a system adversely affects its performance
6	A system designed from the beginning to be UTAP-compliant will not have the adverse performance effects
7	Developing applications for a UTAP-compliant system is not any more difficult than developing applications for a non-UTAP-compliant system

Table V.1. Summary of Conclusions

	Recommendations
1	Explicitly define each UTAP message including the parameters
2	Explicitly define each UTAP module including the functions and purpose
3	Decide on a message passing format or state that both must be supported
4	Use pseudo-code instead of C code for examples

Table V.2. Summary of Recommendations for the UTAP Specification

	Recommendations
1	Develop a gain tuning method and conduct further measurements on Anchor's UTAP-compliant controller
2	Continue Capt Lemley's development of an originally designed UTAP-compliant system
3	Expand the Adept UTAP implementation

Table V.3. Summary of Recommendations for Future Research

Bibliography

1. Adept Technology, Inc. V+ Language User's Guide. Part Number 00961-00230, Rev A, April 1994.

2. Adept Technology, Inc. V+ Quick Reference Guide. Part Number 00961-01400, Rev B, June 1993.

3. Anchor, Kevin P. Design and Evaluation of Standard Telerobotic Control Software. MS thesis, AFIT/GE/ENG/95D-01. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1995.

4. Card, D. N. and W. W. Agresti. "Measuring Software Design Complexity," The Journal of Systems and Software 8: 185-197 (1988).

5. DaCosta, F. and others. "Towards Rapid Implementation of Robotic Systems." Preliminary draft report of work done in late 1994 and early 1995 by Advanced Cybernetics Group, Inc., 227 Humboldt Court, Sunnyvale, CA 94089.

6. Department of Defense. Reference Manual for the Ada Programming Language. ANSI/MIL-STD-1815A-1983. Washington: GPO, 17 February 1983.

7. Klafter, Richard D. and others. Robotic Engineering, An Integrated Approach. Englewood Cliffs, NJ: Prentice-Hall, 1989.

8. Lemley, Kendall. Implementation Of An Ada95 Cross-Compiler For The Real-Time Executive For Military Systems (RTEMS). MS thesis, AFIT/GCS/ENG/96D-16. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1996.

9. National Institute of Standards and Technology (NIST) Intelligent Systems Division. "Unified Telerobotic Architecture Project (UTAP) Interface Document." Working Draft. Gaithersburg, MD: NIST, 18 June 1994.

10. Quarterman, John S. and Susanne Wilhelm. UNIX, POSIX, and Open Systems : The Open Standards Puzzle. Reading, MA: Addison-Wesley, 1993.

11. Robotics and Automation Center for Excellence, World Wide Web Page, http://www.kelly-afb.org/links/orgs/race/utap.htm.

12. Taylor, Paul M. Understanding Robotics. Hong Kong: CRC Press, 1990.

APPENDIX A

V+/UTAP System Users Manual

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

Adept System Power-up and Calibration

Steps:

1. Turn the Adept system's power switch to "On".
2. Plug in the air dryer power cord.
3. Open the air valve on the compressor tank.
4. Turn the compressor power switch to "Auto".
5. Open both air valves located at the back of the manipulator table.
6. Enable power to the manipulator by typing the following at the monitor prompt:

   . enable power

7. Calibrate the Adept manipulator by typing the following at the monitor prompt:

   . calibrate

8. Confirm that calibration should occur by responding with a "y":

   Are you sure (y/n): **y**

Loading the V+/UTAP Interface

Steps:

1. Change to the UTAP directory by issuing the following monitor command:

   . def = c:\utap

2. Load the V+/UTAP interface modules into system memory. Each module can be loaded individually with the **load** command. For example, the sensor module can be loaded with the following command:

   . load sc.pg

   A script can be used to load all of the modules instead of loading them individually. This is accomplished with the following commands:

   . load utap.pg
   . commands setup

At this point, a UTAP-compliant application can be loaded and executed.

Loading and Executing an Application

Steps:

1. Change to the directory containing the application. For example, the following command will change to the directory containing a palletizing application:

   . def = c:\utap\app

2. Load the application into system memory using the **load** command. The UTAP-compliant palletizing application is loaded as follows:

   . load demo.v2
   . load pallet.pg
   . load force.pg

3. Begin application execution by issuing the **execute** command with the main program. The palletizing application is started with the following command:

   . execute utap.demo

Adept System Shutdown

Steps:

1. Close the air valves on the back of the manipulator table.

2. Turn the air compressor power switch to "Off".

3. Close the air valve on the compressor tank.

4. Unplug the air dryer.

5. Turn the Adept System power switch to "Off".

APPENDIX B

Adept V+ Tutorial

I. Introduction.

V+ is Adept Technology's proprietary control system and programming language used with their industrial robots. This tutorial presents exerpts and information from the V+ Language User's Guide [1] and the V+ Quick Reference Guide [2]. It presents the basic information needed to operate and program the Adept 550 manipulator. The tutorial is divided into three sections. The first section covers general V+ issues. The second section describes the basic control system commands (called monitor commands). The third section describes some programming language commands (called language commands).

II. General V+ Issues

II.1. Program Types

There are two types of V+ programs, executable and command.

II.1.1. Executable Programs

There are two classes of executable programs, robot control programs and general programs. A robot control program is a V+ program that directly controls a robot. A general program is any program that does not control a robot. A mixture of robot control and general programs can execute at the same time, but only one robot control program can have control of the robot at any given time. Robot control programs can contain any of the V+ program instructions. With the exception of the **BRAKE** instruction, a general program cannot execute any instruction that affects the robot motion. They can access all other V+ features including AdeptVision, digital signal lines, and the manual control pendant. Executable programs are initiated by the **EXECUTE** monitor command.

II.1.2. Command Programs

Command programs are similar to MS_DOS batch programs or UNIX scripts. They can contain any monitor command and can access language commands through the monitor **DO** command. Command programs are initiated by the **COMMANDS** monitor command.

II.2. Data Types

II.2.1. Declaration and Allocation

V+ does not require you to declare variables or their data types. The first use of a variable will determine its data type and allocated space for that variable.

II.2.2. Variable Name Requirements

The requirements for a valid variable name are:

1. Adept reserved keywords cannot be used.
2. The first character of a variable name must be a letter.
3. The characters after the first character may be letters, numbers, periods, and the underline character.
4. Only the first 15 characters in a variable name are significant.

II.2.3. String Data Type

Variable names are preceded with a "$" sign to indicate that they contain character string data. The dollar sign is not considered in the character count of the variable name. For example, the program instruction:

$string_1 = "Adept V+"

allocates the string variable "string_1" and assigns it the value "Adept V+". The character count of the variable name "string_1" is 8. String variables can contain from 0 to 128 characters and any ASCII character can appear in a string variable. String constants can also have 0 to 128 characters. However, only ASCII characters 32 through 126 (excluding ASCII 34, ") can appear in a string constant.

II.2.4. Real and Integer Data Types

Numbers that have a whole number and a fractional part belong to the data type "real". Numeric values having only a whole number belong to the data type "integer". In general, V+ does not require you to differentiate between these two data types. If an integer is required and you supply a real, V+ will promote the real to an integer by rounding (not truncation). Where real values are required, V+ considers an integer a special case of a real that does not have a fractional part. The default real type is a signed, 32-bit IEEE single-precision number. Real values can also be stored as 64-bit IEEE double-precision numbers if they are specifically typed using the DOUBLE instruction.

The range of integer values is:

-16,777,216 to 16,777,215

The range of single-precision real values is:

$\pm 3.4*10^{38}$

The range of double-precision real values is:

$\pm 1.8*10^{307}$

In almost all situations where a numeric value of variable can be used, a numeric expression can also be used. The following examples all result in "x" having the same value:

```
x = 3
x = 6 / 2
x = SQRT(9)
x = SQR(2) - 1
x = 9 MOD 6
```

V+ does not have a specific logical (Boolean) data type. Any numeric value, variable, or expression can be used as a logical data type. V+ considers 0 to be false and any other value to be true. When a real value is used as a logical data type, the value is first promoted (rounded) to an integer. There are four logical constants, TRUE and ON that will resolve to -1, and FALSE and OFF that will resolve to 0. These constants can be used anywhere a Boolean expression is expected.

II.2.5. Location Data Types

V+ uses two data types to represent locations, transformations and precision points. A transformation is a set of six components that uniquely identifies a location in Cartesian space and the orientation of the motion device end-of-arm tooling at that location. A transformation can also represent the location of an arbitrary local reference frame. The first three components of a transformation variable are the values for the points on the X, Y, and Z axes. The second three components of a transformation variable specify the orientation of the end-of-arm tooling. These three components are yaw, pitch, and roll.

A precision point includes an element for each "joint" in the motion device. Rotational joint values are measured in degrees; translational joint values are measured in millimeters. These values are absolute with respect to the motion device's home sensors and cannot be made relative to other locations or coordinate frames.

II.2.6. Arrays

V+ supports arrays of up to three dimensions. Any V+ data type can be stored in an array. Like simple variables, array allocation and typing is dynamic.

II.2.7. Variable Classes

In addition to having a data type, variables belong to one of three classes, GLOBAL, LOCAL, or AUTOMATIC. These classes determine how a variable can be altered by different calling instances of a program.

II.2.7.1. Global Variables

This is the default class. Unless a variable is specifically declared to be LOCAL or AUTO, a newly created variable will be considered global. Global variables can be accessed by any executing program.

II.2.7.2. Local Variables

Local variables are created by a program instruction similar to:

```
LOCAL the_local_var
```

In this example, "the_local_var" is declared as a local variable. Local variables are available to all calling instances of a program. In other words, a local variable that is changed during a recursive call of a program will be changed in all instances of the recursive program.

II.2.7.3. Automatic Variables

Automatic variables are created by a program instruction similar to:

```
AUTO the_auto_var
```

In this example, "the_auto_var" is declared as an automatic variable. Automatic variables can only be changed by a particular calling instance of a program.

II.3. Operators

II.3.1. Assignment Operator

The equal sign (=) is used to assign a value to a numeric or string variable. The variable being assigned a value must appear by itself on the left side of the operator. The right side of the equal sign can contain any variable or value of the same data type as the left side, or any expression that resolves to the same data type. Location variables use the **SET** instruction for assignment and may not use the equal sign assignment operator.

II.3.2. Mathematical Operators

V+ uses the standard mathematical operators; +, -, *, /, and MOD.

II.3.3. Relational Operators

V+ uses the standard relational operators; ==, <, >, <=, >=, and <>.

II.3.4. Logical Operators

V+ uses the standard logical operators; NOT, AND, OR, and XOR.

II.3.5. String Operator

In V+, strings can be concatenated, or joined, using the "+" operator.

II.3.6. Order of Evaluation

Expressions are not evaluated in a simple left to right fashion in V+. Table B.1 shows the order of evaluation. Operators on the same line have the same precedence and are evaluated left to right. Parenthesis can be used in the normal manner to change the evaluation order within an expression.

Evaluation Order	Operator
First	NOT
	- (Unary minus)
	*, /, MOD, AND
	+, -, OR, XOR
Last	==, <=, >=, <, >, <>

Table B.1. Order of Operator Evaluation

II.4. The SEE Editor

The SEE editor is a full functioned screen editor that is provided with the V+ control system and programming language. The SEE editor performs syntax checks on each line as it is entered. The beginning of each invalid line is marked with a question mark. The editor automatically capitalizes language keywords and insert the appropriate amount of spaces between keywords, variables, and operators. Additionally, the editor automatically indents program lines by the proper amount based on the level of nesting. The ".PROGRAM" line of each program is inserted when the SEE editor first creates the program. The editor also places the .END statement at the end of each program. It is important to note that the editor works on programs in system memory. It does not read

programs from disk or write programs to disk. This must be accomplished with monitor commands. Use of the SEE editor is described in the appropriate Adept Manuals.

III. Monitor Commands

Monitor commands are commands that are entered at the system prompt. The following section describes some of the most used commands. They are grouped into the following three categories; system control and information, disk commands, and program control.

Note: Keywords are shown in uppercase and arguments are shown in lowercase. Required keywords, parameters, and marks such as equal signs and parenthesis are shown in bold type; optional keywords, parameters, and marks are shown in regular type.

III.1. System Control and Information Commands

These commands control various aspects of the system or provide information about the system.

CALIBRATE
Initialize the robot positioning system.

DELETEL loc_var, . . ., loc_var
Delete the named location variables from the system memory.

DELETER real_var, . . ., real_var
Delete the named real-valued variables from the system memory.

DELETES str_var, . . ., str_var
Delete the named string variables from the system memory.

DIRECTORY select
Display the names of some or all the programs in the system memory.

DO instruction
Execute a single program instruction as though it were the next step in the current main control program, or the next step in the specified task/program context.

ENABLE switch, . . ., switch
Turn on one or more system control switches.

FREE select
Display the percentage of available system memory not currently in use.

HERE loc_variable

Define the value of a transformation or precision-point variable to be equal to the current robot location.

LISTL location, . . ., location
Display the values of the list locations.

LISTR expression, . . ., expression
Display the values of the real variables specified.

LISTS string, . . ., string
Display the values of the string variables specified.

SEE program_name/qualifier
Invoke the screen-oriented program editor to allow a program to be created, viewed, or modified.

STATUS select
Display status information for the system and the programs being executed.

SPEED speed_factor
Specify the speed of all subsequent robot motions commanded by a robot control program.

ZERO select
Reinitialize the V+ system and delete all programs and data in the system memory. Delete all user-defined windows, fonts, and icons from graphics memory.

III.2. Program Control Commands

These commands control programs that are currently loaded in memory.

ABORT task_num
Terminate execution of a control program.

COMMANDS program
Initiate processing of a command program.

COPY new_program = old_program
Create a new program in system memory as a copy of an existing program in system memory.

DELETE program, . . ., program
Delete the listed programs from the system memory.

EXECUTE task_num program(param_list), cycles, step, priority[i]

Begin execution of a control program.

KILL task_num
Clear a program execution stack and detach any I/O devices that are attached.

RENAME new_program_name = old_program_name
Change the name of a user program in memory to the new name provided.

III.3. Disk Commands

These commands perform functions related to files and magnetic storage media.

FCOPY new_file = old_file
Copy the information in an existing disk file to a new disk file.

FDELETE file_spec
Delete one or more disk files matching the given file specification.

FDIRECTORY/qualifier file_spec
Display information about the files on a disk, along with the amount of space still available for storage. Create and delete subdirectories on disks.

FLIST file_spec
List the contents of the specified disk file on the system terminal.

FORMAT A:/qualifier
Initialize and erase a FLOPPY disk.

FRENAME new_file = old_file
Change the name of a disk file.

LOAD switch **file_spec**
Load the contents of the specified disk file into the system memory.

STORE /levels **file_spec** = program_name, . . ., program_name
Store programs and variables in a disk file.

STOREL /levels **file_spec** = program_name, . . ., program_name
Store location variables in a disk file.

STORER /levels **file_spec** = program_name, . . ., program_name
Store real variables in a disk file.

STORES /levels **file_spec** = program_name, . . ., program_name
Store string variables in a disk file.

IV. Language Commands

Language commands are commands that are used within executable programs. The following section describes some of the most used commands.

Note: Keywords are shown in uppercase and arguments are shown in lowercase. Required keywords, parameters, and marks such as equal signs and parenthesis are shown in bold type; optional keywords, parameters, and marks are shown in regular type.

APPRO location, distance
APPROS location, distance
Start a robot motion toward a location defined relative to the specified location.

ATTACH (lun)
ATTACH (alun, mode) $device
Make a device available for use by the application program.

AUTO variable, . . ., variable
AUTO type variable, . . ., variable
Declare temporary variables that are automatically created on the program stack when the program is entered.

BRAKE
Abort the current robot motion

BREAK
Suspend program execution until the current motion completes.

CALL program(arg_list)
Suspend execution of the current program and continue execution with a new program (that is, a subroutine).

CASE value OF
Initiate processing of a CASE structure by defining the value of interest.

DEPART distance
DEPARTS distance
Start a robot motion away from the current location.

DETACH (logical_unit)
Release a specified device from the control of the application program.

DO
Introduce a DO program structure.

DOS string, error

Execute a program instruction defined by a string expression.

END

Mark the end of a control structure.

EXECUTE program(param_list)
EXECUTE task_num program(param_list)

Begin execution of a control program.

FOPENR (lun, record_len, mode) **file_spec**
FOPENW . . .
FOPENA . . .
FOPEND . . .

Open a disk file for read-only, read-write, read-write-append, or read-directory, respectively, as indicated by the last letter of the instruction name.

FOR loop_var = initial TO final STEP increment

Execute a group of program instructions a certain number of times.

GLOBAL type **var**, . . ., var

Declare a variable to be global and specify the type of the variable.

GOTO label

Perform an unconditional branch to the program step identified by the given label.

HALT

Stop program execution and do not allow the program to be resumed.

IF logical_expr GOTO label

Branch to the specified label if the value of a logical expression is TRUE (nonzero).

IF logical_expr THEN

Conditionally execute a group of instructions (or one of two groups) depending on the result of a logical expression.

KEYMODE first_key, last_key = **mode**, setting

Set the behavior of a group of keys on the manual control pendant.

LATCH ( )

Return a transformation value representing the location of the robot at the occurrence of the last external trigger.

LATCHED (select)

Return the status of the external trigger, and of the information it causes to be "latched".

LOCAL type **variable**, . . ., variable
Declare permanent variables that are defined only within the current program.

MOVE location
MOVES location
Initiate a robot motion to the position and orientation described by the given location.

PAUSE
Stop program execution but allow the program to be resumed.

PENDANT (select)
Return input from the manual control pendant.

PROMPT output_string, variable_list
Display a string on the system terminal and wait for operator input.

REACT signal_num, program, priority
Initiate continuous monitoring of a specified digital signal and automatically trigger a subroutine call if the signal properly transitions.

REACTI signal_num, program, priority
Initiate continuous monitoring of a specified digital signal. Automatically stop the current robot motion if the signal properly transitions and optionally trigger a subroutine call.

READ (lun, record_num, mode) **var_list**
Read a record from an open file, or from an attached device that is not file oriented.

RETURN
Terminate execution of the current subroutine and resume execution of the last-suspended program at the step following the CALL or CALLS instruction that caused the subroutine to be invoked.

SET location_var = location_value
Set the value of the location variable on the left equal to the location value on the right of the equal sign.

SIG (signal_num, . . ., signal_num)
Return the logical AND of the states of the indicated digital signals.

SIGNAL signal_num, . . ., signal_num

Turn "on" or "off" external digital output signals or internal software signals.

SPEED speed_factor
Set the nominal speed for subsequent robot motions.

SPEED (select)
Return one of the system motion speed factors.

STOP
Terminate execution of the current program cycle.

TIMER timer_number = time_value
Set the specified system timer to the given time value.

TIMER (timer_number)
Return the current time value (in seconds) of the specified system timer.

UNTIL expression
Indicate the end of a DO ... UNTIL control structure and specify the expression that is evaluated to determine when to exit the loop. The loop continues to be executed until the expression value is non-zero.

WAIT.EVENT mask, timeout
Suspend program execution until a specified event has occurred, or until a specified amount of time has elapsed.

WHILE condition DO
Initiate processing of a WHILE structure if the condition is TRUE, or skipping of the WHILE structure if the condition is initially FALSE.

WRITE (lun, record_num, mode**) format_list**
Write a record to an open file, or to an attached device that is file oriented.

APPENDIX C

UTAP Messages

Note: Bold and Italicized messages have been implemented in this Thesis Project. An asterisk indicates a message added to the specification for the purpose of this 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

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_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
- US_AXIS_SERVO_LOAD_CYCLE_TIME
- US_AXIS_SERVO_LOAD_PID_GAIN
- US_AXIS_SERVO_LOAD_JOINT_LIMIT
- US_AXIS_SERVO_LOAD_VELOCITY_LIMIT
- US_AXIS_SERVO_LOAD_GAIN_LIMIT
- US_AXIS_SERVO_LOAD_DAMPING_VALUES
- 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_ORIENT
- 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
US_SENSOR_GET_ATTRIBUTES_READING
US_VECTOR_SENSOR_GET_READING
US_FT_SENSOR_POST_READING
US_SCALAR_SENSOR_POST_READING
US_VECTOR_SENSOR_POST_READING
US_2D_SENSOR_LOAD_ARRAY_PATTERN
US_2D_SENSOR_USE_ARRAY_TYPE
US_2D_SENSOR_GET_READING
US_2D_SENSOR_POST_READING
US_IMAGE_USE_FRAME_GRAB_MODE
US_IMAGE_USE_HISTOGRAM_MODE
US_IMAGE_USE_CENTROID_MODE
US_IMAGE_USE_GRAY_LEVEL_MODE
US_IMAGE_USE_THRESHOLD_MODE
US_IMAGE_COMPUTE_SPATIAL_DERIVATIVES_ MODE
US_IMAGE_COMPUTE_TEMPORAL_ DERIVATIVES_MODE
US_IMAGE_ USE_SEGMENTATION_MODE
US_IMAGE_ USE_RECOGNITION_MODE
US_IMAGE_COMPUTE_RANGE_MODE
US_IMAGE_COMPUTE_FLOW_MODE
US_IMAGE_LOAD_CALIBRATION
US_IMAGE_SET_POSITION
US_IMAGE_ADJUST_POSITION
US_IMAGE_ADJUST_FOCUS
US_IMAGE_POST_SPECIFICATION
US_IMAGE_POST_PIXEL_MAP_READING
US_IMAGE_POST_HISTOGRAM_READING
US_IMAGE_POST_XY_CHAR_READING
US_IMAGE_POST_BYTE_SYMBOLIC_READING
US_IMAGE_POST_THRESHOLD_READING
US_IMAGE_POST_SPATIAL_DERIVATIVE_ READING
US_IMAGE_POST_TEMPORAL_DERIVATIVE_ READING
US_IMAGE_POST_RECOGNITION_READING
US_IMAGE_POST_RANGE_READING
US_IMAGE_POST_FLOW_READING

PROGRAMMABLE_IO (11)
US_PIO_ENABLE
US_PIO_DISABLE
US_PIO_SET_MODE
US_PIO_CONTROL_WRITE
US_PIO_LOAD_SCALE
US_PIO_DATA_WRITE
US_PIO_DATA_READ
US_PIO_BIT_READ
US_PIO_BIT_SET
US_PIO_TOGGLE_BIT
US_PIO_POST_DATA

TASK_LEVEL_CONTROL (78)
US_TLC_USE_JOINT_REFERENCE_FRAME
US_TLC_USE_CARTESIAN_REFERENCE_FRAME
US_TLC_USE_REPRESENTATION_UNITS
US_TLC_USE_ABSOLUTE_POSITIONING_MODE
US_TLC_USE_RELATIVE_POSITIONING_MODE
US_TLC_USE_WRIST_COORDINATE_FRAME
US_TLC_USE_TOOL_TIP_COORDINATE_FRAME
US_TLC_CHANGE_TOOL
US_TLC_USE_MODIFIED_TOOL_LENGTH_ OFFSETS
US_TLC_USE_NORMAL_TOOL_LENGTH_ OFFSETS
US_TLC_USE_NO_TOOL_LENGTH_OFFSETS
US_TLC_USE_KINEMATIC_RING_POSITIONING_ MODE
US_TLC_START_MANUAL_MOTION
US_TLC_STOP_MANUAL_MOTION
US_TLC_START_AUTOMATIC_MOTION
US_TLC_STOP_AUTOMATIC_MOTION
US_TLC_START_TRANSVERSE_MOTION
US_TLC_STOP_TRANSVERSE_MOTION
US_TLC_START_GUARDED_MOTION
US_TLC_STOP_GUARDED_MOTION
US_TLC_START_COMPLIANT_MOTION
US_TLC_STOP_COMPLIANT_MOTION
US_TLC_START_FINE_MOTION
US_TLC_STOP_FINE_MOTION
US_TLC_START_MOVE_UNTIL_MOTION
US_TLC_STOP_MOVE_UNTIL_MOTION
US_TLC_START_STANDOFF_DISTANCE
US_TLC_STOP_STANDOFF_DISTANCE
US_TLC_START_FORCE_POSITIONING_MODE
US_TLC_STOP_FORCE_POSITIONING_MODE
US_TLC_LOAD_DOF
US_TLC_LOAD_CYCLE_TIME
US_TLC_LOAD_REPRESENTATION_UNITS
US_TLC_LOAD_LENGTH_UNITS
US_TLC_LOAD_RELATIVE_POSITIONING
US_TLC_ZERO_RELATIVE_POSITIONING
US_TLC_ZERO_PROGRAM_ORIGIN
US_TLC_LOAD_KINEMATIC_RING_ POSITIONING_MODE
US_TLC_LOAD_BASE_PARAMETERS
US_TLC_LOAD_TOOL_PARAMETERS
US_TLC_LOAD_OBJECT
US_TLC_LOAD_OBJECT_BASE
US_TLC_LOAD_OBJECT_OFFSET
US_TLC_LOAD_DELTA
US_TLC_LOAD_OBSTACLE_VOLUME
US_TLC_LOAD_NEIGHBORHOOD
US_TLC_LOAD_FEED_RATE
US_TLC_LOAD_TRAVERSE_RATE
US_TLC_LOAD_ACCELERATION
US_TLC_LOAD_JERK
US_TLC_LOAD_PROXIMITY
US_TLC_LOAD_CONTACT_FORCES
US_TLC_LOAD_JOINT_LIMIT
US_TLC_LOAD_CONTACT_FORCE_LIMIT
US_TLC_LOAD_CONTACT_TORQUE_LIMIT
US_TLC_LOAD_SENSOR_FUSION_POS_LIMIT
US_TLC_LOAD_SENSOR_FUSION_ORIENT_LIMIT
US_TLC_LOAD_SEGMENT_TIME
US_TLC_LOAD_TERMINATION_CONDITION
US_TLC_INCR_VELOCITY

US_TLC_INCR_ACCELERATION
US_TLC_SET_GOAL_POSITION
US_TLC_GOAL_SEGMENT
US_TLC_ADJUST_AXIS
US_TLC_UPDATE_SENSOR_FUSION
US_TLC_SELECT_PLANE
US_TLC_USE_CUTTER_RADIUS_COMPENSATION
US_TLC_START_CUTTER_RADIUS_COMPENSATION
US_TLC_STOP_CUTTER_RADIUS_COMPENSATION
US_TLC_STRAIGHT_TRAVERSE
US_TLC_ARC_FEED
US_TLC_STRAIGHT_FEED
US_TLC_PARAMETRIC_2D_CURVE_FEED
US_TLC_PARAMETRIC_3D_CURVE_FEED
US_TLC_NURBS_KNOT_VECTOR
US_TLC_NURBS_CONTROL_POINT
US_TLC_NURBS_FEED
US_TLC_TELEOP_FORCE_REFLECTION_UPDATE

TASK_DESCRIPTION (10)
US_TDS_LOAD_USER
US_TDS_SELECT_PROGRAM
US_TDS_EXECUTE_PROGRAM
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_PTPS_SELECT_AGENT
US_PTPS_SELECT_TOOL
US_PTPS_SELECT_SENSOR
US_PTPS_INTERP_RUN_PLAN
US_PTPS_INTERP_HALT_PLAN
US_PTPS_INPUT_REQUEST
US_PTPS_OUTPUT_ENABLE_SUBSYSTEM

TASK_PROGRAM_SEQUENCING (10)
US_TPS_FREESPACE_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)
US_ADS_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 the UTAP Specification)

US_GET_EXT_LOCATION_DATA
US_SET_EXT_LOCATION_DATA
US_GRIPPER_CLOSE
US_GRIPPER_OPEN
US_FT_SENSOR_DISABLE
US_FT_SENSOR_ENABLE
US_FT_SENSOR_MODE_SELECT

APPENDIX D

Trajectory Plots

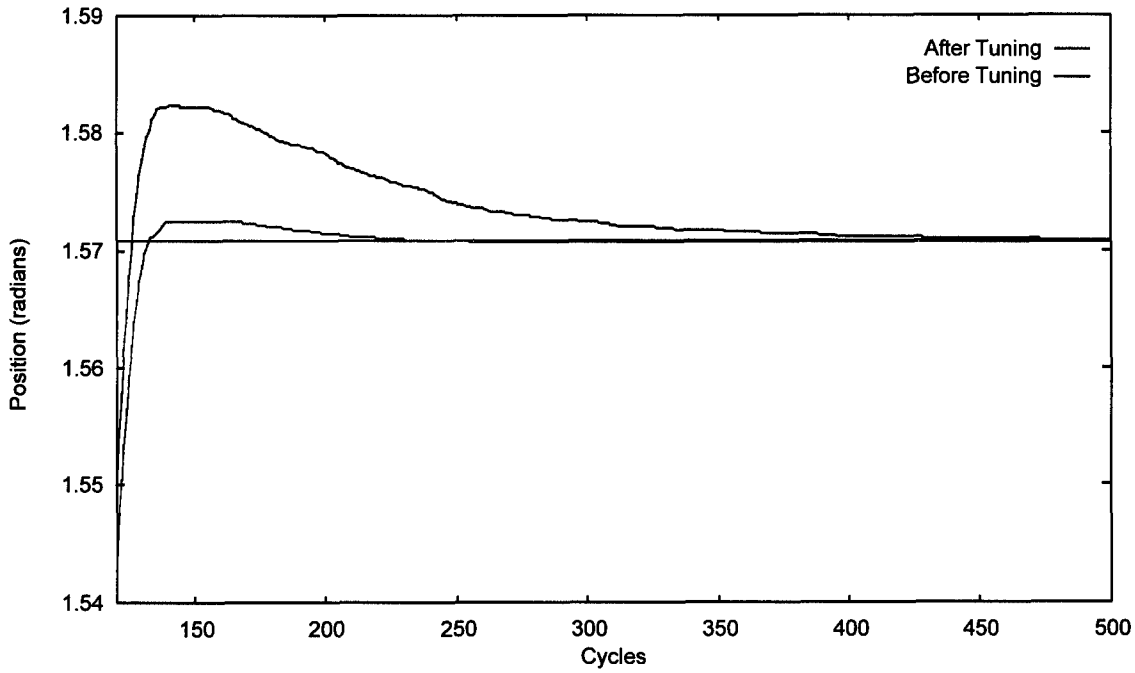


Figure D.1. Joint 1 Position Plots

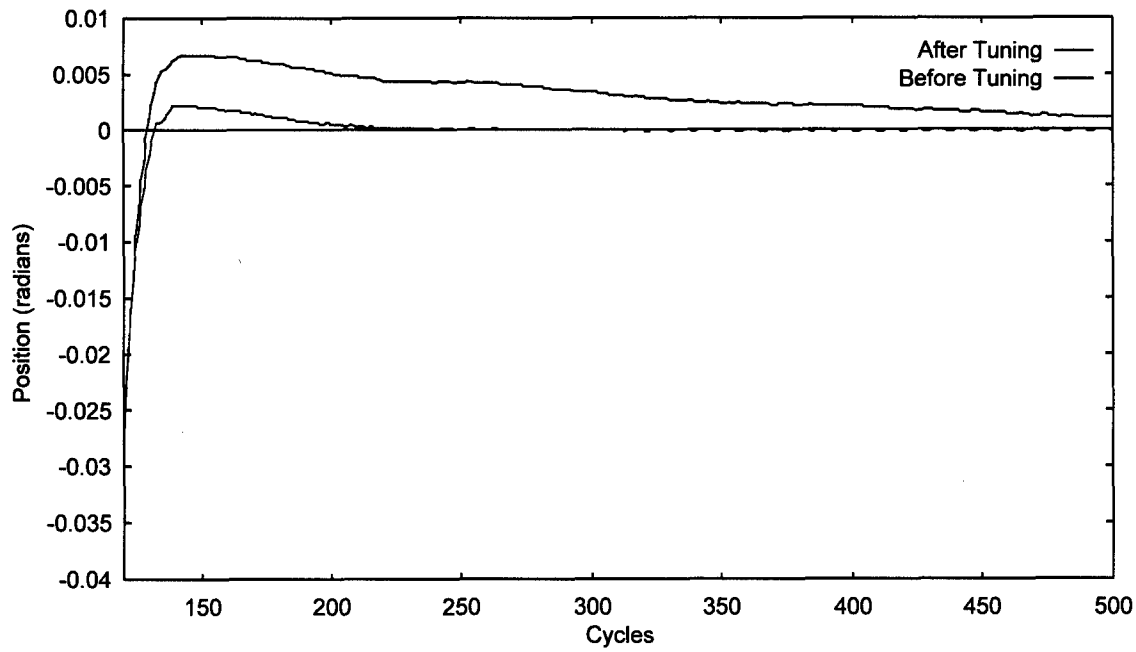


Figure D.2. Joint 2 Position Plots

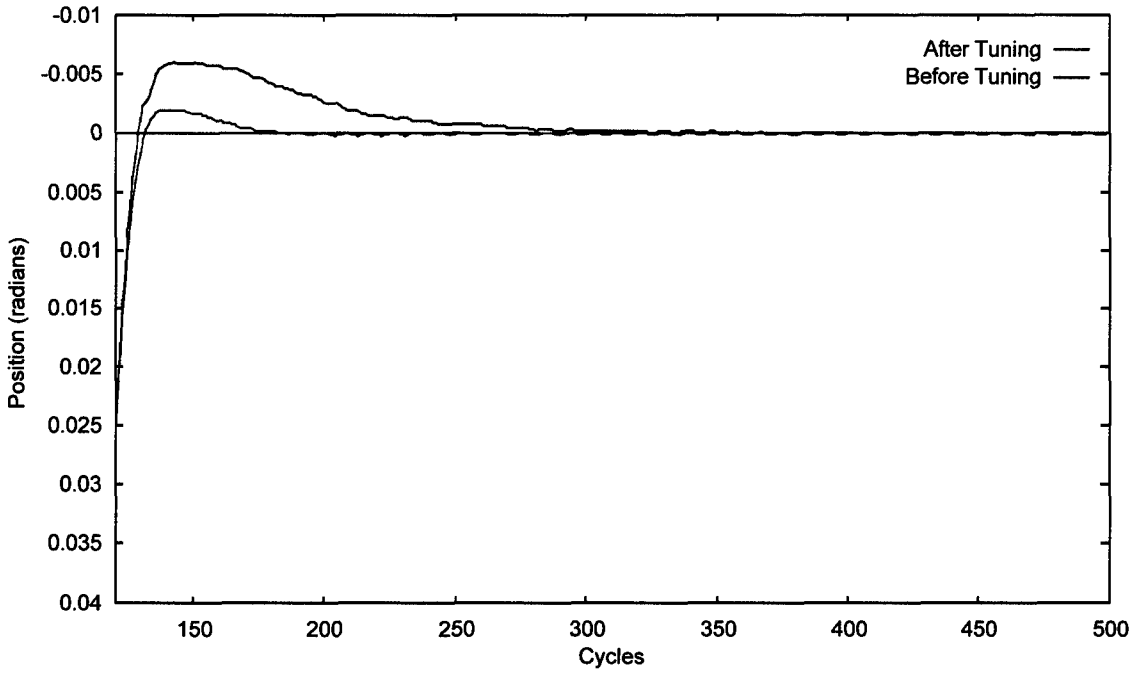


Figure D.3. Joint 3 Position Plots

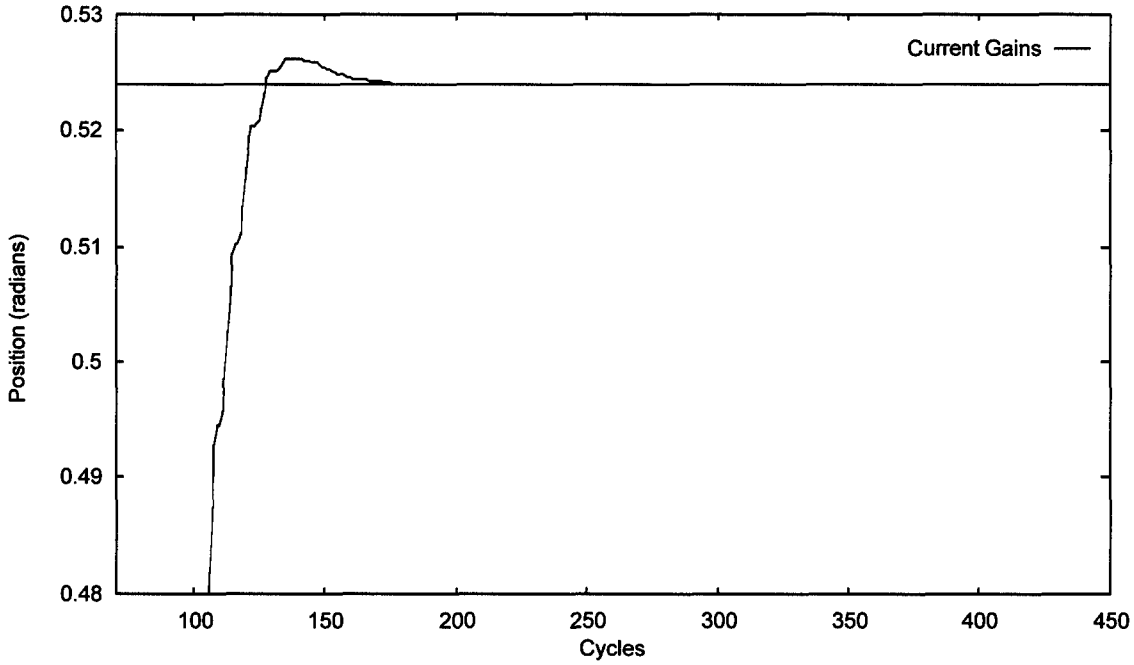


Figure D.4. Joint 4 Position Plots

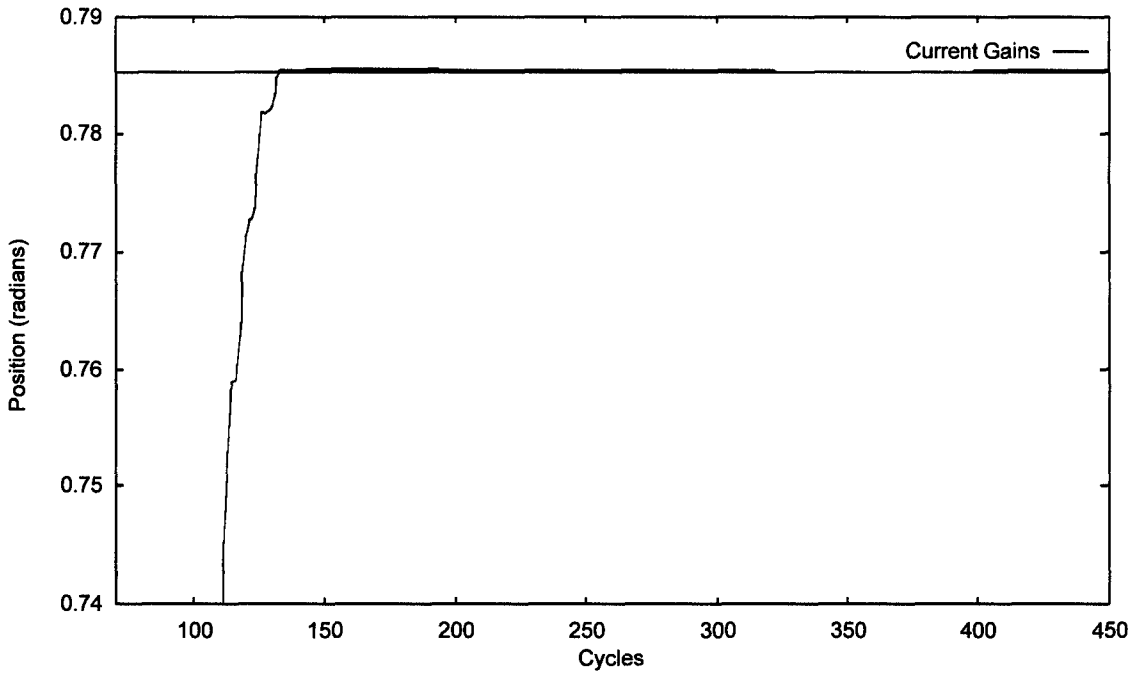


Figure D.5. Joint 5 Position Plots

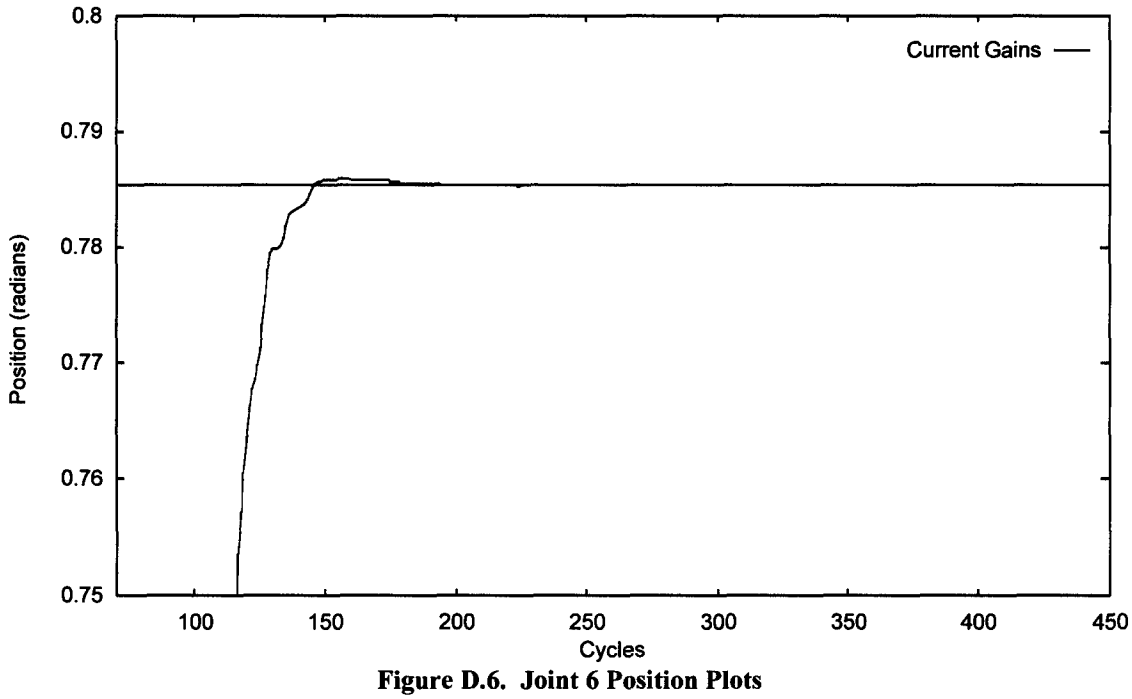


Figure D.6. Joint 6 Position Plots

APPENDIX E

Position Error Plots

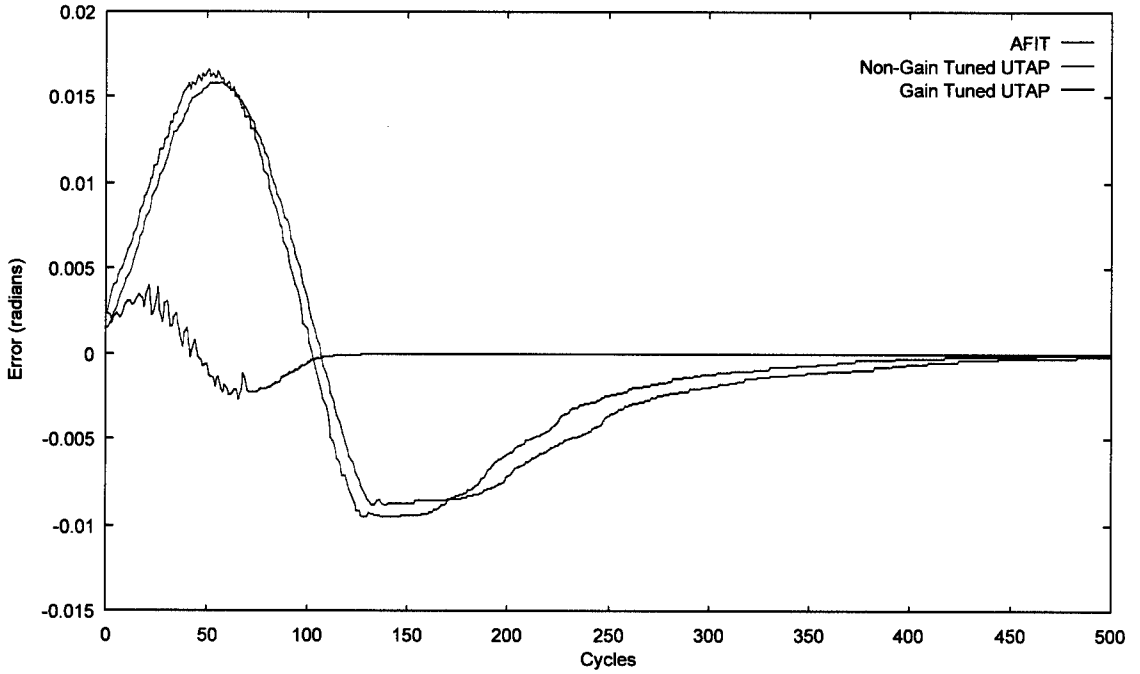


Figure E.1. Joint 1 Error for Nominal Trajectory

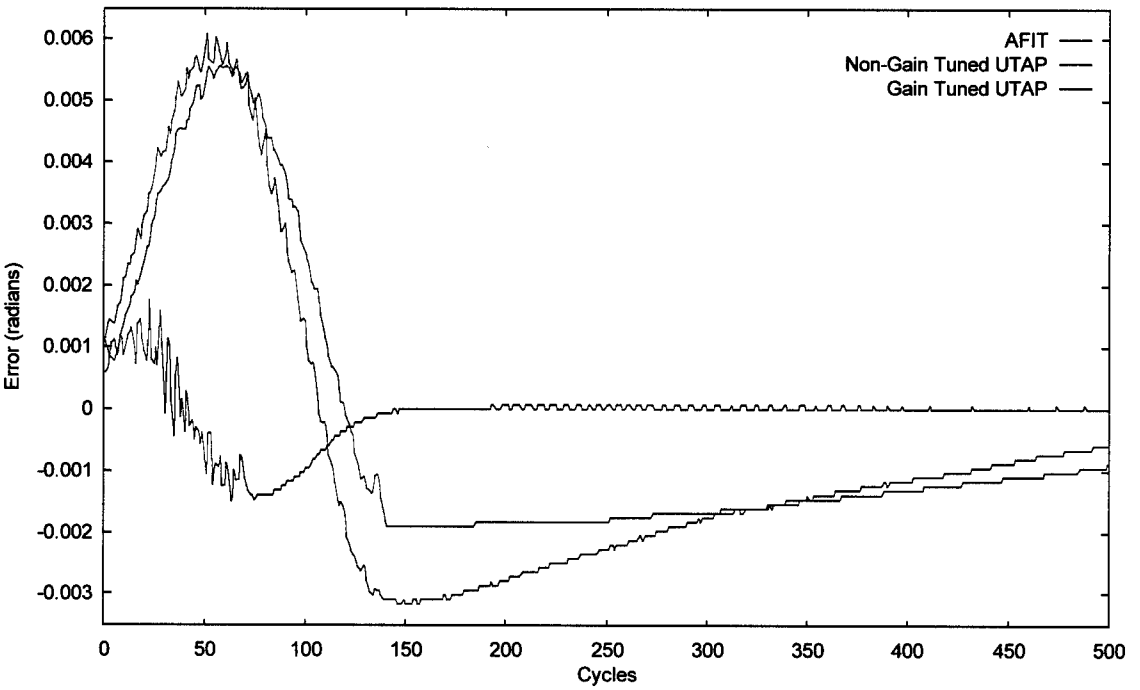


Figure E.2. Joint 2 Error for Nominal Trajectory

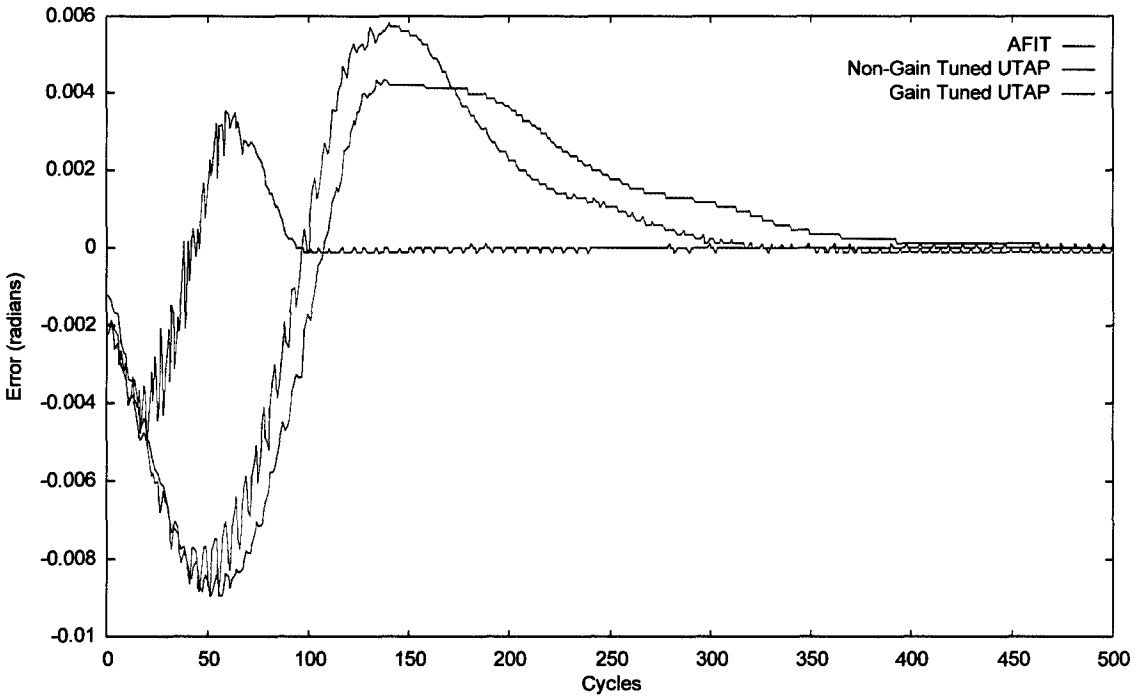


Figure E.3. Joint 3 Error for Nominal Trajectory

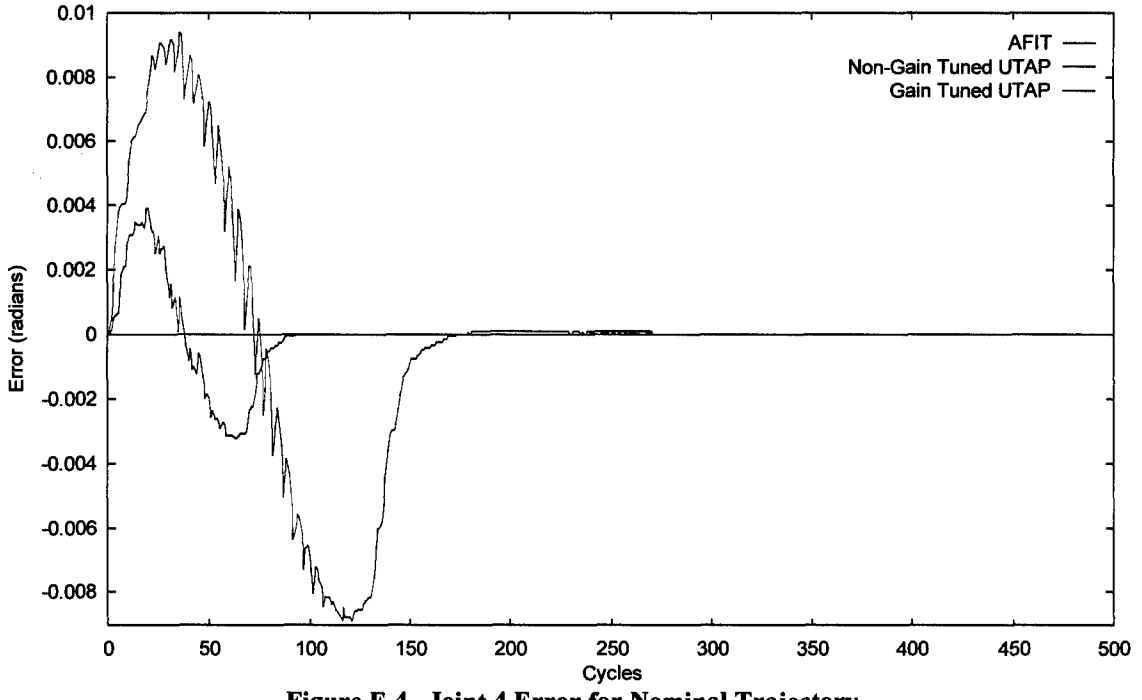


Figure E.4. Joint 4 Error for Nominal Trajectory

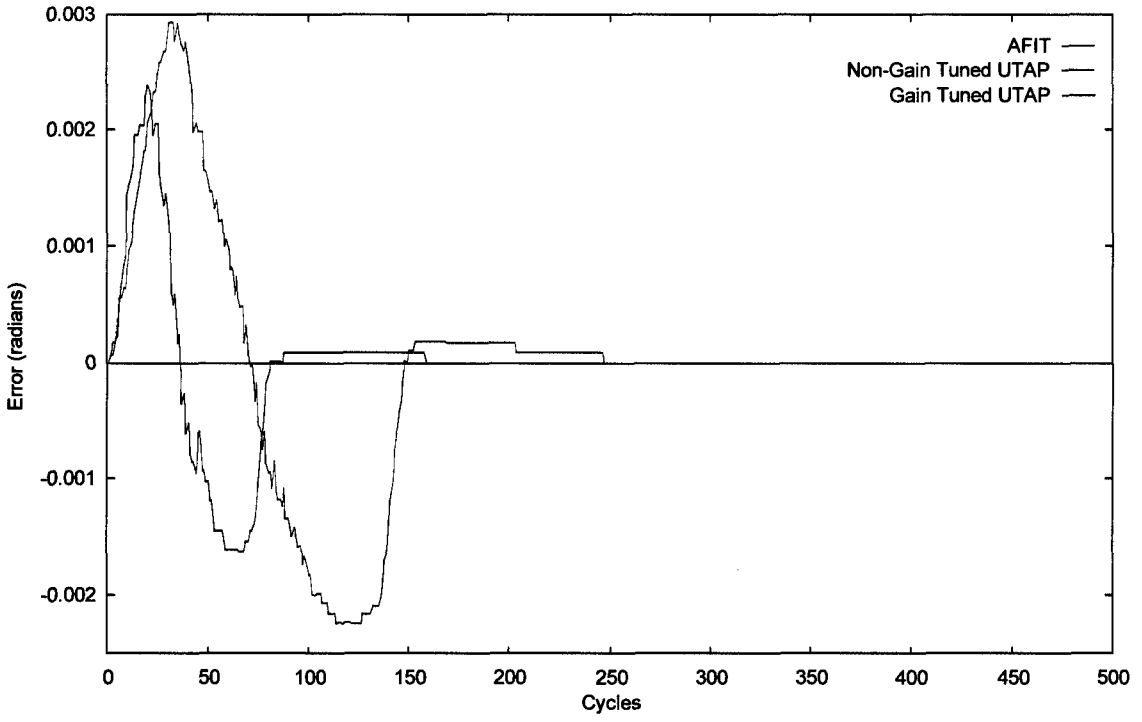


Figure E.5. Joint 5 Error for Nominal Trajectory

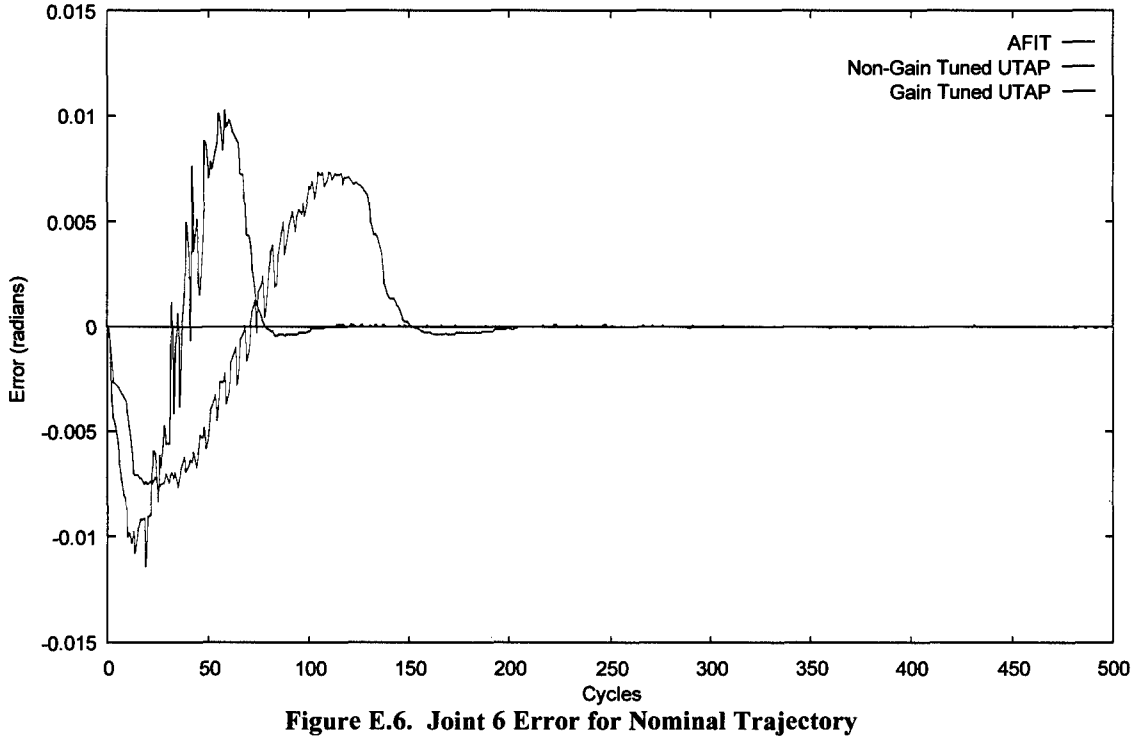


Figure E.6. Joint 6 Error for Nominal Trajectory

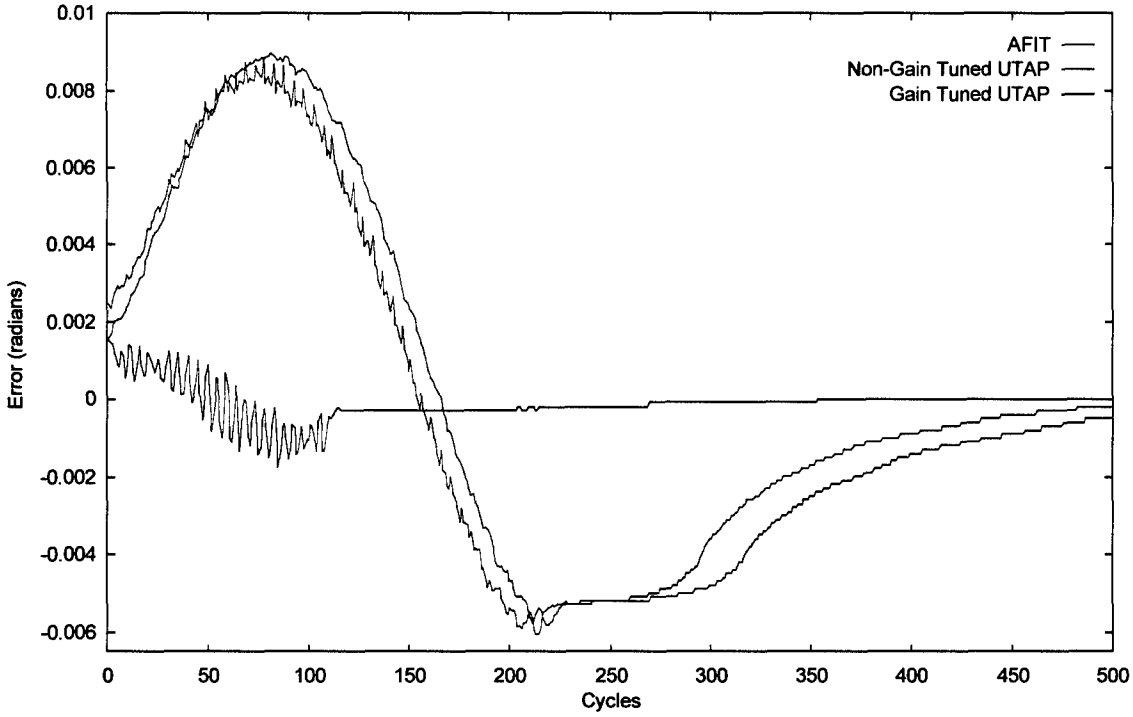


Figure E.7. Joint 1 Error for Slow Trajectory

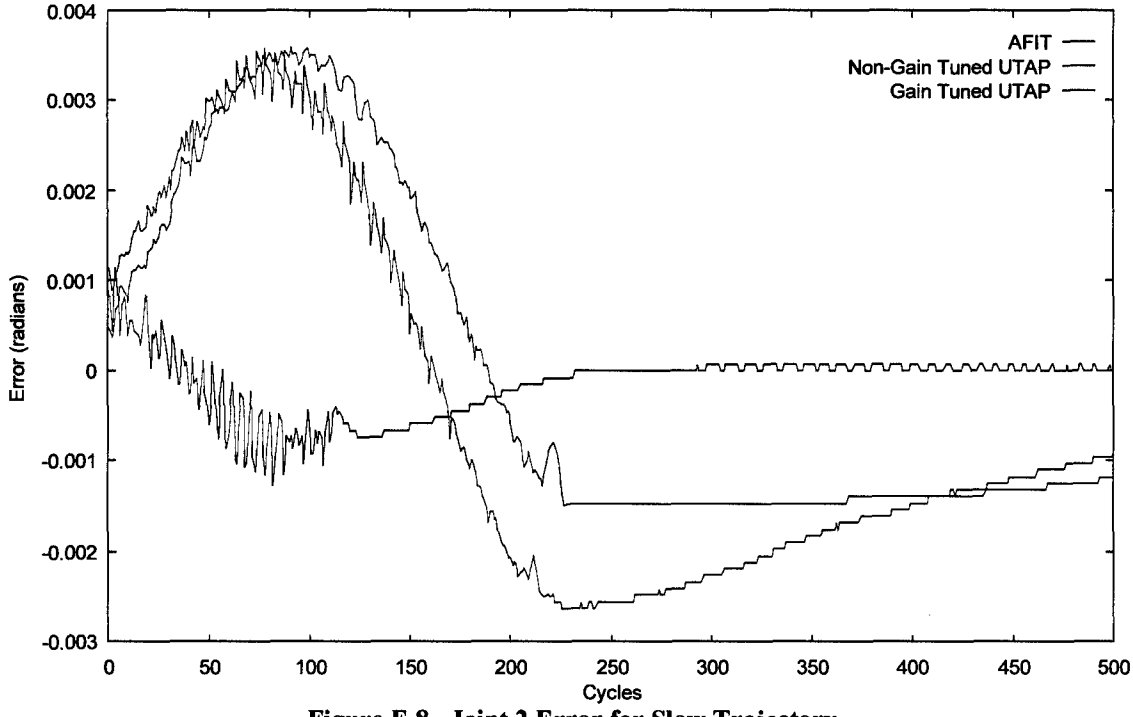


Figure E.8. Joint 2 Error for Slow Trajectory

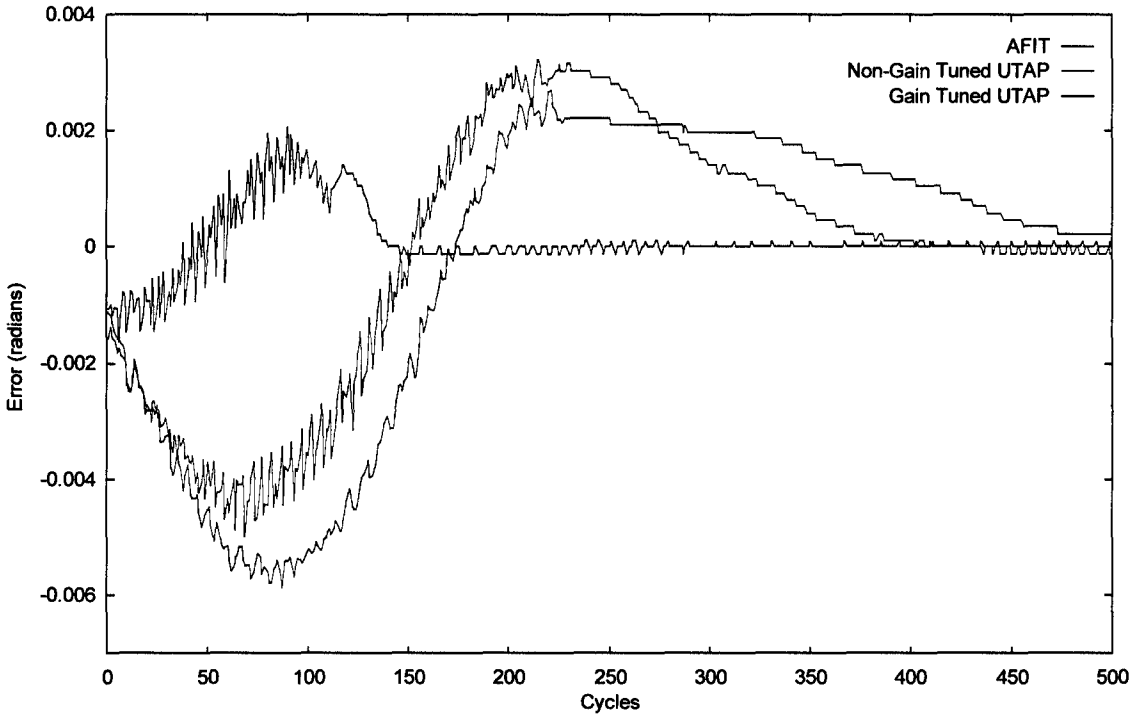


Figure E.9. Joint 3 Error for Slow Trajectory

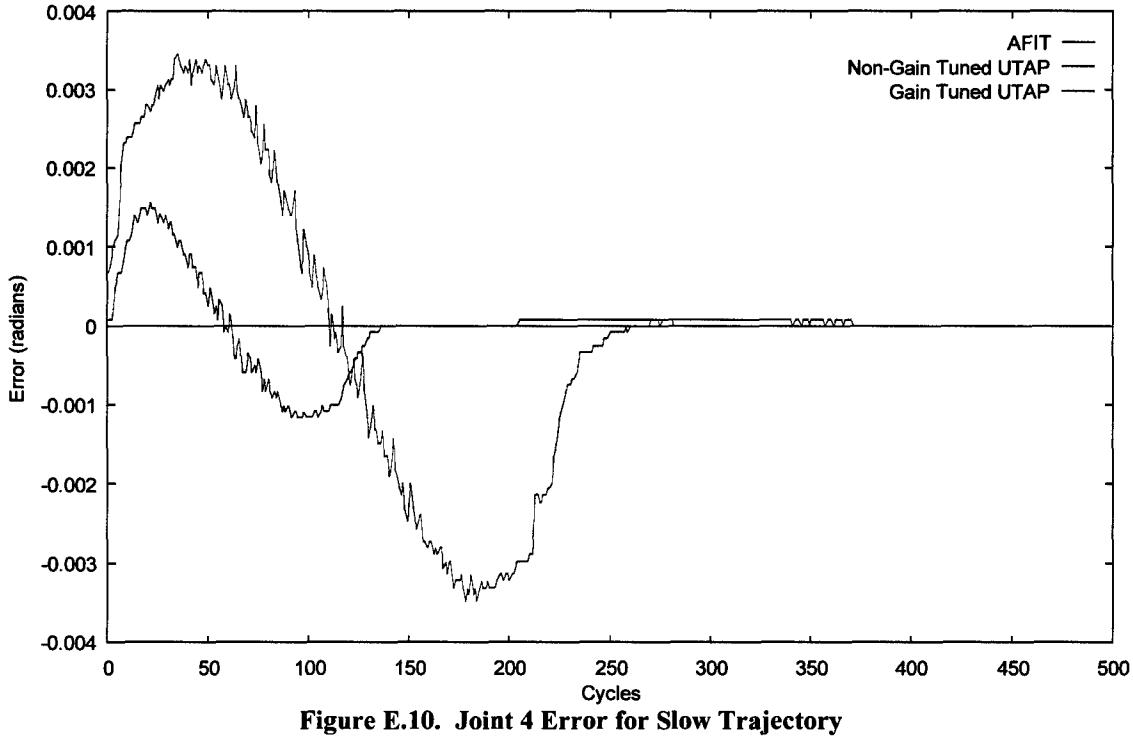


Figure E.10. Joint 4 Error for Slow Trajectory

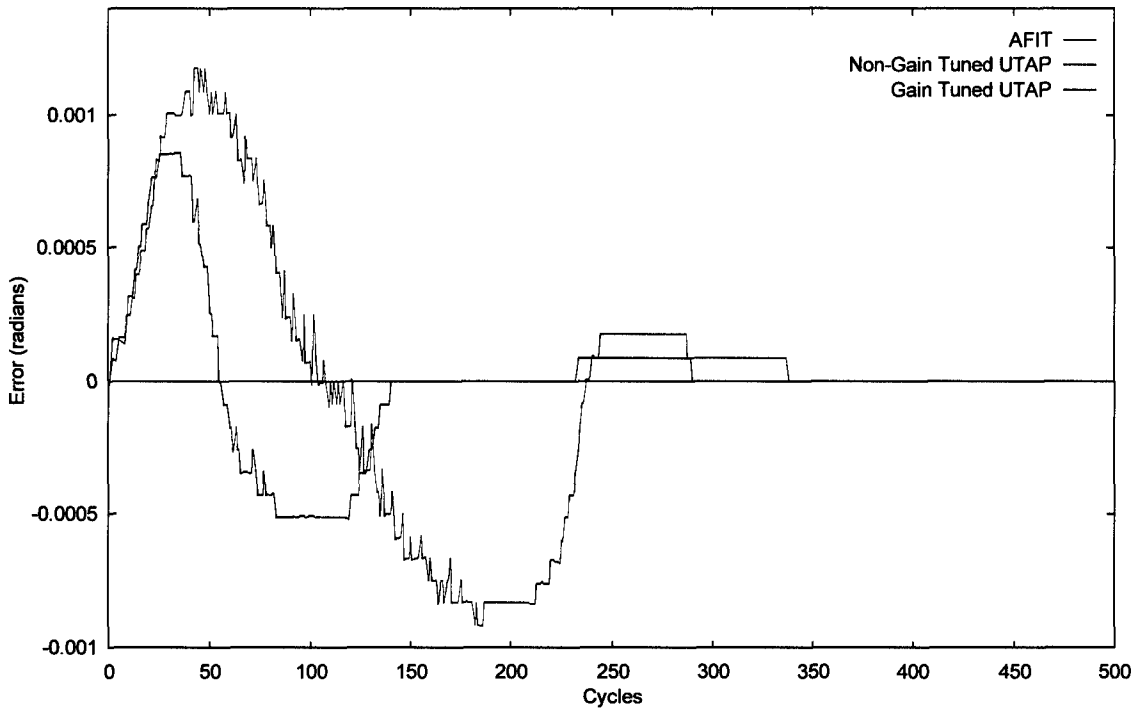


Figure E.11. Joint 5 Error for Slow Trajectory

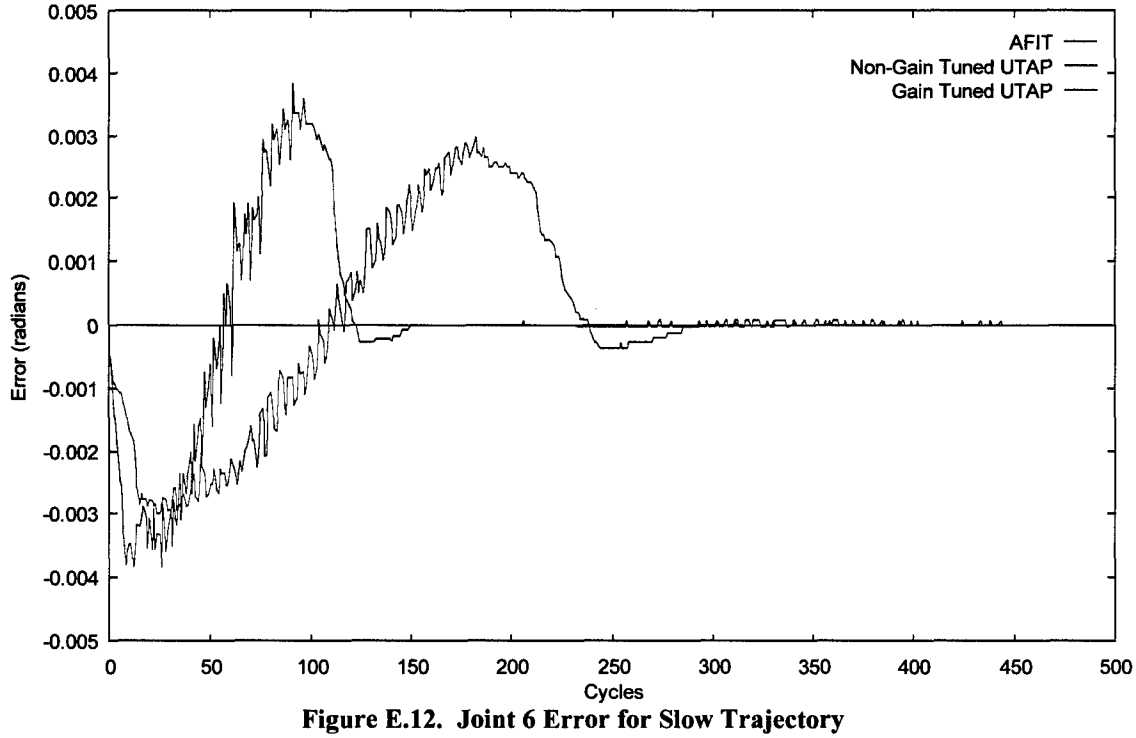


Figure E.12. Joint 6 Error for Slow Trajectory

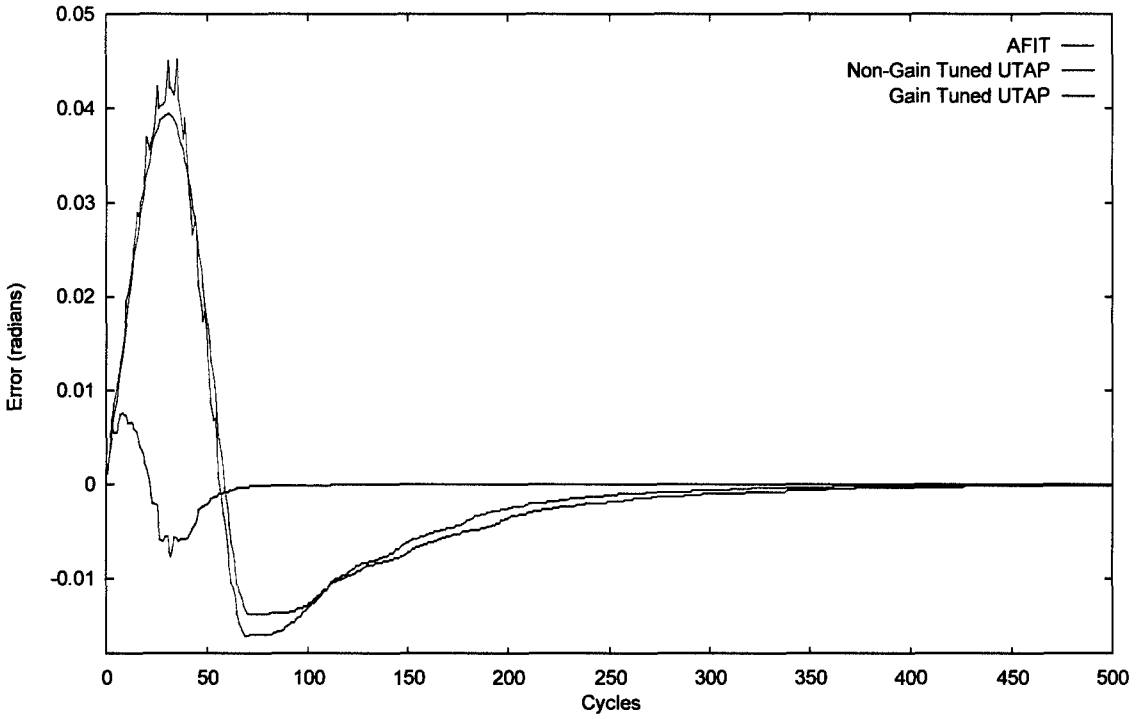


Figure E.13. Joint 1 Error for Fast Trajectory

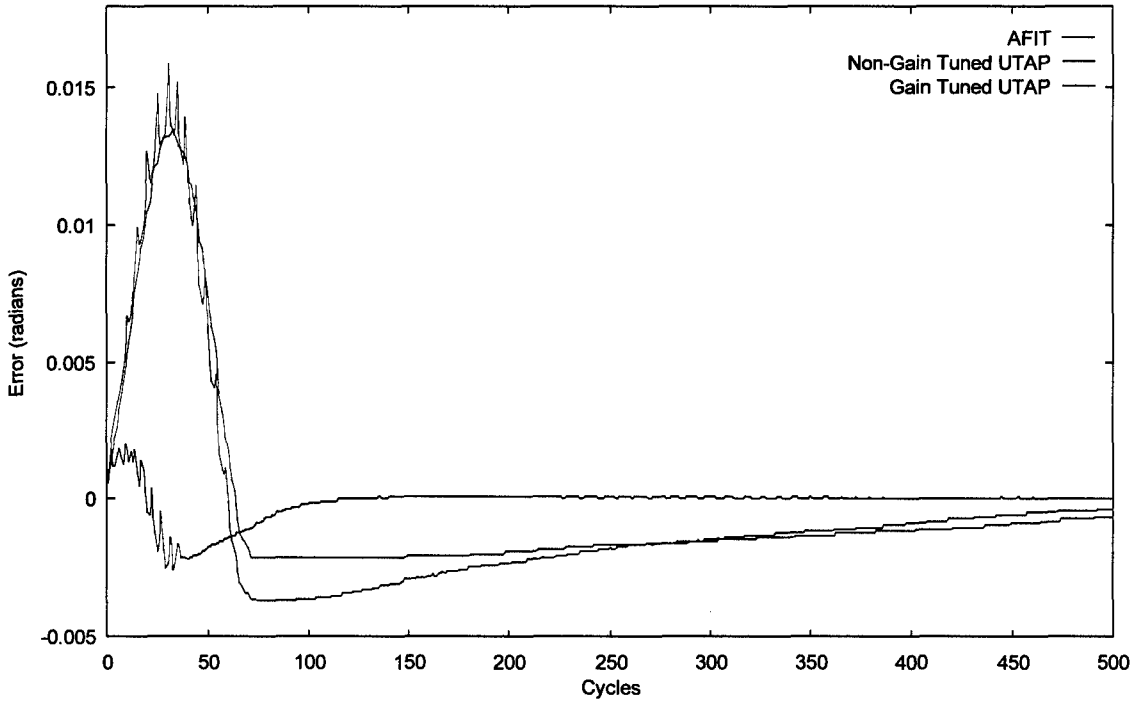


Figure E.14. Joint 2 Error for Fast Trajectory

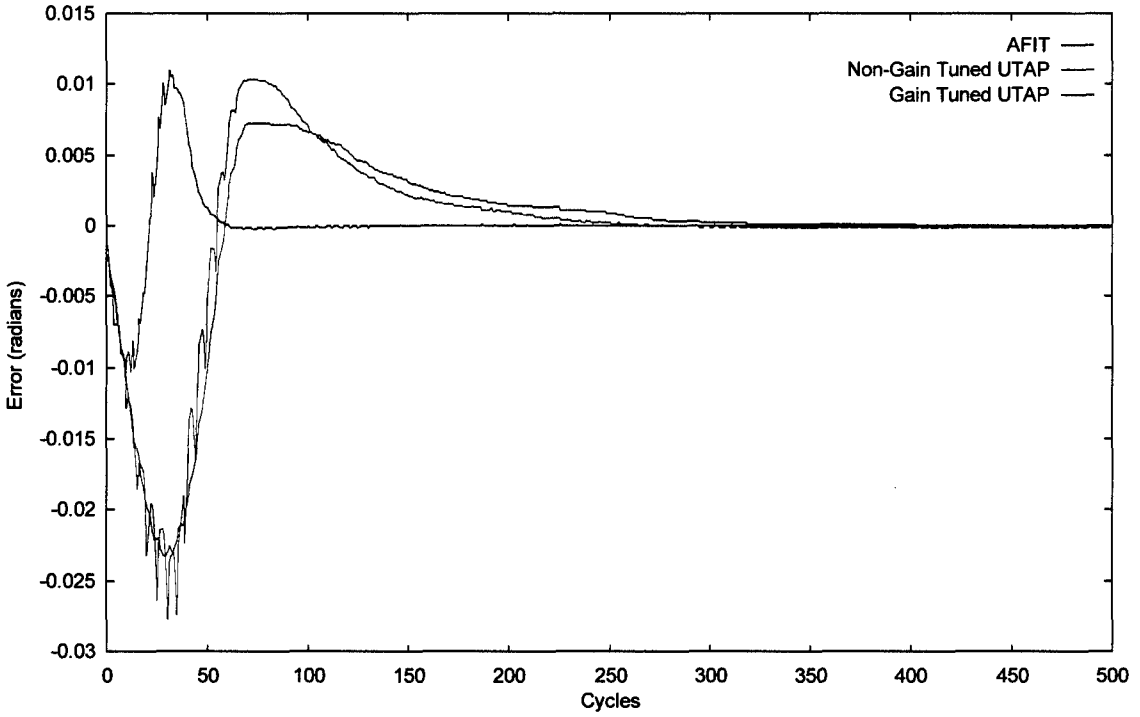


Figure E.15. Joint 3 Error for Fast Trajectory

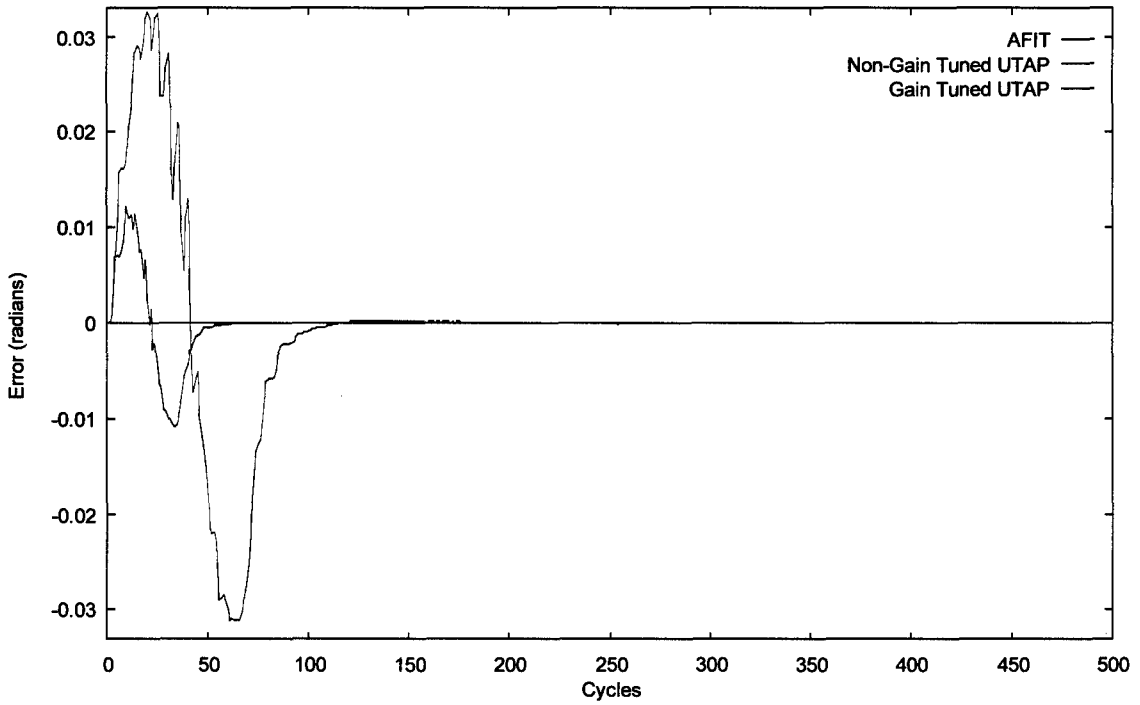


Figure E.16. Joint 4 Error for Fast Trajectory

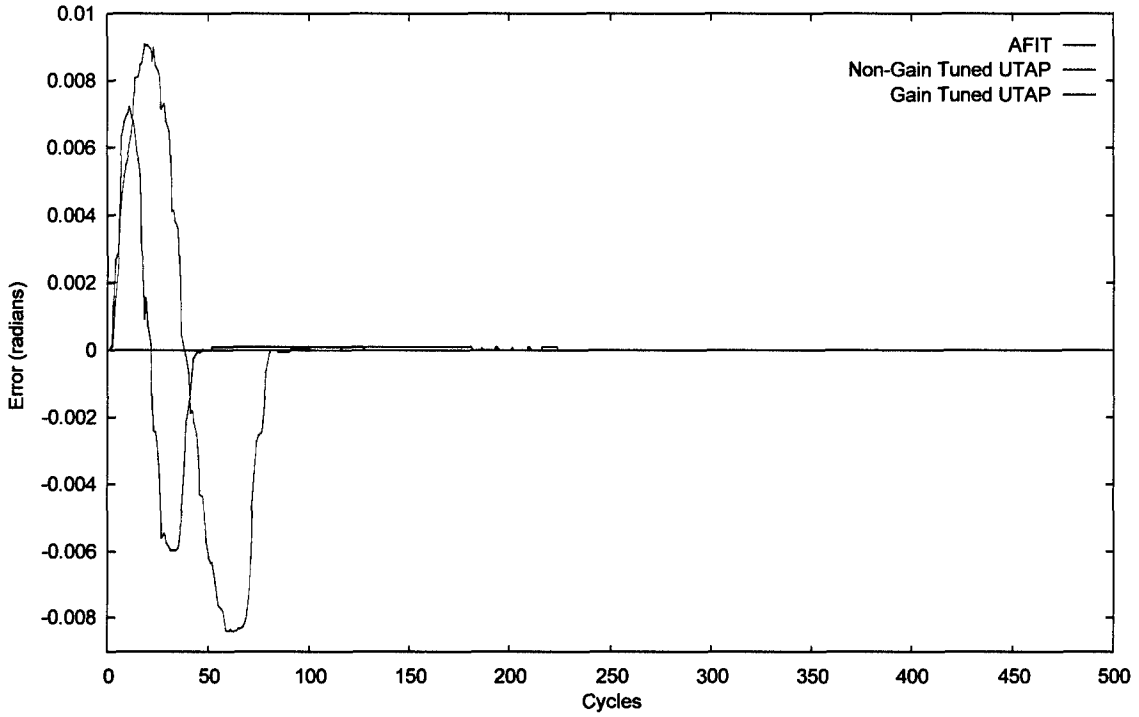


Figure E.17. Joint 5 Error for Fast Trajectory

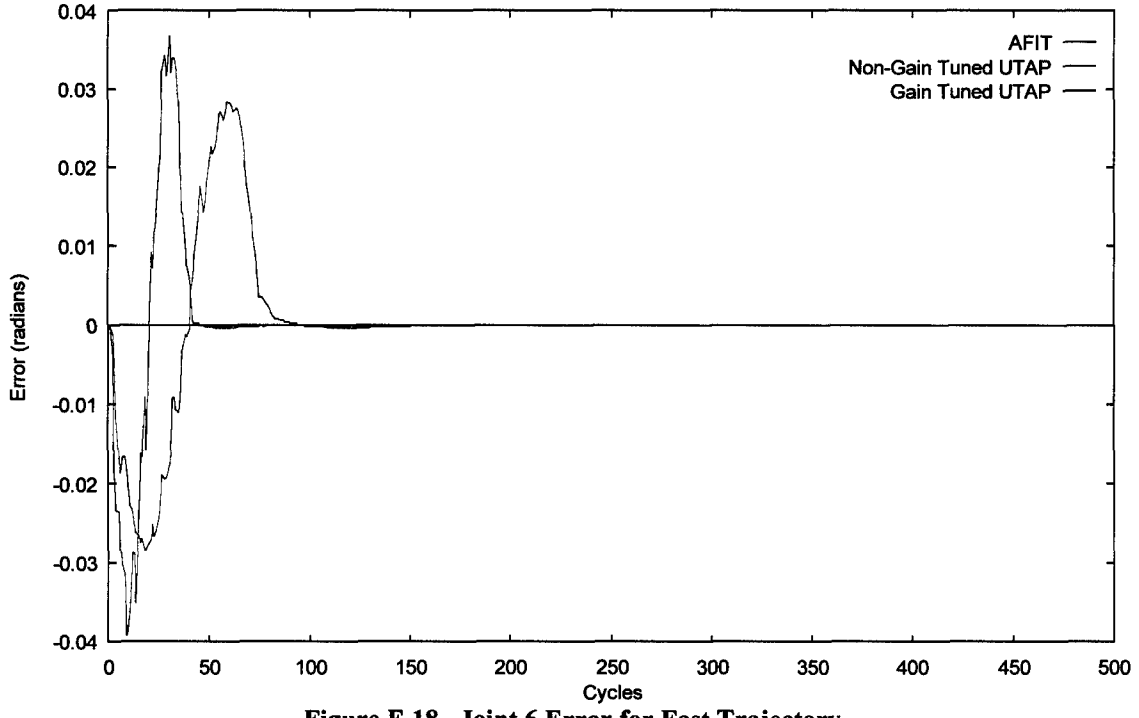


Figure E.18. Joint 6 Error for Fast Trajectory

APPENDIX F

TrjjgenCycle Function Source Code

Original TrjjgenCycle Code

```
/* ********************************************************************/
/*      trjjgenCycle                Process module information.       */
/* ********************************************************************/

int             trjjgenCycle(local, stask)
trjjgenLocal_t  *local;
sbsTask_t       *stask;
{
   int      flag, i, n = *(local->Ndof), type = local->type;
   float    *qmez = local->Qmez, *qref = local->Qref;
   double   nt, temp;
   double   *qdelta = local->qdelta, *qinit = local->qinit;


/* Calculate the next set of joint positions.                         */

   if ((nt = local->stepnum++ * local->stepsize) > 1.0)
      temp = 1.0;
   else
      temp = profilefunc(nt, type);

   for (i = 0; i < n; ++i)
      *(qref++) = (float) (*(qinit++) + temp * *(qdelta++));


/* See if it is time to stop.  Ideally, when nt == 1.0, the robot     */
/* has reached the desired position, but just to be sure, check the   */
/* actual position against the desired position before turning off.   */

   if (nt > 1.0)
   {
      qref = local->Qref;
      flag = 1;
      for (i = 0; i < n; ++i)
         if (fabs(*(qref++) - *(qmez++)) > QEPSILON)
            flag = 0;

      if (flag)
         return SBS_OFF;
   }


/* Return at the end of each cycle.                                   */

   return I_OK;
}
```

Modified TrjjgenCycle Code

```
/********************************************************************/
/*      trjjgenCycle                Process module information.       */
/*                                                                    */
/*      Modified to perform a step function - Capt. Matthew L. June   */
/*                                            GCS - 96D               */
/*                                                                    */
/* ********************************************************************/

int             trjjgenCycle(local, stask)
trjjgenLocal_t  *local;
sbsTask_t       *stask;
{
   int      flag, i, n = *(local->Ndof), type = local->type;
   float    *qmez = local->Qmez, *qref = local->Qref;
   double   nt, temp;
   double   *qdelta = local->qdelta, *qinit = local->qinit;

   for (i = 0; i < n; ++i)
      *(qref++) = (float) (*(qinit++) + *(qdelta++));

/* Return at the end of each cycle.                                   */

   return I_OK;
}
```

APPENDIX G

Original Palletizing Application V+ Source Code

```
.PROGRAM nonutap.demo()

;  Name:       nonutap.demo
;  Author:     Capt Matthew L. June - GCS-96D

;  Abstract:  This is the main program that starts the pallet, force1,
;  and force2 programs.  It prompts the user for number of rows and
;  columns on a pallet, the approach and depart height, and the number
;  of the pallet with parts on it.

        AUTO $rows, $cols, $ht, rows, cols, ht, $pal, pal, flun, $fname

        FINE ALWAYS

; Clear the screen

        TYPE /C50, /U25

; LUN for disk I/O

        flun = 5

; Attach to LUN

        ATTACH (flun) "disk"

; Prompt for audit trail file name

        PROMPT "Enter the file name for the audit trail (Include drive
               letter): ", $fname
        TYPE

; Open the file in append mode

        FOPENA (flun, 0, 0) $fname

; Prompt for the number of rows

    10  PROMPT "Enter the number of rows on the pallet: ", $rows
        rows = VAL($rows)
        IF ((rows < 1) OR (rows > 5)) GOTO 10

; Prompt for the number of columns

    20  PROMPT "Enter the number of columns on the pallet: ", $cols
        cols = VAL($cols)
        IF ((cols < 1) OR (cols > 5)) GOTO 20

; Prompt for the approach and depart height

        PROMPT "Enter an approach height: ", $ht
        ht = VAL($ht)

; Prompt for the pallet number with parts on it

    30  PROMPT "Enter the number of the pallet with parts on it: ", $pal
        pal = VAL($pal)
        IF ((pal < 1) OR (pal > 2)) GOTO 30

; Call the pallatizing program

        CALL pallet(rows, cols, pal, ht, flun)

        IF pal == 1 THEN
            pal = 2
        ELSE
            pal = 1
        END

; Call the first force program

        CALL force1(pal, rows, cols, flun)

; Call the second force program

        CALL force2(pal, flun)

; Close and detach file access

        FCLOSE (flun)
        DETACH (flun)

        RETURN
.END


.PROGRAM audit(pick, place, row, col, flun)

;  Name:       audit
;  Author:     Capt Matthew L. June - GCS-96D
;  Abstract:  This program writes pick and place location information
;  and row and column information to a disk file specified by a logical
;  unit number.

        AUTO pic[5], plac[5], i

; Get the six values from locations pick and place

        DECOMPOSE pic[] = pick
        DECOMPOSE plac[] = place

; Write the date, time, row, and col info to the audit file
```

```
        WRITE (flun) $TIME()
        WRITE (flun) "Row = ", row, "   Col = ", col

; Write the pick location info to the audit file

        WRITE (flun) "The pick location coordinates are:"
        WRITE (flun) "(", /S
        FOR i = 0 TO 5
            WRITE (flun) pic[i], /S
        END
        WRITE (flun) ")"

; Write the place location info to the audit file

        WRITE (flun) "The place location coordinates are:"
        WRITE (flun) "(", /S
        FOR i = 0 TO 5
            WRITE (flun) plac[i], /S
        END
        WRITE (flun) ")"
        WRITE (flun)

        RETURN
.END

.PROGRAM coords.info()

;  Name:      coords.info
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  When a software signal is activated, this program displays
;  the manipulator's current world coordinate system values to the
;  screen.

        AUTO curr.loc

; While the RC Task is running

        WHILE SIG(2100) DO

; Wait for a signal

            WAIT SIG(2102) OR SIG(-2100)

; If the signal was for coordinate display

            IF SIG(2102) THEN

                SET curr.loc = HERE

                TYPE
                TYPE "World Coordinate System X value: ", DX(curr.loc)
                TYPE "World Coordinate System Y value: ", DY(curr.loc)
                TYPE "World Coordinate System Z value: ", DZ(curr.loc)
                TYPE

            END

        END

        RETURN
.END

.PROGRAM display.info()

;  Name:      display.info
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  When a software signal is activated, this program displays
;  various system values to the screen.

; While the RC Task is running

        WHILE SIG(2100) DO

; Wait for a signal

            WAIT (SIG(2101) OR SIG(-2100))

; If it was the display signal

            IF SIG(2101) THEN

; Clear the screen

                TYPE /C50, /U25

                TYPE "Time since start of cycle: ", TIMER(1)
                TYPE
                TYPE "The speed settings are:"
                TYPE "Monitor   Perm   Curr   Next"
                TYPE " ", SPEED(1), "      ", SPEED(2), "   ",
                     SPEED(3), "   ", SPEED(4)
                TYPE

                TYPE "The gripper delay: ", PARAMETER(HAND.TIME)
                TYPE "The acceleration:  ", ACCEL(1)
                TYPE "The deceleration:  ", ACCEL(2)
                TYPE

                TYPE "Continuous path is ", /S
```

```
            IF SWITCH(CP) THEN
                TYPE "Enabled"
            ELSE
                TYPE "Disabled"
            END
            TYPE

            TYPE "Null / NoNull setting for next motion: ", /S
            IF CONFIG(3) BAND 4 THEN
                TYPE "NONULL"
            ELSE
                TYPE "NULL"
            END
            TYPE

            TYPE "Fine / Coarse setting for next motion: ", /S
            IF CONFIG(3) BAND 2 THEN
                TYPE "COARSE"
            ELSE
                TYPE "FINE"
            END

        END

    END

    RETURN
.END


.PROGRAM mcp.disp.menu()

;  Name:      mcp.disp.menu
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program writes a menu to the manual control pendant
;  display.

    AUTO mcp

    mcp = 1

; Clear the display and write the top line

    WRITE (mcp) $CHR(12), $CHR(18), $CHR(16), "MAIN MENU", /S

; Write the menu options

    WRITE (mcp) $CHR(18), $CHR(41), /S
    WRITE (mcp) "Cycle", $CHR(9), "Posit", $CHR(9), /S
    WRITE (mcp) "React", $CHR(9), "Reacti", $CHR(9), "   Stop", /S

    RETURN
.END


.PROGRAM mcp.main()

;  Name:      mcp.main
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program monitors the manual control pendant for input
;  and, based on the input, activates certain software signal or calls
;  other programs.

    AUTO quit, mcp

    mcp = 1
    quit = FALSE

    ATTACH (mcp)

    KEYMODE 1, 3 = 0        ; Keyboard mode
    KEYMODE 4, 5 = 1        ; Toggle mode

    WHILE SIG(2100) DO

        DO

            CALL mcp.disp.menu()

            CASE TRUE OF
              VALUE PENDANT(1):          ; Stop button

                WRITE (mcp) $CHR(31), $CHR(1), /S
                CALL mcp.main.quit(quit)
                WRITE (mcp) $CHR(28), $CHR(1), /S

              VALUE PENDANT(2):          ; Reacti button

                WRITE (mcp) $CHR(31), $CHR(2), /S
                SIGNAL 2004
                WAIT (SIG(-2004) OR SIG(-2100))
                WRITE (mcp) $CHR(28), $CHR(2), /S

              VALUE PENDANT(3):          ; React button

                WRITE (mcp) $CHR(31), $CHR(3), /S
                SIGNAL 2003
                WAIT (SIG(-2003) OR SIG(-2100))
                WRITE (mcp) $CHR(28), $CHR(3), /S

            END
```

```
            IF NOT quit THEN

                IF PENDANT(4) THEN       ; Posit button
                    SIGNAL 2102
                    WAIT (NOT PENDANT(4)) OR SIG(-2100)
                    SIGNAL -2102
                END

                IF PENDANT(5) THEN       ; Cycyle info button
                    SIGNAL 2101
                    WAIT (NOT PENDANT(5)) OR SIG(-2100)
                    SIGNAL -2101
                END

            END

        UNTIL quit OR SIG(-2100)

    END

; Detach the MCP

    DETACH (mcp)

    RETURN
.END


.PROGRAM mcp.main.quit(quit)

;  Name:      mcp.main.quit
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program displays the quit menu on the manual control
;  pendant and activates a software signal if the "yes" button is
;  pressed.

    AUTO button, mcp

    quit = FALSE
    mcp = 1

; Set softkeys 1 and 2 to keyboard mode

    KEYMODE 1, 2 = 0

; Display sub-menu and start the "NO" option blinking

    WRITE (mcp) $CHR(12), "Quit. Are you sure?"
    WRITE (mcp) $CHR(9), $CHR(9), $CHR(9), "  YES", /S
    WRITE (mcp) $CHR(9), $CHR(2), "   NO", $CHR(3), /S
    button = PENDANT(0)

; Signal 2005 and set quit to true if verified

    IF button == 2 THEN
        SIGNAL 2005        ; Signal to halt the robot
        quit = TRUE
    END

    RETURN
.END


.PROGRAM pallet(rows, cols, pal, ht, flun)

;  Name:      pallet
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This is the main program for the palletizing application.
;  It initializes the reactions to certain signals, launches the other
;  programs, and moves the parts from one pallet to another.

;  Signals used:
;
;  2100 - runsig
;  2101 - Cycle info display
;  2102 - Position info display
;  2003 - React, move to safe and resume exec when return is pressed
;  2004 - Reacti, move to safe and go back to where leftoff when
;         return is pressed
;  2005 - Reacti, Halt program

    AUTO $speed, speed, i, j

; Prompt for the operating speed

 10 PROMPT "Enter the speed for this operation: ", $speed
    speed = VAL($speed)
    IF ((speed < 1) OR (speed > 100)) GOTO 10

; set timer 2 for testing purposes

    TIMER 2 = 0

; Initialize signals

    SIGNAL -2101, -2102, -2003, -2004, -2005

; Signal that program is running

    RUNSIG 2100

; Setup reaction to signals 2003, 2004, and 2005
```

```
        REACT 2003, safe.resume
        REACTI 2004, safe.goback
        REACTI 2005, stop.prog

; Execute programs

        EXECUTE 3 mcp.main()

        EXECUTE 4 display.info()

        EXECUTE 5 coords.info()

; Establish two frames of reference for Pallet1 and Pallet2

        SET pal.1 = FRAME(loc1,loc2,loc3,loc1)
        SET pal.2 = FRAME(loc4,loc5,loc6,loc4)

        SPEED speed ALWAYS

        WRITE (flun) "START OF PALLETIZING INFORMATION"
        WRITE (flun)

        MOVE safe
        TIMER 1 = 0

        FOR i = 0 TO rows-1

            FOR j = 0 TO cols-1

                IF (pal == 1) THEN
                    CALL pnp(pal.1:TRANS(50*i,50*j),
                             pal.2:TRANS(50*i,50*j), ht)
                    CALL audit(pal.1:TRANS(50*i,50*j),
                               pal.2:TRANS(50*i,50*j), i+1, j+1, flun)
                ELSE
                    CALL pnp(pal.2:TRANS(50*i,50*j),
                             pal.1:TRANS(50*i,50*j), ht)
                    CALL audit(pal.2:TRANS(50*i,50*j),
                               pal.1:TRANS(50*i,50*j), i+1, j+1, flun)
                END

            END

        END

        MOVE safe

        RETURN
.END
```

```
.PROGRAM pnp(pick, place, ht)

;  Name:      pnp
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This is a simple pick and place program.  It is given the
;  pick and place locations and a height for approaches and departures.
;  It obtains a part from the pick location and deposits it at the place
;  location.

        AUTO grip.sig.open, grip.sig.close, grip.delay

; Set gripper signals

        grip.sig.open = -1
        grip.sig.close = 1
        grip.delay = 0.5

; move to the pick location

        SIGNAL grip.sig.open
        APPRO pick, ht
        MOVE pick
        BREAK
        SIGNAL grip.sig.close
        WAIT.EVENT 0, grip.delay

; move to a location clear of the pick location

        DEPART ht

; move to the place location

        APPRO place, ht
        MOVE place
        BREAK
        SIGNAL grip.sig.open
        WAIT.EVENT 0, grip.delay

; move to a location clear of the place location

        DEPART ht

        RETURN
.END


.PROGRAM safe.goback()

;  Name:      safe.goback
;  Author:    Capt Matthew L. June - GCS-96D
```

```
;  Abstract:  This program is used in conjuction with a reacti statement
;  in another program.  It moves the manipulator to a safe location.
;  When the user presses the return key, it moves back to the location
;  where is left off and continues the motion to the destination.

        AUTO left.off, going.to, $cont

; Save locations

        SET left.off = HERE
        SET going.to = DEST

; Move to safe location

        MOVE safe

; Clear the screen

        TYPE /C50, /U25

; Prompt to continue

        TYPE "Robot has been moved to a safe position"
        TYPE "Press return to go back to where the robot"
        PROMPT "left off and continue execution", $cont

; Move to where the robot stopped

        APPRO left.off, 20
        MOVE left.off

; Move to the interrupted destination

        MOVE going.to

; Clear the react signal

        SIGNAL -2004

; Reset the react signal

        REACTI 2004, safe.goback

        RETURN
.END


.PROGRAM safe.resume()

;  Name:      safe.resume
;  Author:    Capt Matthew L. June - GCS-96D
```

```
;  Abstract:  This program is used in conjuction with a react statement
;  in another program.  It moves the manipulator to a safe location.
;  When the user presses the return key, it moves the manipulator to
;  the destination.

        AUTO $cont

; Move to a safe position

        MOVE safe

; Clear the screen

        TYPE /C50, /U25

; Prompt to continue

        TYPE "Robot has been moved to a safe position"
        PROMPT "Press enter to continue", $cont

; Clear the react signal

        SIGNAL -2003

; Reinstate the react

        REACT 2003, safe.resume

        RETURN
.END


.PROGRAM stop.prog()

;  Name:      stop.prog
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program moves the manipulator to a safe location
;  and halts execution of the program.

        MOVE safe
        BREAK

        HALT

        RETURN
.END


.PROGRAM acquire.part(pallet, i, j)

;  Name:      acquire.part
```

```
;  Author:     Capt Matthew L. June - GCS-96D
;  Abstract:  This program moves the manipulator to a position
;  determined from the pallet number and i and j offsets.  Once there,
;  pick up a part and move to a position 75mm above the pick location.

        AUTO pick

        SPEED 100

        IF pallet == 1 THEN
            SET pick = pal.1:TRANS(50*i,50*j)
        ELSE
            SET pick = pal.2:TRANS(50*i,50*j)
        END

        SIGNAL -1
        WAIT.EVENT 0, 0.5

        APPRO pick, 75
        MOVE pick
        BREAK

        SIGNAL 1
        WAIT.EVENT 0, 0.5

        DEPART 75

        RETURN
.END


.PROGRAM force.init()

;  Name:       force.init
;  Author:     Capt Matthew L. June - GCS-96D
;  Abstract:  Initialize the force sensor to it's default state and
;  enable protect mode with the trip condition set to the Z-force
;  rating.


        AUTO ratings[5]

        FORCE.MODE (21)
        FORCE.READ (3) ratings[]
        FORCE.MODE (3) ratings[2]

        RETURN
.END


.PROGRAM force.moveg(goal.loc, zero_force, trip, forces[], trip.pos)

;  Name:       force.moveg
;  Author:     Capt Matthew L. June - GCS-96D
;  Abstract:  Enables guarded mode force sensing and attempts to move
;  to the goal location.  Returns trip status of the sensor, forces at
;  the time of the trip, and the trip location.

        IF zero_force THEN
            FORCE.OFFSET (1)
        END

        FORCE.MODE (1)

        MOVE goal.loc
        BREAK

        trip = LATCHED(1)
        IF trip THEN
            FORCE.READ (1) forces[]
            SET trip.pos = LATCH(1)
        END

        FORCE.MODE (-1)

        RETURN
.END


.PROGRAM force1(pallet, row, col, flun)

;  Name:       force1
;  Author:     Capt Matthew L. June - GCS-96D
;  Abstract:  Simulates a vertical insertion, where if the forces are
;  too high the inertion is jammed (force > 2 Lb).  A high-speed
;  approach to 100mm above the assembly location "goal1" is attempted
;  first, followed by a fast approach to 20mm above the insertion
;  location and a slow guarded move to the goal location.  Using a
;  second, low approach point minimizes the length of the relatively
;  slow guarded move.
;
;  The program acquires a part from a pallet and attempts to "insert"
;  it.  If the insertion fails, the part is returned to the pallet and
;  another part is tried.  If the insertion succeeds, the part is
;  released, the manipulator moves to a safe location, and the program
;  terminates.
;
;  The initial insertion location is below the table surface.  Therefore
;  a failure is guaranteed.  The insertion location is incremented up by
;  1mm for each iteration until the part is placed on the table surface.

        AUTO trip, trip.pos, offset, quit, i, j
```

```
        GLOBAL forces[]

; Initialize variables

        offset = -1
        quit = FALSE
        trip = FALSE
        i = 0
        j = 0

; Initialize force sensor and enable protected mode

        CALL force.init()

; Set force limits but don't enable guarded mode

        FORCE.MODE (-1) ^H12, 2

        MOVE safe
        BREAK

; Write initial lines to audit file and screen

        TYPE "START OF FORCE1 INFORMATION"
        TYPE
        TYPE "attempting first part"
        WRITE (flun) "START OF FORCE1 INFORMATION"
        WRITE (flun)
        WRITE (flun) "attempting first part"

; Start operation to 'insert' a part

        WHILE NOT quit DO

            CALL acquire.part(pallet, i, j)

            SPEED 100
            APPRO goal1:TRANS(0,0,-offset), 100
            BREAK

            SPEED 50
            APPRO goal1:TRANS(0,0,-offset), 20
            BREAK

            SPEED 5
            CALL force.moveg(goal1:TRANS(0,0,-offset), TRUE, trip,
                             forces[], trip.pos)
            BREAK

            IF trip THEN              ; guard force limit exceeded

                TYPE "Insertion jammed",
                       DISTANCE(goal1:TRANS(0,0,-offset),trip.pos),
                       " mm from goal"
                TYPE "Z direction force at time of jam: ", forces[2]
                TYPE
                WRITE (flun) "Insertion jammed",
                        DISTANCE(goal1:TRANS(0,0,-offset),trip.pos),
                        " mm from goal"
                WRITE (flun) "Z direction force at time of jam: ",
                             forces[2]
                WRITE (flun)

                DEPART 75
                CALL replace.part(pallet, i, j)       ; return the part

                offset = offset+1
                IF (i <= row-1) THEN           ; increment to next part

                    IF j < col-1 THEN
                        j = j+1
                    ELSE
                        i = i+1
                        j = 0
                    END  ; if-else

                END ; if

                IF i == row THEN              ; last part has been tried

                    TYPE "Last part failed, part cannot be inserted"
                    TYPE
                    WRITE (flun) "Last part failed, part cannot
                                  be inserted"
                    WRITE (flun)
                    MOVE safe
                    BREAK
                    quit = TRUE

                ELSE                        ; last part hasn't been tried

                    TYPE "Attempting next part"
                    WRITE (flun) "Attempting next part"

                END ; if-else

            ELSE             ; successful insert

                TYPE "Insertion succeeded."
                TYPE
                WRITE (flun) "Insertion succeeded."
                WRITE (flun)
```

```
                SIGNAL -1
                WAIT.EVENT 0, 0.5
                DEPART 80
                MOVE safe
                BREAK
                quit = TRUE

            END  ; if-else

        END ; while

        RETURN
.END


.PROGRAM force2(pallet, flun)

;  Name:       force2
;  Author:     Capt Matthew L. June - GCS-96D
;  Abstract:  This program acquires a part from a pallet and sets it on
;  the table surface.  It accomplishes this be moving the part to a
;  position above the surface and repeatedly moving the part down 1mm
;  until the Z direction force exceeds 2 Lbs.  At this point, the part
;  is released and the manipulator moves to a safe location

        AUTO trip, trip.pos, offset, i

; Initialize variables

        i = 1
        offset = 0
        trip = FALSE

; Initialize force sensor and enable protected mode

        CALL force.init()

; Set force limits but don't enable guarded mode

        FORCE.MODE (-1) ^H12, 2

        MOVE safe

        CALL acquire.part(pallet, 3, 1)

        SPEED 100
        APPRO goal2, 100
        BREAK

        SPEED 50
        APPRO goal2, 20
        BREAK

; Write initial lines to audit file

        TYPE "START OF FORCE2 INFORMATION"
        TYPE
        WRITE (flun) "START OF FORCE2 INFORMATION"
        WRITE (flun)

; Start the operation to place the part on the surface

        WHILE NOT trip DO

            SPEED 5
            CALL force.moveg(goal2:TRANS(0,0,-offset), TRUE, trip,
                             forces[], trip.pos)
            BREAK

            IF NOT trip THEN         ; part is not touching the surface

                TYPE "Failed attempt #", i
                WRITE (flun) "Failed attempt #", i
                offset = offset-1
                i = i+1

            ELSE                     ; part is touching the surface

                SIGNAL -1
                WAIT.EVENT 0, 0.5
                TYPE "Succeeded on attempt", i
                TYPE "Actual position is", DISTANCE(goal2,trip.pos),
                      " mm from the first attempt"
                TYPE
                WRITE (flun) "Succeeded on attempt", i
                WRITE (flun) "Actual position is",
                              DISTANCE(goal2,trip.pos),
                              " mm from the first attempt"
                WRITE (flun)
                DEPART 80
                MOVE safe
                BREAK

            END  ; if-else

        END ; while

        RETURN
.END
```

```
.PROGRAM replace.part(pallet, i, j)

;  Name:       replace.part
;  Author:     Capt Matthew L. June - GCS-96D
;  Abstract:  This program starts with a part in the gripper.  it moves
;  the part to a location specified by a pallet number and i and j
;  offsets.  Then, it releases the part and moves to a location 75mm
;  above the part.

        AUTO place

        SPEED 100

        IF pallet == 1 THEN
            SET place = pal.1:TRANS(50*i,50*j)
        ELSE
            SET place = pal.2:TRANS(50*i,50*j)
        END

        APPRO place, 75
        MOVE place
        BREAK

        SIGNAL -1
        WAIT.EVENT 0, 0.5

        DEPART 75

        RETURN
.END
```

APPENDIX H

UTAP-compliant Palletizing Application V+ Source Code

```
.PROGRAM utap.demo()

;  Name:       utap.demo
;  Author:     Capt Matthew L. June - GCS-96D
;  Abstract:  This is the UTAP-compliant main program that starts the
;  pallet, force1, and force2 programs.  It prompts the user for number
;  of rows and columns on a pallet, the approach and depart height, and
;  the number of the pallet with parts on it.

        AUTO $rows, $cols, $ht, $pal, rows, cols

        CALL init_ok()

        CALL tl_str_fine_mot("ALL")

; Clear the screen

        CALL post_ext_data(screen, "/C50, /U25")

; Enable append mode disk access

        CALL pi_enable(disk)
        CALL pi_set_mode(disk, "0,0")

; Prompt for the number of rows

    10  CALL get_ext_data(keyboard, "Enter the number of rows on the
                                    pallet: ", $rows)
        rows = VAL($rows)
        IF ((rows < 1) OR (rows > 5)) GOTO 10

; Prompt for the number of columns

    20  CALL get_ext_data(keyboard, "Enter the number of columns on the
                                    pallet: ", $cols)
        cols = VAL($cols)
        IF ((cols < 1) OR (cols > 5)) GOTO 20

; Prompt for the approach and depart height

        CALL get_ext_data(keyboard, "Enter an approach height: ", $ht)

; Prompt for the pallet number with parts on it

    30  CALL get_ext_data(keyboard, "Enter the number of the pallet with
                                    parts on it: ", $pal)
        IF (($pal <> "1") AND ($pal <> "2")) GOTO 30

; Call the pallatizing program

        CALL td_exec_prog("pallet("+$rows+", "+$cols+", "+$pal+",
                                                    "+$ht+")")

        IF $pal == "1" THEN
            $pal = "2"
        ELSE
            $pal = "1"
        END

; Call the first force program

        CALL td_exec_prog("force1("+$pal+", "+$rows+", "+$cols+")")

; Call the second force program

        CALL td_exec_prog("force2("+$pal+")")

; Close and detach file access

        CALL pi_disable(disk)

        RETURN
.END


.PROGRAM audit(row, col)

;  Name:       audit
;  Author:     Capt Matthew L. June - GCS-96D
;  Abstract:  This program writes pick and place location information
;  and row and column information to a disk file specified by a logical
;  unit number.

;  This program is UTAP-compliant

        AUTO pick, place, pic[5], plac[5], i

; Get the pick and place locations

        CALL get_ext_loc_dat("location_1", pick)
        CALL get_ext_loc_dat("location_2", place)

; Get the six values from locations pick and place

        DECOMPOSE pic[] = pick
        DECOMPOSE plac[] = place

; Write the date, time, row, and col info to the audit file

        CALL post_ext_data(disk, $TIME())
        CALL post_ext_data(disk, "Row = "+$ENCODE(row)+
                                 "    Col = "+$ENCODE(col))
```

```
; Write the pick location info to the audit file

        CALL post_ext_data(disk, "The pick location coordinates are:")
        CALL post_ext_data(disk, "(/S")
        FOR i = 0 TO 5
            CALL post_ext_data(disk, $ENCODE(pic[i])+"/S")
        END
        CALL post_ext_data(disk, ")")

; Write the place location info to the audit file

        CALL post_ext_data(disk, "The place location coordinates are:")
        CALL post_ext_data(disk, "(/S")
        FOR i = 0 TO 5
            CALL post_ext_data(disk, $ENCODE(plac[i])+"/S")
        END
        CALL post_ext_data(disk, ")")
        CALL post_ext_data(disk, "")

        RETURN
.END


.PROGRAM coords.info()

;  Name:      coords.info
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  When a software signal is activated, this program displays
;  the manipulator's current world coordinate system values to the
;  screen.

;  This program is UTAP-compliant

        AUTO curr.loc, running, coord_info

; While the RC Task is running

        CALL pi_bit_read(signal, 2100, running)
        WHILE running DO

; Wait for a signal

            CALL hold("signal 2102 -2100")

; If the signal was for coordinate display

            CALL pi_bit_read(signal, 2102, coord_info)
            IF coord_info THEN

                SET curr.loc = HERE

                CALL post_ext_data(screen, "")
                CALL post_ext_data(screen, "World Coordinate System X
                                            value:  /S")
                CALL post_ext_data(screen, $ENCODE(DX(curr.loc)))
                CALL post_ext_data(screen, "World Coordinate System Y
                                            value:  /S")
                CALL post_ext_data(screen, $ENCODE(DY(curr.loc)))
                CALL post_ext_data(screen, "World Coordinate System Z
                                            value:  /S")
                CALL post_ext_data(screen, $ENCODE(DZ(curr.loc)))
                CALL post_ext_data(screen, "")

            END
            CALL pi_bit_read(signal, 2100, running)

        END

        RETURN
.END


.PROGRAM display.info()

;  Name:      display.info
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  When a software signal is activated, this program displays
;  various system values to the screen.

;  This program is UTAP-compliant

        AUTO running, disp_info, cycle_time, $data

; While the RC Task is running

        CALL pi_bit_read(signal, 2100, running)
        WHILE running DO

; Wait for a signal

            CALL hold("signal 2101 -2100")

; If it was the display signal

            CALL pi_bit_read(signal, 2101, disp_info)
            IF disp_info THEN

; Clear the screen

                CALL post_ext_data(screen, "/C50, /U25")
```

```
            CALL pi_bit_read(timer, 1, cycle_time)
            CALL post_ext_data(screen, "Time since start
                               of cycle: "+$ENCODE(cycle_time))
            CALL post_ext_data(screen, "")
            CALL post_ext_data(screen, "The speed settings are:")
            CALL post_ext_data(screen,
                               "Monitor   Perm   Curr   Next")
            CALL ok_attrib_query("Monitor Speed", $data)
            CALL post_ext_data(screen, " "+$data+" /S")
            CALL ok_attrib_query("Permanent Speed", $data)
            CALL post_ext_data(screen, "    "+$data+" /S")
            CALL ok_attrib_query("Current Speed", $data)
            CALL post_ext_data(screen, "  "+$data+" /S")
            CALL ok_attrib_query("Next Speed", $data)
            CALL post_ext_data(screen, "  "+$data)
            CALL post_ext_data(screen, "")

            CALL ok_attrib_query("Gripper Delay", $data)
            CALL post_ext_data(screen, "The gripper delay: "+$data)
            CALL ok_attrib_query("Acceleration", $data)
            CALL post_ext_data(screen, "The acceleration:  "+$data)
            CALL ok_attrib_query("Deceleration", $data)
            CALL post_ext_data(screen, "The deceleration:  "+$data)
            CALL post_ext_data(screen, "")

            CALL ok_attrib_query("Continuous Path", $data)
            CALL post_ext_data(screen, "Continuous path is "+$data)
            CALL post_ext_data(screen, "")

            CALL ok_attrib_query("Null-NoNull", $data)
            CALL post_ext_data(screen, "Null - NoNull setting for
                                        next motion:  "+$data)
            CALL post_ext_data(screen, "")

            CALL ok_attrib_query("Fine-Coarse", $data)
            CALL post_ext_data(screen, "Fine - Coarse setting for
                                        next motion:  "+$data)

        END
        CALL pi_bit_read(signal, 2100, running)

    END

    RETURN
.END


.PROGRAM mcp.disp.menu()

;  Name:      mcp.disp.menu
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program writes a menu to the manual control pendant
;  display.

;  This program is UTAP-compliant

; Clear the display and write the top line

        CALL post_ext_data(mcp, "$CHR(12),$CHR(18),$CHR(16),MAIN MENU")

; Write the menu options

        CALL post_ext_data(mcp, "$CHR(18),$CHR(41)")
        CALL post_ext_data(mcp, "Cycle,$CHR(9), Posit,$CHR(9)")
        CALL post_ext_data(mcp, " React,$CHR(9), Reacti,$CHR(9),
                                 Stop")

        RETURN
.END


.PROGRAM mcp.main()

;  Name:      mcp.main
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program monitors the manual control pendant for input
;  and, based on the input, activates certain software signal or calls
;  other programs.

;  This program is UTAP-compliant

        AUTO running, button_value

        quit = FALSE
        CALL pi_bit_read(signal, 2100, running)

        CALL pi_enable(mcp)

        CALL pi_bit_set(mcp, 1, 0)
        CALL pi_bit_set(mcp, 2, 0)
        CALL pi_bit_set(mcp, 3, 0)
        CALL pi_bit_set(mcp, 4, 1)
        CALL pi_bit_set(mcp, 5, 1)

        WHILE running DO

            DO

                CALL td_exec_prog("mcp.disp.menu()")

                CALL pi_bit_read(mcp, 1, button_value)     ;Stop button
                IF button_value THEN
```

```
            CALL post_ext_data(mcp, "$CHR(31), $CHR(1)")
            CALL td_exec_prog("mcp.main.quit()")
            CALL post_ext_data(mcp, "$CHR(28), $CHR(1)")
        END

        CALL pi_bit_read(mcp, 2, button_value)    ;Reacti button
        IF button_value THEN
            CALL post_ext_data(mcp, "$CHR(31), $CHR(2)")
            CALL pi_bit_set(signal, 2004, 1)
            CALL hold("signal -2004 -2100")
            CALL post_ext_data(mcp, "$CHR(28), $CHR(2)")
        END

        CALL pi_bit_read(mcp, 3, button_value)     ;React button
        IF button_value THEN
            CALL post_ext_data(mcp, "$CHR(31), $CHR(3)")
            CALL pi_bit_set(signal, 2003, 1)
            CALL hold("signal -2003 -2100")
            CALL post_ext_data(mcp, "$CHR(28), $CHR(3)")
        END

        IF NOT quit THEN

            CALL pi_bit_read(mcp, 4, button_value)
            IF button_value THEN                  ;Posit button
                CALL pi_bit_set(signal, 2102, 1)
                CALL hold("signal -2100 pendant -4")
                CALL pi_bit_set(signal, 2102, -1)
            END

            CALL pi_bit_read(mcp, 5, button_value)
            IF button_value THEN              ;Cycle info button
                CALL pi_bit_set(signal, 2101, 1)
                CALL hold("signal -2100 pendant -5")
                CALL pi_bit_set(signal, 2101, -1)
            END

        END
        CALL pi_bit_read(signal, 2100, running)

    UNTIL quit OR (NOT running)

END

; Detach the MCP

    CALL pi_disable(mcp)

    RETURN
.END
```

```
.PROGRAM mcp.main.quit()

;  Name:      mcp.main.quit
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program displays the quit menu on the manual control
;  pendant and activates a software signal if the "yes" button is
;  pressed.

;  This program is UTAP-compliant

        AUTO button

; Set softkeys 1 and 2 to keyboard mode

        CALL pi_bit_set(mcp, 1, 0)
        CALL pi_bit_set(mcp, 2, 0)

; Display sub-menu and start the "NO" option blinking

        CALL post_ext_data(mcp, "$CHR(12),Quit. Are you sure?")
        CALL post_ext_data(mcp, "$CHR(18),$CHR(41)")
        CALL post_ext_data(mcp, "$CHR(9),$CHR(9),$CHR(9),  YES")
        CALL post_ext_data(mcp, "$CHR(9),$CHR(2),   NO, $CHR(3)")
        CALL pi_bit_read(mcp, 0, button)

; Signal 2005 and set quit to true if verified

        IF button == 2 THEN
            CALL pi_bit_set(signal, 2005, 1)  ; Signal to halt the robot
            quit = TRUE
        END

        RETURN
.END
```

```
.PROGRAM pallet(rows, cols, pal, ht)

;  Name:      pallet
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This is the main program for the palletizing application.
;  It initializes the reactions to certain signals, launches the other
;  programs, and moves the parts from one pallet to another.

;  This program is UTAP-compliant

;  Signals used:
;
;  2100 - runsig
;  2101 - Cycle info display
```

```
;  2102 - Position info display
;  2003 - React, move to #safe and resume exec when return is pressed
;  2004 - Reacti, move to #safe and go back to where leftoff when
;         return is pressed
;  2005 - Reacti, Halt program

        AUTO $speed, speed, i, j

; Prompt for the operating speed

    10  CALL get_ext_data(keyboard, "Enter the speed for this
                                    operation: ", $speed)
        speed = VAL($speed)
        IF ((speed < 1) OR (speed > 100)) GOTO 10

; Set timer 2 for testing purposes

        TIMER 2 = 0

; Initialize signals

        CALL pi_bit_set(signal, 2101, -1)
        CALL pi_bit_set(signal, 2102, -1)
        CALL pi_bit_set(signal, 2003, -1)
        CALL pi_bit_set(signal, 2004, -1)
        CALL pi_bit_set(signal, 2005, -1)

; Signal that program is running

        CALL pi_set_mode(signal, "RUNSIG 2100")

; Setup reaction to signals 2003, 2004, and 2005

        CALL pi_set_mode(signal, "REACT 2003, safe.resume")
        CALL pi_set_mode(signal, "REACTI 2004, safe.goback")
        CALL pi_set_mode(signal, "REACTI 2005, stop.prog")

; Execute programs

        CALL ps_select_agent(3, "mcp.main()")

        CALL ps_select_agent(4, "display.info()")

        CALL ps_select_agent(5, "coords.info()")

; Establish two frames of reference for Pallet1 and Pallet2

        SET pal.1 = FRAME(loc1,loc2,loc3,loc1)
        SET pal.2 = FRAME(loc4,loc5,loc6,loc4)

        CALL as_set_velocity(speed)

        CALL post_ext_data(disk, "START OF PALLETIZING INFORMATION")
        CALL post_ext_data(disk, "")

        CALL as_set_position(safe)
        CALL pi_bit_set(timer, 1, 0)

        FOR i = 0 TO rows-1

            FOR j = 0 TO cols-1

                IF (pal == 1) THEN
                    CALL set_ext_loc_dat("location_1",
                                         pal.1:TRANS(50*i,50*j))
                    CALL set_ext_loc_dat("location_2",
                                         pal.2:TRANS(50*i,50*j))
                ELSE
                    CALL set_ext_loc_dat("location_1",
                                         pal.2:TRANS(50*i,50*j))
                    CALL set_ext_loc_dat("location_2",
                                         pal.1:TRANS(50*i,50*j))
                END
                CALL td_exec_prog("pnp("+$ENCODE(ht)+")")
                CALL td_exec_prog("audit("+$ENCODE(i)+","
                                         +$ENCODE(j)+")")

            END

        END

; Move to a safe position

        CALL as_set_position(safe)

        RETURN
.END


.PROGRAM pnp(ht)

;  Name:      pnp
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This is a simple pick and place program.  It is given the
;  pick and place locations and a height for approaches and departures.
;  It obtains a part from the pick location and deposits it at the place
;  location.

;  This program is UTAP-compliant

        AUTO pick, place
```

```
; get pick and place locations

        CALL get_ext_loc_dat("location_1", pick)
        CALL get_ext_loc_dat("location_2", place)

; move to the pick location

        CALL gripper_open()
        CALL as_set_position(pick:TRANS(0,0,-ht))
        CALL as_set_position(pick)
        BREAK
        CALL gripper_close()

; move to a location clear of the pick location

        CALL as_set_position(pick:TRANS(0,0,-ht))

; move to the place location

        CALL as_set_position(place:TRANS(0,0,-ht))
        CALL as_set_position(place)
        BREAK
        CALL gripper_open()

; move to a location clear of the place location

        CALL as_set_position(place:TRANS(0,0,-ht))

        RETURN
.END


.PROGRAM safe.goback()

;  Name:      safe.goback
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program is used in conjuction with a reacti statement
;  in another program.  It moves the manipulator to a safe location.
;  When the user presses the return key, it moves back to the location
;  where is left off and continues the motion to the destination.

;  This program is UTAP-compliant

        AUTO left.off, going.to, $cont

; Save locations

        SET left.off = HERE
        SET going.to = DEST

; Move to safe location

        CALL as_set_position(safe)

; Clear the screen

        CALL post_ext_data(screen, "/C50, /U25")

; Prompt to continue

        CALL post_ext_data(screen, "Robot has been moved to a
                                    safe position")
        CALL post_ext_data(screen, "Press return to go back to
                                    where the robot")
        CALL get_ext_data(keyboard, "left off and continue execution",
                                    $cont)

; Move to where the robot stopped

        CALL as_set_position(left.off:TRANS(0,0,-20))
        CALL as_set_position(left.off)

; Move to the interrupted destination

        CALL as_set_position(going.to)

; Clear the react signal

        CALL pi_bit_set(signal, 2004, -1)

; Reset the react signal

        CALL pi_set_mode(signal, "REACTI 2004, safe.goback")

        RETURN
.END


.PROGRAM safe.resume()

;  Name:      safe.resume
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program is used in conjuction with a react statement
;  in another program.  It moves the manipulator to a safe location.
;  When the user presses the return key, it moves the manipulator to
;  the destination.

;  This program is UTAP-compliant

        LOCAL $cont

; Move to a safe position
```

```
        CALL as_set_position(safe)

; Clear the screen

        CALL post_ext_data(2, "/C50, /U25")

; Prompt to continue

        CALL post_ext_data(screen, "Robot has been moved to a
                                    safe position")
        CALL get_ext_data(keyboard, "Press enter to continue", $cont)

; Clear the react signal

        CALL pi_bit_set(signal, 2003, -1)

; Reinstate the react

        CALL pi_set_mode(signal, "REACT 2003, safe.resume")

        RETURN
.END


.PROGRAM stop.prog()

;  Name:      stop.prog
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program moves the manipulator to a safe location
;  and halts execution of the program.

;  This program is UTAP-compliant

        CALL as_set_position(safe)
        BREAK

        HALT

        RETURN
.END


.PROGRAM acquire.part(pallet, i, j)

;  Name:      acquire.part
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program moves the manipulator to a position
;  determined from the pallet number and i and j offsets.  Once there,
;  pick up a part and move to a position 75mm above the pick location.

;  This program is UTAP-compliant

        AUTO pick

        CALL as_set_velocity(100)

        IF pallet == 1 THEN
            SET pick = pal.1:TRANS(50*i,50*j)
        ELSE
            SET pick = pal.2:TRANS(50*i,50*j)
        END

        CALL gripper_open()

        CALL as_set_position(pick:TRANS(0,0,-75))
        CALL as_set_position(pick)
        BREAK

        CALL gripper_close()

        CALL as_set_position(pick:TRANS(0,0,-75))

        RETURN
.END


.PROGRAM force.init()

;  Name:      force.init
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  Initialize the force sensor to it's default state and
;  enable protect mode with the trip condition set to the Z-force
;  rating.

;  This program is UTAP-compliant

        AUTO ratings[5]

        CALL ft_sc_mode_sel(21, "")
        CALL sc_get_att_read("ratings", ratings[])
        CALL ft_sc_mode_sel(3, $ENCODE(ratings[2]))
        CALL ft_sc_enable(3)

        RETURN
.END


.PROGRAM force.moveg(zero_force)

;  Name:      force.moveg
;  Author:    Capt Matthew L. June - GCS-96D
```

```
;  Abstract:  Enables guarded mode force sensing and attempts to move
;  to the goal location.  Returns trip status of the sensor, forces at
;  the time of the trip, and the trip location.

;  This program is UTAP-compliant

        AUTO LOC goal.loc

        CALL get_ext_loc_dat("location_1", goal.loc)

        IF zero_force THEN
            CALL sc_get_att_read("offset")
        END

        CALL ft_sc_enable(1)

        CALL as_set_position(goal.loc)
        BREAK

        CALL sc_get_reading("trip")          ;   trip = LATCHED(1)
        IF trip THEN
            CALL sc_get_reading("forces")    ;   FORCE.READ (1) forces[]
            CALL sc_get_reading("trip.pos")  ;   SET trip.pos = LATCH(1)
        END

        CALL ft_sc_disable(1)

        RETURN
.END


.PROGRAM force1(pallet, row, col)

;  Name:      force1
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  Simulates a vertical insertion, where if the forces are
;  too high the inertion is jammed (force > 2 Lb).  A high-speed
;  approach to 100mm above the assembly location "goal1" is attempted
;  first, followed by a fast approach to 20mm above the insertion
;  location and a slow guarded move to the goal location.  Using a
;  second, low approach point minimizes the length of the relatively
;  slow guarded move.
;
;  The program acquires a part from a pallet and attempts to "insert"
;  it.  If the insertion fails, the part is returned to the pallet and
;  another part is tried.  If the insertion succeeds, the part is
;  released, the manipulator moves to a safe location, and the program
;  terminates.
;
;  The initial insertion location is below the table surface.  Therefore
;  a failure is guaranteed.  The insertion location is incremented up by
;  1mm for each iteration until the part is placed on the table surface.

;  This program is UTAP-compliant

        AUTO offset, quit, i, j

; Initialize variables

        offset = -1
        quit = FALSE
        trip = FALSE
        i = 0
        j = 0

; Initialize force sensor and enable protected mode

        CALL td_exec_prog("force.init()")

; Set force limits but don't enable guarded mode

        CALL ft_sc_mode_sel(1, "^H12, 2")

        CALL as_set_position(safe)
        BREAK

; Write initial lines to audit file and screen

        CALL post_ext_data(screen, "START OF FORCE1 INFORMATION")
        CALL post_ext_data(screen, "")
        CALL post_ext_data(screen, "attempting first part")
        CALL post_ext_data(disk, "START OF FORCE1 INFORMATION")
        CALL post_ext_data(disk, "")
        CALL post_ext_data(disk, "attempting first part")

; Start operation to 'insert' a part

        WHILE NOT quit DO

            CALL td_exec_prog("acquire.part("+$ENCODE(pallet)+",
                              "+$ENCODE(i)+", "+$ENCODE(j)+")")

            CALL as_set_velocity(100)
            CALL as_set_position(goal1:TRANS(0,0,-offset-100))
            BREAK

            CALL as_set_velocity(50)
            CALL as_set_position(goal1:TRANS(0,0,-offset-20))
            BREAK

            CALL as_set_velocity(5)
            CALL set_ext_loc_dat("location_1", goal1:TRANS(0,0,-offset))
```

```
CALL td_exec_prog("force.moveg(TRUE)")
BREAK
CALL as_set_velocity(100)

IF trip THEN                    ; guard force limit exceeded

    CALL post_ext_data(screen, "Insertion jammed/S")
    CALL post_ext_data(screen, $ENCODE(DISTANCE(
                        goal1:TRANS(0,0,-offset),
                        trip.pos))+"/S")
    CALL post_ext_data(screen, " mm from goal")
    CALL post_ext_data(screen, "Z direction force at time
                         of jam: /S")
    CALL post_ext_data(screen, $ENCODE(forces[2]))
    CALL post_ext_data(screen, "")
    CALL post_ext_data(disk, "Insertion jammed/S")
    CALL post_ext_data(disk, $ENCODE(DISTANCE(
                        goal1:TRANS(0,0,-offset),
                        trip.pos))+"/S")
    CALL post_ext_data(disk, " mm from goal")
    CALL post_ext_data(disk, "Z direction force at time
                          of jam: /S")
    CALL post_ext_data(disk, $ENCODE(forces[2]))
    CALL post_ext_data(disk, "")

    CALL as_set_position(trip.pos:TRANS(0,0,-75))
    CALL td_exec_prog("replace.part("+$ENCODE(pallet)+",
                      "+$ENCODE(i)+", "+$ENCODE(j)+")")
                                    ; return the part

    offset = offset+1
    IF (i <= row-1) THEN          ; increment to next part

        IF j < col-1 THEN
            j = j+1
        ELSE
            i = i+1
            j = 0
        END ; if-else

    END ; if

    IF i == row THEN              ; last part has been tried

        CALL post_ext_data(screen, "Last part failed, part
                                    cannot be inserted")
        CALL post_ext_data(screen, "")
        CALL post_ext_data(disk, "Last part failed, part
                                    cannot be inserted")
        CALL post_ext_data(disk, "")
        CALL as_set_position(safe)
        BREAK
        quit = TRUE

    ELSE                          ; last part hasn't been tried

        CALL post_ext_data(screen, "Attempting next part")
        CALL post_ext_data(disk, "Attempting next part")

    END ; if-else

ELSE                              ; successful insert

    CALL post_ext_data(screen, "Insertion succeeded.")
    CALL post_ext_data(screen, "")
    CALL post_ext_data(disk, "Insertion succeeded.")
    CALL post_ext_data(disk, "")
    CALL gripper_open()
    CALL as_set_position(goal1:TRANS(0,0,-offset-80))
    CALL as_set_position(safe)
    BREAK
    quit = TRUE

END ; if-else

END ; while

RETURN
.END


.PROGRAM force2(pallet)

;  Name:      force2
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program acquires a part from a pallet and sets it on
;  the table surface.  It accomplishes this be moving the part to a
;  position above the surface and repeatedly moving the part down 1mm
;  until the Z direction force exceeds 2 Lbs.  At this point, the part
;  is released and the manipulator moves to a safe location

;  This program is UTAP-compliant

        AUTO offset, i

; Initialize variables

        i = 1
        offset = 0
        trip = FALSE

; Initialize force sensor and enable protected mode
```

```
        CALL td_exec_prog("force.init()")

; Set force limits but don't enable guarded mode

        CALL ft_sc_mode_sel(1, "^H12, 2")

        CALL as_set_position(safe)

        CALL td_exec_prog("acquire.part("+$ENCODE(pallet)+", 3, 1)")

        CALL as_set_velocity(100)
        CALL as_set_position(goal2:TRANS(0,0,-100))
        BREAK

        CALL as_set_velocity(50)
        CALL as_set_position(goal2:TRANS(0,0,-20))
        BREAK

; Write initial lines to audit file

        CALL post_ext_data(screen, "START OF FORCE2 INFORMATION")
        CALL post_ext_data(screen, "")
        CALL post_ext_data(disk, "START OF FORCE2 INFORMATION")
        CALL post_ext_data(disk, "")

; Start the operation to place the part on the surface

        WHILE NOT trip DO

            CALL as_set_velocity(5)
            CALL set_ext_loc_dat("location_1", goal2:TRANS(0,0,-offset))
            CALL td_exec_prog("force.moveg(TRUE)")
            BREAK

            IF NOT trip THEN          ; part is not touching the surface

                CALL post_ext_data(screen, "Failed attempt #/S")
                CALL post_ext_data(screen, $ENCODE(i))
                CALL post_ext_data(disk, "Failed attempt #/S")
                CALL post_ext_data(disk, $ENCODE(i))
                offset = offset-1
                i = i+1

            ELSE                      ; part is touching the surface

                CALL gripper_open()
                CALL post_ext_data(screen, "Succeeded on attempt/S")
                CALL post_ext_data(screen, $ENCODE(i))
                CALL post_ext_data(screen, "Actual position is/S")
                CALL post_ext_data(screen, $ENCODE(DISTANCE(
                                                goal2,trip.pos))+"/S")
                CALL post_ext_data(screen, " mm from the first
                                                attempt")
                CALL post_ext_data(screen, "")
                CALL post_ext_data(disk, "Succeeded on attempt/S")
                CALL post_ext_data(disk, $ENCODE(i))
                CALL post_ext_data(disk, "Actual position is/S")
                CALL post_ext_data(disk, $ENCODE(DISTANCE(
                                                goal2,trip.pos))+"/S")
                CALL post_ext_data(disk, " mm from the first attempt")
                CALL post_ext_data(disk, "")
                CALL as_set_velocity(100)
                CALL as_set_position(trip.pos:TRANS(0,0,-80))
                CALL as_set_position(safe)
                BREAK

            END  ; if-else

        END ; while

        RETURN
.END


.PROGRAM replace.part(pallet, i, j)

;  Name:      replace.part
;  Author:    Capt Matthew L. June - GCS-96D
;  Abstract:  This program starts with a part in the gripper.  it moves
;  the part to a location specified by a pallet number and i and j
;  offsets.  Then, it releases the part and moves to a location 75mm
;  above the part.

;  This program is UTAP-compliant

        AUTO place

        CALL as_set_velocity(100)

        IF pallet == 1 THEN
            SET place = pal.1:TRANS(50*i,50*j)
        ELSE
            SET place = pal.2:TRANS(50*i,50*j)
        END

        CALL as_set_position(place:TRANS(0,0,-75))
        CALL as_set_position(place)
        BREAK

        CALL gripper_open()
```

```
        CALL as_set_position(place:TRANS(0,0,-75))

        RETURN
.END
```

APPENDIX I

V+/UTAP Interface Source Code

Generic Module

```
.PROGRAM get_ext_data(channel, $text, $value)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           get_ext_data
;  UTAP Message:   US_GET_EXT_DATA_VALUE
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to receive data from an IO device.  Maps
;  to the V+ "PROMPT" command.

        CASE channel OF

          VALUE keyboard:
            PROMPT $text, $value

        END

        RETURN
.END


.PROGRAM get_ext_loc_dat($loc_name, loc_value)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           get_ext_loc_dat
;  UTAP Message:   US_GET_EXT_LOCATION_DATA_VALUE
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       New UTAP message to retrieve a V+ transformation
;  location value from the Object Knowledgebase.  Maps to the V+ "SET"
;  command.

        IF $loc_name == "location_1" THEN
            SET loc_value = location_1
        END

        IF $loc_name == "location_2" THEN
            SET loc_value = location_2
        END

        RETURN
.END


.PROGRAM hold($condition)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           hold
;  UTAP Message:   US_HOLD
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to suspend a program until a condition
;  has been met.  Maps to the V+ commands "WAIT", "SIG", and "PENDANT".

        AUTO i, $arg[12], $temp, $command, temp

; Initialize argument array

        FOR i = 0 TO 12
            $arg[i] = ""
        END

; Parse $condition into argument array

        i = 0
        DO
            $arg[i] = $DECODE($condition," ",0)
            $temp = $DECODE($condition," ",1)
            i = i+1
        UNTIL ($condition == "") OR (i > 12)

        $command = "WAIT "

; If the Hold condition is based on SIGNAL values,
; then create the signal portion of the "WAIT" instruction

        i = 0
        IF $arg[i] == "signal" THEN
            $command = $command+"SIG("+$arg[i+1]+")"
            i = i+2
            WHILE ($arg[i] <> "pendant") AND ($arg[i] <> "") DO
                $command = $command+" OR SIG("+$arg[i]+")"
                i = i+1
            END
        END

; If the Hold condition includes waiting on pendant values,
; then add the pendant portion of the "WAIT" instruction

        IF $arg[i] == "pendant" THEN
            i = i+1
            WHILE $arg[i] <> "" DO
                IF i > 1 THEN
                    $command = $command+" OR "
                END
                temp = VAL($arg[i])
                IF temp < 0 THEN
                    $arg[i] = $ENCODE(-temp)
                    $command = $command+"(NOT PENDANT("+$arg[i]+"))"
                ELSE
                    $command = $command+"PENDANT("+$arg[i]+")"
                END
```

```
            i = i+1
        END
    END

; Execute the instruction

    DOS $command

    RETURN
.END


.PROGRAM post_ext_data(channel, $value)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          post_ext_data
;  UTAP Message:  US_POST_EXT_DATA_VALUE
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      UTAP message to write data to an IO device.  Maps to
;  the V+ commands "WRITE" and "TYPE".

    AUTO $command, $temp

    CASE channel OF

      VALUE mcp:
        $command = ""
        DO
            $temp = $DECODE($value,",",0)
            IF POS($temp,"$") THEN
                IF $command == "" THEN
                    $command = $temp
                ELSE
                    $command = $command+","+$temp
                END
            ELSE
                IF $command <> "" THEN
                    DOS "WRITE (mcp) "+$command+", /S"
                END
                WRITE (mcp) $temp, /S
                $command = ""
            END
            $temp = $DECODE($value,",",1)
        UNTIL $temp == ""
        IF $command <> "" THEN
            DOS "WRITE (mcp) "+$command+", /S"
        END

      VALUE screen:
        $temp = $DECODE($value,"/",0)
        TYPE $temp, /S
        DOS "TYPE "+$value

      VALUE disk:
        $temp = $DECODE($value,"/",0)
        WRITE (disk) $temp, /S
        DOS "WRITE (disk) "+$value

    END

    RETURN
.END


.PROGRAM set_ext_loc_dat($loc_name, loc_value)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          set_ext_loc_dat
;  UTAP Message:  US_SET_EXT_LOCATION_DATA_VALUE
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      New UTAP message to write a V+ transformation location
;  value to the Object Knowledgebase.  Maps to the V+ "SET" command.

    IF $loc_name == "location_1" THEN
        SET location_1 = loc_value
    END

    IF $loc_name == "location_2" THEN
        SET location_2 = loc_value
    END

    RETURN
.END
```

Axis Servo Module

```
.PROGRAM as_set_position(joint_position)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          as_set_position
;  UTAP Message:  US_AXIS_SERVO_SET_POSITION
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      UTAP message to move the manipulator.  Maps to the
;  V+ "MOVE" command.

    MOVE joint_position

    RETURN
.END
```

```
.PROGRAM as_set_velocity(joint_velocity)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          as_set_velocity
;  UTAP Message:  US_AXIS_SERVO_SET_VELOCITY
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      UTAP message to set the velocity for the next
;  manipulator movement.  Maps to the V+ "SPEED" command.

        SPEED joint_velocity ALWAYS

        RETURN
.END
```

Object Knowledgebase Module

```
.PROGRAM init_ok()

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          init_ok
;  UTAP Message:  US_INIT_OK
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      UTAP message to initialize the Object Knowledgebase.

        AUTO i

; System channel identifiers

        GLOBAL robot, mcp, screen, keyboard, disk, signal, timer

; Global location variables

        GLOBAL LOC location_1, location_2, location_3

; Force sensor attributes

        GLOBAL trip, trip.pos, forces[]

; Initialize values

        robot = 0
        mcp = 1
        screen = 2
        keyboard = 3
        disk = 5
        signal = 6
        timer = 7

        SET trip.pos = HERE
        trip = FALSE
        FOR i = 0 TO 5
            forces[i] = 0
        END

        RETURN
.END


.PROGRAM ok_attrib_query($attrib, $data)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          ok_attrib_query
;  UTAP Message:  US_OK_ATTRIBUTE_QUERY
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      UTAP message to retrieve data from the Object
;  Knowledgebase.  Maps to the V+ commands "SPEED(x)", "ACCEL(x)",
;  "SWITCH(x)", and "CONFIG(x)"

        IF $attrib == "Monitor Speed" THEN
            $data = $ENCODE(SPEED(1))
        END

        IF $attrib == "Permanent Speed" THEN
            $data = $ENCODE(SPEED(2))
        END

        IF $attrib == "Current Speed" THEN
            $data = $ENCODE(SPEED(3))
        END

        IF $attrib == "Next Speed" THEN
            $data = $ENCODE(SPEED(4))
        END

        IF $attrib == "Acceleration" THEN
            $data = $ENCODE(ACCEL(1))
        END

        IF $attrib == "Deceleration" THEN
            $data = $ENCODE(ACCEL(2))
        END

        IF $attrib == "Gripper Delay" THEN
            $data = $ENCODE(PARAMETER(HAND.TIME))
        END

        IF $attrib == "Continuous Path" THEN
            IF SWITCH(CP) THEN
                $data = "Enabled"
```

```
            ELSE
                $data = "Disabled"
            END
        END

        IF $attrib == "Null-NoNull" THEN
            IF CONFIG(3) BAND 4 THEN
                $data = "NONULL"
            ELSE
                $data = "NULL"
            END
        END

        IF $attrib == "Fine-Coarse" THEN
            IF CONFIG(3) BAND 2 THEN
                $data = "   COARSE"
            ELSE
                $data = "FINE"
            END
        END

        RETURN
.END
```


Programmable I/O Module

```
.PROGRAM pi_bit_read(channel, bit, data)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           pi_bit_read
;  UTAP Message:   US_PIO_BIT_READ
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to read a value from a programmable IO
;  device.  Maps to the V+ commands "PENDANT(x)", "SIG(x)", and
;  "TIMER(x)".

        CASE channel OF

          VALUE mcp:
            data = PENDANT(bit)

          VALUE signal:
            data = SIG(bit)

          VALUE timer:
            data = TIMER(bit)

        END

        RETURN
.END


.PROGRAM pi_bit_set(channel, bit, data)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           pi_bit_set
;  UTAP Message:   US_PIO_BIT_SET
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to write a value to a programmable IO
;  device.  Maps to the V+ commands "KEYMODE", "SIGNAL", and "TIMER".

        CASE channel OF

          VALUE mcp:
            KEYMODE bit = data

          VALUE signal:
            IF data == -1 THEN
                SIGNAL -bit
            ELSE
                SIGNAL bit
            END

          VALUE timer:
            TIMER bit = data

        END

        RETURN
.END


.PROGRAM pi_disable(channel)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           pi_disable
;  UTAP Message:   US_PIO_DISABLE
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to disable a programmable IO device.
;  Maps to the V+ commands "DETACH" and "FCLOSE".

        CASE channel OF

          VALUE mcp:
            DETACH (channel)

          VALUE disk:
            FCLOSE (channel)
            DETACH (channel)
```

```
        END

        RETURN
.END


.PROGRAM pi_enable(channel)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           pi_enable
;  UTAP Message:   US_PIO_ENABLE
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to enable a programmable IO device.
;  Maps to the V+ "ATTACH" command.

        CASE channel OF

          VALUE mcp:
            ATTACH (channel)

          VALUE disk:
            ATTACH (channel) "disk"

        END

        RETURN
.END


.PROGRAM pi_set_mode(channel, $mode)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           pi_set_mode
;  UTAP Message:   US_PIO_SET_MODE
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to set the mode of a programmable IO
;  device.  Maps to the V+ command "FOPENA" and the V+ "REACT" family
;  of signals

        AUTO $fname

        CASE channel OF

          VALUE disk:

            ; Prompt for file name

            PROMPT "Enter the file name for the audit trail
                   (Include drive letter): ", $fname
            TYPE

            IF $mode == "0,0" THEN

                ; Open the file in append mode

                FOPENA (channel, 0, 0) $fname

            END

          VALUE signal:

            DOS $mode

        END

        RETURN
.END
```

Parent Task Program Sequencer Module

```
.PROGRAM ps_select_agent(task_id, $agent)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           ps_select_agent
;  UTAP Message:   US_PTPS_SELECT_AGENT
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to begin the execution of a subprogram.
;  Maps to the V+ "EXECUTE" command.

        AUTO $command

        DOS "EXECUTE "+$ENCODE(task_id)+" "+$agent

        RETURN
.END
```

Sensor Module

```
.PROGRAM ft_sc_disable(mode)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           ft_sc_disable
;  UTAP Message:   US_FT_SENSOR_DISABLE
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       New UTAP message to disable a force/torque sensor.
;  Maps to the V+ "FORCE.MODE" command.
```

```
        FORCE.MODE (-mode)

        RETURN
.END


.PROGRAM ft_sc_enable(mode)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          ft_sc_enable
;  UTAP Message:  US_FT_SENSOR_ENABLE
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      New UTAP message to enable a force/torque sensor.
;  Maps to the V+ "FORCE.MODE" command.

        FORCE.MODE (mode)

        RETURN
.END


.PROGRAM ft_sc_mode_sel(mode, $attr)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          ft_sc_mode_sel
;  UTAP Message:  US_FT_SENSOR_MODE_SELECT
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      New UTAP message to select the mode of operation for
;  a force/torque sensor.  Maps to the V+ "FORCE.MODE" command.

        IF (mode >= 1) OR (mode <= 3) THEN

            DOS "FORCE.MODE ("+$ENCODE(-mode)+") "+$attr

        ELSE

            FORCE.MODE (mode)

        END

        RETURN
.END


.PROGRAM sc_get_att_read($attr_name, attr_val[])

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          sc_get_att_read
;  UTAP Message:  US_SENSOR_GET_ATTRIBUTES_READING
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      UTAP message to read sensor attribute values.  Maps
;  to the V+ commands "FORCE.READ" and "FORCE.OFFSET".

        IF $attr_name == "ratings" THEN

            FORCE.READ (3) attr_val[]

        END

        IF $attr_name == "offset" THEN

            IF DEFINED(attr_val[]) THEN

                FORCE.OFFSET (2) attr_val[]

            ELSE

                FORCE.OFFSET (1)

            END

        END

        RETURN
.END


.PROGRAM sc_get_reading($attribute)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:          sc_get_reading
;  UTAP Message:  US_SENSOR_GET_READING
;  Author:        Capt Matthew L. June - GCS-96D
;  Abstract:      UTAP message to retrieve sensor readings.  Maps to the
;  V+ commands "LATCH", "LATCHED", and "FORCE.READ".

        IF $attribute == "trip" THEN

            trip = LATCHED(1)

        END

        IF $attribute == "trip.pos" THEN

            SET trip.pos = LATCH(1)

        END

        IF $attribute == "forces" THEN

            FORCE.READ (1) forces[]
```

```
        END

        RETURN
.END
```

Tool Control Module

```
.PROGRAM gripper_close()

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           gripper_close
;  UTAP Message:   US_GRIPPER_CLOSE
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       New UTAP message to close the gripper.  Maps to the V+
;  "SIGNAL 1" command.  Incorporates a gripper delay.

        AUTO grip.sig.close, grip_delay

        grip.sig.close = 1
        grip_delay = 0.5

        SIGNAL grip.sig.close
        WAIT.EVENT 0, grip_delay

        RETURN
.END


.PROGRAM gripper_open()

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           gripper_open
;  UTAP Message:   US_GRIPPER_OPEN
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       New UTAP message to open the gripper.  Maps to the V+
;  "SIGNAL -1" command.  Incorporates a gripper delay.

        AUTO grip.sig.open, grip_delay

        grip.sig.open = -1
        grip_delay = 0.5

        SIGNAL grip.sig.open
        WAIT.EVENT 0, grip_delay

        RETURN
.END
```

Task Description Module

```
.PROGRAM td_exec_prog($prog_name)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           td_exec_prog
;  UTAP Message:   US_TDS_EXECUTE_PROGRAM
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to call a subroutine.  Maps to the V+
;  "CALL" command.

        AUTO $command

        DOS "CALL "+$prog_name

        RETURN
.END
```

Task Level Control Module

```
.PROGRAM tl_str_fine_mot($axis_mask)

;          UTAP - V+ INTERFACE PROGRAM
;  Name:           tl_str_fine_mot
;  UTAP Message:   US_US_TLC_START_FINE_MOTION
;  Author:         Capt Matthew L. June - GCS-96D
;  Abstract:       UTAP message to use fine control for manipulator
;  movements.  Maps to the V+ "FINE" command.

        FINE ALWAYS

        RETURN
.END
```

APPENDIX J

Palletizing Application Output

Original Application Output
Part Movement from Pallet 1 to Pallet 2

START OF PALLETIZING INFORMATION

13-Sep-96 09:19:13
Row = 1 Col = 1
The pick location coordinates are:
(230.9396 -248.7725 214.8365 0 179.9998 89.62424)
The place location coordinates are:
(228.2834 52.41686 214.8373 0 179.9993 90.22086)

13-Sep-96 09:19:16
Row = 1 Col = 2
The pick location coordinates are:
(280.9385 -248.4446 214.8366 0 179.9998 89.62424)
The place location coordinates are:
(278.283 52.22413 214.837 0 179.9993 90.22086)

13-Sep-96 09:19:20
Row = 1 Col = 3
The pick location coordinates are:
(330.9375 -248.1167 214.8366 0 179.9998 89.62424)
The place location coordinates are:
(328.2826 52.0314 214.8367 0 179.9993 90.22086)

13-Sep-96 09:19:23
Row = 2 Col = 1
The pick location coordinates are:
(230.6117 -198.7736 214.8364 0 179.9998 89.62424)
The place location coordinates are:
(228.4761 102.4165 214.8369 0 179.9993 90.22086)

13-Sep-96 09:19:27
Row = 2 Col = 2
The pick location coordinates are:
(280.6106 -198.4456 214.8364 0 179.9998 89.62424)
The place location coordinates are:
(278.4757 102.2238 214.8365 0 179.9993 90.22086)

13-Sep-96 09:19:30
Row = 2 Col = 3
The pick location coordinates are:
(330.6096 -198.1177 214.8365 0 179.9998 89.62424)
The place location coordinates are:
(328.4753 102.031 214.8362 0 179.9993 90.22086)

13-Sep-96 09:19:34
Row = 3 Col = 1
The pick location coordinates are:
(230.2838 -148.7746 214.8363 0 179.9998 89.62424)
The place location coordinates are:
(228.6688 152.4161 214.8364 0 179.9993 90.22086)

13-Sep-96 09:19:37
Row = 3 Col = 2
The pick location coordinates are:
(280.2827 -148.4467 214.8363 0 179.9998 89.62424)
The place location coordinates are:
(278.6685 152.2234 214.8361 0 179.9993 90.22086)

13-Sep-96 09:19:41
Row = 3 Col = 3
The pick location coordinates are:
(330.2816 -148.1188 214.8363 0 179.9998 89.62424)
The place location coordinates are:
(328.6681 152.0307 214.8357 0 179.9993 90.22086)

13-Sep-96 09:19:45
Row = 4 Col = 1
The pick location coordinates are:
(229.9559 -98.77571 214.8361 0 179.9998 89.62424)
The place location coordinates are:
(228.8616 202.4157 214.836 0 179.9993 90.22086)

13-Sep-96 09:19:48
Row = 4 Col = 2
The pick location coordinates are:
(279.9548 -98.44778 214.8362 0 179.9998 89.62424)

```
The place location coordinates are:
( 278.8612 202.223 214.8356 0 179.9993 90.22086)

13-Sep-96 09:19:52
Row =  4   Col =  3
The pick location coordinates are:
( 329.9537 -98.11987 214.8362 0 179.9998 89.62424)
The place location coordinates are:
( 328.8608 202.0303 214.8353 0 179.9993 90.22086)

START OF FORCE1 INFORMATION

attempting first part
Insertion jammed 8.63994 mm from goal
Z direction force at time of jam:  6.051636

Attempting next part
Insertion jammed 7.468828 mm from goal
Z direction force at time of jam:  5.245972

Attempting next part
Insertion jammed 6.51549 mm from goal
Z direction force at time of jam:  2.566528

Attempting next part
Insertion jammed 5.866747 mm from goal
Z direction force at time of jam:  5.386353

Attempting next part
Insertion jammed 4.540012 mm from goal
Z direction force at time of jam:  4.507446

Attempting next part
Insertion jammed 3.517776 mm from goal
Z direction force at time of jam:  2.624512

Attempting next part
Insertion jammed 2.449036 mm from goal
Z direction force at time of jam:  2.23999

Attempting next part
Insertion jammed 1.8489 mm from goal
Z direction force at time of jam:  5.648804

Attempting next part
Insertion jammed 0.8689575 mm from goal
Z direction force at time of jam:  2.606201

Attempting next part
Insertion succeeded.

START OF FORCE2 INFORMATION

Failed attempt # 1
Failed attempt # 2
Failed attempt # 3
Failed attempt # 4
Failed attempt # 5
Failed attempt # 6
Failed attempt # 7
Failed attempt # 8
Failed attempt # 9
Failed attempt # 10
Failed attempt # 11
Failed attempt # 12
Succeeded on attempt 13
Actual position is 11.26028 mm from the first attempt
```

Original Application Output
Part Movement from Pallet 2 to Pallet 1

```
START OF PALLETIZING INFORMATION

13-Sep-96 09:22:11
Row =  1   Col =  1
The pick location coordinates are:
( 228.2834 52.41686 214.8373 0 179.9993 90.22086)
The place location coordinates are:
( 230.9396 -248.7725 214.8365 0 179.9998 89.62424)

13-Sep-96 09:22:14
Row =  1   Col =  2
The pick location coordinates are:
( 278.283 52.22413 214.837 0 179.9993 90.22086)
The place location coordinates are:
( 280.9385 -248.4446 214.8366 0 179.9998 89.62424)

13-Sep-96 09:22:18
Row =  1   Col =  3
The pick location coordinates are:
( 328.2826 52.0314 214.8367 0 179.9993 90.22086)
The place location coordinates are:
( 330.9375 -248.1167 214.8366 0 179.9998 89.62424)

13-Sep-96 09:22:21
Row =  2   Col =  1
The pick location coordinates are:
( 228.4761 102.4165 214.8369 0 179.9993 90.22086)
The place location coordinates are:
( 230.6117 -198.7736 214.8364 0 179.9998 89.62424)

13-Sep-96 09:22:25
Row =  2   Col =  2
The pick location coordinates are:
( 278.4757 102.2238 214.8365 0 179.9993 90.22086)
The place location coordinates are:
( 280.6106 -198.4456 214.8364 0 179.9998 89.62424)

13-Sep-96 09:22:28
Row =  2   Col =  3
The pick location coordinates are:
( 328.4753 102.031 214.8362 0 179.9993 90.22086)
The place location coordinates are:
( 330.6096 -198.1177 214.8365 0 179.9998 89.62424)

13-Sep-96 09:22:32
Row =  3   Col =  1
The pick location coordinates are:
( 228.6688 152.4161 214.8364 0 179.9993 90.22086)
The place location coordinates are:
( 230.2838 -148.7746 214.8363 0 179.9998 89.62424)

13-Sep-96 09:22:36
Row =  3   Col =  2
The pick location coordinates are:
( 278.6685 152.2234 214.8361 0 179.9993 90.22086)
The place location coordinates are:
( 280.2827 -148.4467 214.8363 0 179.9998 89.62424)

13-Sep-96 09:22:39
Row =  3   Col =  3
The pick location coordinates are:
( 328.6681 152.0307 214.8357 0 179.9993 90.22086)
The place location coordinates are:
( 330.2816 -148.1188 214.8363 0 179.9998 89.62424)

13-Sep-96 09:22:43
Row =  4   Col =  1
The pick location coordinates are:
( 228.8616 202.4157 214.836 0 179.9993 90.22086)
The place location coordinates are:
( 229.9559 -98.77571 214.8361 0 179.9998 89.62424)

13-Sep-96 09:22:47
Row =  4   Col =  2
The pick location coordinates are:
( 278.8612 202.223 214.8356 0 179.9993 90.22086)
```

```
The place location coordinates are:
( 279.9548 -98.44778 214.8362 0 179.9998 89.62424)

13-Sep-96 09:22:51
Row =  4    Col =  3
The pick location coordinates are:
( 328.8608 202.0303 214.8353 0 179.9993 90.22086)
The place location coordinates are:
( 329.9537 -98.11987 214.8362 0 179.9998 89.62424)

START OF FORCE1 INFORMATION

attempting first part
Insertion jammed 8.842226 mm from goal
Z direction force at time of jam:  2.56958

Attempting next part
Insertion jammed 7.446672 mm from goal
Z direction force at time of jam:  5.24292

Attempting next part
Insertion jammed 6.764458 mm from goal
Z direction force at time of jam:  4.53186

Attempting next part
Insertion jammed 5.815569 mm from goal
Z direction force at time of jam:  4.885864

Attempting next part
Insertion jammed 4.266678 mm from goal
Z direction force at time of jam:  3.326416

Attempting next part
Insertion jammed 3.417786 mm from goal
Z direction force at time of jam:  3.796387

Attempting next part
Insertion jammed 2.54216 mm from goal
Z direction force at time of jam:  4.718018

Attempting next part
Insertion jammed 1.697792 mm from goal
Z direction force at time of jam:  3.674316

Attempting next part
Insertion jammed 0.6267684 mm from goal
Z direction force at time of jam:  3.90625

Attempting next part
Insertion succeeded.

START OF FORCE2 INFORMATION

Failed attempt # 1
Failed attempt # 2
Failed attempt # 3
Failed attempt # 4
Failed attempt # 5
Failed attempt # 6
Failed attempt # 7
Failed attempt # 8
Failed attempt # 9
Failed attempt # 10
Failed attempt # 11
Failed attempt # 12
Succeeded on attempt 13
Actual position is 11.84918 mm from the first attempt
```

UTAP-compliant Application Output
Part Movement from Pallet 1 to Pallet 2

```
START OF PALLETIZING INFORMATION

13-Sep-96 09:47:09
Row =  0   Col =  0
The pick location coordinates are:
( 230.9396 -248.7725 214.8365 0 179.9998 89.62424)
The place location coordinates are:
( 228.2834 52.41686 214.8373 0 179.9993 90.22086)

13-Sep-96 09:47:13
Row =  0   Col =  1
The pick location coordinates are:
( 280.9385 -248.4446 214.8366 0 179.9998 89.62424)
The place location coordinates are:
( 278.283 52.22413 214.837 0 179.9993 90.22086)

13-Sep-96 09:47:17
Row =  0   Col =  2
The pick location coordinates are:
( 330.9375 -248.1167 214.8366 0 179.9998 89.62424)
The place location coordinates are:
( 328.2826 52.0314 214.8367 0 179.9993 90.22086)

13-Sep-96 09:47:21
Row =  1   Col =  0
The pick location coordinates are:
( 230.6117 -198.7736 214.8364 0 179.9998 89.62424)
The place location coordinates are:
( 228.4761 102.4165 214.8369 0 179.9993 90.22086)

13-Sep-96 09:47:25
Row =  1   Col =  1
The pick location coordinates are:
( 280.6106 -198.4456 214.8364 0 179.9998 89.62424)
The place location coordinates are:
( 278.4757 102.2238 214.8365 0 179.9993 90.22086)

13-Sep-96 09:47:29
Row =  1   Col =  2
The pick location coordinates are:
( 330.6096 -198.1177 214.8365 0 179.9998 89.62424)
The place location coordinates are:
( 328.4753 102.031 214.8362 0 179.9993 90.22086)

13-Sep-96 09:47:33
Row =  2   Col =  0
The pick location coordinates are:
( 230.2838 -148.7746 214.8363 0 179.9998 89.62424)
The place location coordinates are:
( 228.6688 152.4161 214.8364 0 179.9993 90.22086)

13-Sep-96 09:47:37
Row =  2   Col =  1
The pick location coordinates are:
( 280.2827 -148.4467 214.8363 0 179.9998 89.62424)
The place location coordinates are:
( 278.6685 152.2234 214.8361 0 179.9993 90.22086)

13-Sep-96 09:47:41
Row =  2   Col =  2
The pick location coordinates are:
( 330.2816 -148.1188 214.8363 0 179.9998 89.62424)
The place location coordinates are:
( 328.6681 152.0307 214.8357 0 179.9993 90.22086)

13-Sep-96 09:47:45
Row =  3   Col =  0
The pick location coordinates are:
( 229.9559 -98.77571 214.8361 0 179.9998 89.62424)
The place location coordinates are:
( 228.8616 202.4157 214.836 0 179.9993 90.22086)

13-Sep-96 09:47:49
Row =  3   Col =  1
The pick location coordinates are:
( 279.9548 -98.44778 214.8362 0 179.9998 89.62424)
```

```
The place location coordinates are:
( 278.8612 202.223 214.8356 0 179.9993 90.22086)

13-Sep-96 09:47:53
Row =  3    Col =  2
The pick location coordinates are:
( 329.9537 -98.11987 214.8362 0 179.9998 89.62424)
The place location coordinates are:
( 328.8608 202.0303 214.8353 0 179.9993 90.22086)

START OF FORCE1 INFORMATION

attempting first part
Insertion jammed 8.613272 mm from goal
Z direction force at time of jam:  5.526733

Attempting next part
Insertion jammed 7.539873 mm from goal
Z direction force at time of jam:  3.741455

Attempting next part
Insertion jammed 6.320009 mm from goal
Z direction force at time of jam:  4.51355

Attempting next part
Insertion jammed 5.915636 mm from goal
Z direction force at time of jam:  5.285645

Attempting next part
Insertion jammed 4.56438 mm from goal
Z direction force at time of jam:  2.978516

Attempting next part
Insertion jammed 3.540012 mm from goal
Z direction force at time of jam:  2.526855

Attempting next part
Insertion jammed 2.449041 mm from goal
Z direction force at time of jam:  2.279663

Attempting next part
Insertion jammed 1.915497 mm from goal
Z direction force at time of jam:  2.255249

Attempting next part
Insertion jammed 0.9022217 mm from goal
Z direction force at time of jam:  3.442383

Attempting next part
Insertion succeeded.

START OF FORCE2 INFORMATION

Failed attempt # 1
Failed attempt # 2
Failed attempt # 3
Failed attempt # 4
Failed attempt # 5
Failed attempt # 6
Failed attempt # 7
Failed attempt # 8
Failed attempt # 9
Failed attempt # 10
Failed attempt # 11
Failed attempt # 12
Succeeded on attempt 13
Actual position is 11.27139 mm from the first attempt
```

UTAP-compliant Application Output
Part Movement from Pallet 2 to Pallet 1

```
START OF PALLETIZING INFORMATION

13-Sep-96 09:54:19
Row =  0   Col =  0
The pick location coordinates are:
( 228.2834 52.41686 214.8373 0 179.9993 90.22086)
The place location coordinates are:
( 230.9396 -248.7725 214.8365 0 179.9998 89.62424)

13-Sep-96 09:54:23
Row =  0   Col =  1
The pick location coordinates are:
( 278.283 52.22413 214.837 0 179.9993 90.22086)
The place location coordinates are:
( 280.9385 -248.4446 214.8366 0 179.9998 89.62424)

13-Sep-96 09:54:27
Row =  0   Col =  2
The pick location coordinates are:
( 328.2826 52.0314 214.8367 0 179.9993 90.22086)
The place location coordinates are:
( 330.9375 -248.1167 214.8366 0 179.9998 89.62424)

13-Sep-96 09:54:31
Row =  1   Col =  0
The pick location coordinates are:
( 228.4761 102.4165 214.8369 0 179.9993 90.22086)
The place location coordinates are:
( 230.6117 -198.7736 214.8364 0 179.9998 89.62424)

13-Sep-96 09:54:35
Row =  1   Col =  1
The pick location coordinates are:
( 278.4757 102.2238 214.8365 0 179.9993 90.22086)
The place location coordinates are:
( 280.6106 -198.4456 214.8364 0 179.9998 89.62424)

13-Sep-96 09:54:39
Row =  1   Col =  2
The pick location coordinates are:
( 328.4753 102.031 214.8362 0 179.9993 90.22086)
The place location coordinates are:
( 330.6096 -198.1177 214.8365 0 179.9998 89.62424)

13-Sep-96 09:54:43
Row =  2   Col =  0
The pick location coordinates are:
( 228.6688 152.4161 214.8364 0 179.9993 90.22086)
The place location coordinates are:
( 230.2838 -148.7746 214.8363 0 179.9998 89.62424)

13-Sep-96 09:54:47
Row =  2   Col =  1
The pick location coordinates are:
( 278.6685 152.2234 214.8361 0 179.9993 90.22086)
The place location coordinates are:
( 280.2827 -148.4467 214.8363 0 179.9998 89.62424)

13-Sep-96 09:54:51
Row =  2   Col =  2
The pick location coordinates are:
( 328.6681 152.0307 214.8357 0 179.9993 90.22086)
The place location coordinates are:
( 330.2816 -148.1188 214.8363 0 179.9998 89.62424)

13-Sep-96 09:54:55
Row =  3   Col =  0
The pick location coordinates are:
( 228.8616 202.4157 214.836 0 179.9993 90.22086)
The place location coordinates are:
( 229.9559 -98.77571 214.8361 0 179.9998 89.62424)

13-Sep-96 09:54:59
Row =  3   Col =  1
The pick location coordinates are:
( 278.8612 202.223 214.8356 0 179.9993 90.22086)
```

```
The place location coordinates are:
( 279.9548 -98.44778 214.8362 0 179.9998 89.62424)

13-Sep-96 09:55:04
Row =  3   Col =  2
The pick location coordinates are:
( 328.8608 202.0303 214.8353 0 179.9993 90.22086)
The place location coordinates are:
( 329.9537 -98.11987 214.8362 0 179.9998 89.62424)

START OF FORCE1 INFORMATION

attempting first part
Insertion jammed 8.815492 mm from goal
Z direction force at time of jam:  4.180908

Attempting next part
Insertion jammed 7.444384 mm from goal
Z direction force at time of jam:  4.504395

Attempting next part
Insertion jammed 6.615496 mm from goal
Z direction force at time of jam:  5.960083

Attempting next part
Insertion jammed 5.842226 mm from goal
Z direction force at time of jam:  6.265259

Attempting next part
Insertion jammed 4.591052 mm from goal
Z direction force at time of jam:  4.019165

Attempting next part
Insertion jammed 3.422165 mm from goal
Z direction force at time of jam:  4.440308

Attempting next part
Insertion jammed 2.59111 mm from goal
Z direction force at time of jam:  3.637695

Attempting next part
Insertion jammed 1.715569 mm from goal
Z direction force at time of jam:  2.74353

Attempting next part
Insertion jammed 0.6622521 mm from goal
Z direction force at time of jam:  4.174805

Attempting next part
Insertion succeeded.

START OF FORCE2 INFORMATION

Failed attempt # 1
Failed attempt # 2
Failed attempt # 3
Failed attempt # 4
Failed attempt # 5
Failed attempt # 6
Failed attempt # 7
Failed attempt # 8
Failed attempt # 9
Failed attempt # 10
Failed attempt # 11
Failed attempt # 12
Succeeded on attempt 13
```
Actual position is 11.83362 mm from the first attempt

VITA

Capt Matthew L. June [redacted] He graduated from Rogers High School in Wyoming, MI, in 1980. He enlisted in the United States Air Force and entered active duty on July 20, 1981. He received his Bachelor of Science degree in Computer Science from Wright State University on June 9, 1990 and received his Air Force commission on October 1, 1990.

His first assignment as a commissioned officer was at Ramstein AB, GE, as Chief, Network Design and Implementation Section. He then moved to Wright-Patterson AFB, OH where he served as the Computer Resource Manager for the F-15 System Program Office. In June of 1995 he entered the School of Engineering at the Air Force Institute of Technology.

[redacted]

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1996	Master's Thesis

4. TITLE AND SUBTITLE
Analysis And Design Of Standard Telerobotic Control Software

5. FUNDING NUMBERS

6. AUTHOR(S)
Matthew L. June
Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/96D-12

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt Thomas Deeter
SA-ALC.TIER Bldg 324
505 Perrin Rd
Kelly AFB, TX 78241-6435

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Robotics and Automation Center for Excellence (RACE) has defined an open telerobotics control architecture. This architecture, called the Unified Telerobotic Architecture Project (UTAP), is a proposed standard for all Air Force telerobotic systems. Implementation of UTAP will reduce the cost of robotic applications by increasing software modularity, portability, and reusability. This thesis continued the effort to prove the feasibility of UTAP. In December, 1995, 1st Lt Anchor implemented a portion of the UTAP specification on a PUMA robot. The UTAP-compliant controller exhibited some degradation in the system performance. However, the performance degradation was not fully measured. This thesis extended the measurements of Anchor's implementation. Additionally, a portion of the UTAP specification was implemented on an Adept 550 manipulator and the performance effects were measured. The implementation included portions of the generic, robot/axis servo control, tool control, sensor control, programmable IO, subsystem task level control, task description and supervision, parent task program sequencer, task program sequencer, and object knowledge modules. Performance measurements of this implementation indicated that, although performance was adversely affected, the degradation was caused by the interface between the UTAP-compliant application and the non-UTAP-compliant operating system. There was little difference between the complaint and non- compliant applications. Successful implementation of the UTAP specification on the PUMA and Adept manipulators proves that it is a feasible telerobotic architecture. Further study of the specification is recommended. Specifically, the development of a UTAP-compliant operating system should be continued.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Robot, Robotics, Standardization, Control, Archetecture	142
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL