DESIGN AND IMPLEMENTATION OF A MODULAR MANIPULATOR ARCHITECTURE

By

OGNJEN SOSA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULLFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING

UNIVERSITY OF FLORIDA

2004

Copyright 2004

by

Ognjen Sosa

To my mother, father and brother.

“If a man empties his purse into his head, no man can take it away from him. An investment in knowledge always pays the best interest.”

— Benjamin Franklin

ACKNOWLEDGMENTS

I would like to express my thanks to my entire committee (Dr. Carl Crane, Dr. John Schueller, and Dr. John Ziegert) for their support of my study. Furthermore, I would like to extend my thanks to Dr. Robert Bicker (of the University of Newcastle upon Tyne) for his insightful ideas and hands-on support.

Special thanks go to my advisor Prof. Carl Crane, for his patience with my endeavors; to Mr. Dave Armstrong, our project manager, for always trying to keep me on track; and to Tyndall Airforce Base (contract # F08637-00-C6008) for providing financial support for my studies. I am also grateful to the entire staff of the Center for Intelligent Machines and Robotics (with distinguished contributions from David Kent, Tom Galluzzo, Shannon Ridgeway, and Hyun Kwon Jung).

Finally I would like to thank all of my friends and extended family for their moral support and guidance over the last 7 years. However, without the love and patience of those closest to me, this work would have never been accomplished. Thus, my greatest appreciation goes to my mother, Jadranka; my father Darko; and my brother Vedran. Their pride and recognition of my accomplishments had an immense impact on my motivation and provided a strong driving mechanism during my studies.

TABLE OF CONTENTS

page

ACKNOWLEDGMENTS iv

LIST OF TABLES x

LIST OF FIGURES xii

ABSTRACT xv

CHAPTER

1 INTRODUCTION 1

1.1 Introduction 1
1.2 Background 3
1.3 Joint Architecture for Unmanned Systems (JAUS) 4
1.3.1 Overview 4
1.3.2 Standards 5
1.3.3 System Topology 7
1.3.4 Component Definition 9
1.3.5 Message Specification 10

2 SYSTEM ANALYSIS AND OVERVIEW 13

2.1 Mechanism Overview 14
2.2 Kinematic Analysis of the Puma 762 Robot 14
2.2.1 Notation 14
2.2.2 Forward and Reverse Position Analysis 17
2.2.3 Forward and Reverse Velocity Analysis 23
2.2.4 Singularity Determination 24
2.3.5 Quaternion Representations 26

3 PUMA 762 CONTROLLER SYSTEM 28

3.1 Overview 28
3.2 Reverse Engineering the Puma 762 Robot 29
3.2.1 Existing Architecture 29
3.2.2 Encoder and Potentiometer Val Interface 30
3.2.3 Amplifier Digital-to-Analog Converter Signals and Control Lines 31

3.2.4 Safety 32
3.2.5 Tuning 32
3.3 Galil DMC-2100 Functionality 33
3.3.1 Command Modes 33
3.3.2 Theory of Operation 34
3.4 Galil C/C++ Application Programming Interface (API) 35

4 LOW-LEVEL MANIPULATOR CONTROL COMPONENT 37

4.1 Primitive Manipulator Component 37
4.1.1 Definition of Coordinate Systems 37
4.1.1.1 Global coordinate system 37
4.1.1.2 Vehicle coordinate system 37
4.1.1.3 Manipulator base coordinate system 38
4.1.1.4 End-effector coordinate system 38
4.1.2 Component Function 38
4.1.3 Associated Messages 39
4.1.4 Component Description 39
4.1.5 Input and Output Messages 40
4.1.5.1 Code 0601h: Set Joint Effort 40
4.1.5.2 Code 2600h: Query Manipulator Specifications 41
4.1.5.3 Code 2601h: Query Joint Effort 41
4.1.5.4 Code 4600h: Report Manipulator Specifications 41
4.1.5.5 Code 4601h: Report Joint Effort 41
4.2 Primitive Manipulator Applications to the Puma system 44

5 MANIPULATOR SENSOR COMPONENTS 45

5.1 Manipulator Joint Position Sensor Component 45
5.1.1 Component Function 45
5.1.2 Associated Messages 45
5.1.3 Component Description 46
5.1.4 Input and Output Messages 46
5.1.4.1 Code 0602h: Set Joint Positions message 46
5.1.4.2 Code 2602h: Query Joint Positions message 47
5.1.4.3 Code 4602h: Report Joint Positions message 47
5.2 Manipulator Joint Velocity Sensor Component 47
5.2.1 Component Function 47
5.2.2 Associated Messages 47
5.2.3 Component Description 47
5.2.4 Input and Output Messages 48
5.2.4.1 Code 0603h: Set Joint Velocities message 48
5.2.4.3 Code 4603h: Report Joint Velocities message 48
5.3 Manipulator Joint Force/Torque Sensor Component 49
5.3.1 Component Function 49
5.3.2 Associated Messages 49
5.3.3 Component Description 49

5.3.4 Input and Output Messages 49
5.3.4.1 Code 2605: Query Joint Force/Torques 49
5.3.4.2 Code 4605h: Report Joint Force/Torques 49
5.4 Sensor Component Applications to the Puma System 50

6 MANIPULATOR LOW-LEVEL POSITION AND VELOCITY DRIVER COMPONENTS 51

6.1 Manipulator Joint Positions Driver Component 51
6.1.1 Component Function 51
6.1.2 Associated Messages 51
6.1.3 Component Description 52
6.2 Manipulator End-Effector Pose Driver Component 52
6.2.1 Component Function 52
6.2.2 Associated Messages 52
6.2.3 Component Description 53
6.2.4 Input and Output Messages 53
6.2.4.1 Code 0604h: Set Tool Point message 53
6.2.4.2 Code 0605h: Set End-Effector Pose message 54
6.2.4.3 Code 2604h: Query Tool Point 54
6.2.4.4 Code 4604h: Report Tool Point 55
6.3 Manipulator Joint Velocities Driver Component 55
6.3.1 Component Function 55
6.3.2 Associated Messages 55
6.3.3 Component Description 55
6.4 Manipulator End-Effector Velocity State Driver Component 56
6.4.1 Component Function 56
6.4.2 Associated Messages 56
6.4.3 Component Description 57
6.4.4 Code 0606h: Set End-Effector Velocity State message 57
6.5 Applications of the Low-Level Driver Components to the Puma System 58

7 MID-LEVEL POSITION AND VELOCITY DRIVER COMPONENTS 61

7.1 Manipulator Joint Move Driver Component 61
7.1.1 Component Function 61
7.1.2 Associated Messages 61
7.1.3 Component Description 62
7.1.4 Code 0607: Set Joint Motion 63
7.2 Manipulator End-Effector Discrete Pose Driver Component 65
7.2.1 Component Function 65
7.2.2 Associated Messages 65
7.2.3 Component Description 65
7.2.4 Code 0608h: Set End-Effector Path Motion 66
7.3 Applications of the Mid-Level Driver Components to the Puma System 66

8 OVERVIEW OF SOFTWARE DESIGN 69

8.1 Overview 69
8.2 The Interface to the Galil Controller 70
8.3 Component Level Software Development 72
8.3.1 Primitive Manipulator State Thread 73
8.3.2 Manipulator Sensor Components 74
8.3.3 Low-Level Position and Velocity Drivers 75
8.4 Message Level Software Development 76
8.5 Node Level Software Development 76
8.6 Node Manager and Communicator 77

9 TESTING AND RESULTS 78

9.1 Subsystem Commander Component Overview 78
9.2 Case 1: Set Joint Effort 79
9.3 Case 2: Set Joint Position 81
9.3.1 The “Average” Set 82
9.3.2 The “Low” Set 84
9.3.3 The “High” Set 85
9.4 Case 3: Set End-Effector Pose 88
9.5 Case 4: Set Joint Velocities 89
9.6 Case 5: Set End-Effector Velocity State 90
9.7 Case 6: Set Joint Motion 91
9.7.1 Set Joint Motion: Pose 1 91
9.7.2 Set Joint Motion: Pose 2 93
9.7.3 Set Joint Motion: Pose 3 94
9.8 Case 7: Set End-Effector Path Motion 96

10 CONCLUSIONS AND FUTURE WORK 99

10.1 Conclusions 99
10.2 Future Work 101

APPENDIX

A EQUATIONS FOR A SPHERICAL HEPTAGON 102

B JAUS MANIPULATOR COMPONENTS: SOURCE CODE 104

B.1 The GalilInterface.c File and the Corresponding Header GalilInterface.h 104
B.2 The Pm.c File and the Corresponding Header Pm.h 117
B.3 The Mjps.c File and the Corresponding Header Mjps.h 130
B.4 The Meepd.c File and the Corresponding Header Meepd.h 141
B.5 The Meedpd.c File and the Corresponding Header Meedpd.h 159
B.6 The Mc.c File and the Corresponding Header Mc.h 178
B.7 The Main.c File 182

C SOURCE CODE FOR THE USER DEFINED JAUS MESSAGES 200

C.1 The JointEffort.c File and the Corresponding Header JointEffort.h.....200
C.2 The JointPosition.c File and the Corresponding Header JointPosition.h.....202
C.3 The EndEffectorPose.c File 204

LIST OF REFERENCES.....207

BIOGRAPHICAL SKETCH 209

LIST OF TABLES

Table		page
1-1	The JAUS core message set	9
1-2	Segmentation of command codes by class	10
1-3	Message header data format	12
2-1	Mechanical specifications of the Puma 762 robot	14
2-2	Mechanism parameters of the Puma 762 robot	15
2-3	Closed-loop mechanism parameters of the Puma 762 robot	19
3-1	Pin connections between Val and Galil ICM-2900 interconnect modules	31
3-2	Controller gain values	33
4-1	Set Joint Effort message parameters	40
4-2	Report Manipulator Specifications parameters	42
5-1	Set Joint Positions message parameters	46
5-2	Set Joint Velocities message parameters	48
5-3	Report Joint Force/Torque message parameters	50
6-1	Set Tool Point message parameters	54
6-2	Set End-Effector Pose message parameters	54
6-3	Set End-Effector Velocity State message parameters	58
7-1	Set Joint Motion message parameters	63
7-2	Set End-Effector Path Motion message parameters	67
9-1	The Puma 762 platform specific conversion factors	81
9-2	Values of the K constant and corresponding range of motion parameters	82

9-3 Values of the K constant and “average” set of motion parameters.....82

9-4 Values of the K constant and “low” set of motion parameters.....84

9-5 Values of the K constant and “high” set of motion parameters.....86

A-1 Fundamental formulas for a Spherical Heptagon 102

A-2 Subsidiary formulas for a Spherical Heptagon Set 1.....103

LIST OF FIGURES

Figure		page
1-1	Manipulator linkage parameters for link ij	6
1-2	Manipulator linkage parameters for revolute joint j	6
1-3	Manipulator linkage parameters for prismatic joint j	7
1-4	Architecture hierarchy	8
1-5	Architecture hierarchy of the manipulator control node	8
1-6	The JAUS message header detail	11
1-7	Bit layout of message properties	11
2-1	Puma 762, 6-DOF robot manipulator	13
2-2	Labeled kinematic model of the Puma 762 manipulator	15
2-3	Reverse analysis solution tree for Puma 762 robot	22
2-4	Wrist singularity of Puma 762 robot	25
2-5	Forearm boundary singularity of Puma 762 robot	25
2-6	Forearm interior singularity of Puma 762 robot	26
3-1	Schematic representation entire manipulator system	29
3-2	Arm signal interconnects between Val and ICM-2900 modules	30
3-3	BRK-ON-HI connection diagram	32
3-4	Functional elements of a motion control system	34
3-5	Initializing the Galil libraries	35
3-6	Establishing communications with the Galil Controller	35
3-7	Closing the connection to the Galil Controller	35

3-8 Sending commands to the Galil Controller....36
3-9 Resetting the controller....36
4-1 Joint effort provides basic manipulator mobility....40
5-1 Joint position sensor component....46
5-2 Joint velocity sensor component....48
6-1 Manipulator Joint Positions Driver component....52
6-2 Manipulator End-Effector Pose Driver component....53
6-3 Manipulator Joint Velocities Driver component....56
6-4 Manipulator End-Effector Velocity State Driver component....57
7-1 Manipulator Joint Move Driver component....62
7-2 Manipulator End-Effector Discrete Pose Driver component....66
8-1 The galilInterface.c logic flow diagram....70
8-2 Generic component logic flow diagram....72
8-3 The Primitive Manipulator state thread logic flow diagram....73
8-4 Manipulator Joint Position Sensor logic flow diagram....74
8-5 Manipulator End-Effector Pose Driver logic flow diagram....75
9-1 Subsystem Commander operator graphical user interface....79
9-2 Primitive Manipulator Screen as it responds to the Set Joint Effort message....80
9-3 External closed-loop control diagram....81
9-4 Joint step responses using “average” motion parameters....83
9-5 The MJPD screen as motion is completed under the “average” set....83
9-6 The MJPD screen as motion is completed under the “low” set....84
9-7 Joint step responses using “low” motion parameters....85
9-8 Joint step responses using “high” motion parameters....86
9-9 The MJPD screen as motion is completed under the “high” set....87

9-10 Subsystem Commander screen showing the final configuration of Puma 762 87

9-11 The MEEPD screen as it responds to Set End-Effector Pose message.....89

9-12 The MJVD Screen as it responds to Set Joint Velocities message 90

9-13 The MJMD screen showing component status upon completion of pose 1 91

9-14 Joint performance plots for motion parameters set in pose 1 92

9-15 The MJMD screen showing component status upon completion of pose 2 93

9-16 Joint performance plots for motion parameters set in pose 2 94

9-17 The MJMD screen showing component status upon completion of pose 3 95

9-18a Joint performance plots for motion parameters set in pose 3 95

9-18b Joint performance plots for motion parameters set in pose 3 96

9-19 The MEEDPD screen after completion of pose 1.....97

9-20 The MEEDPD screen after completion of pose 5.....97

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering

DESIGN AND IMPLEMENTATION OF A MODULAR MANIPULATOR ARCHITECTURE

By

Ognjen Sosa

December 2004

Chair: Carl D. Crane, III
Major Department: Mechanical and Aerospace Engineering

The Joint Architecture for Unmanned Systems (JAUS) has successfully established a well-defined component interface for unmanned mobile systems, but has yet to address the implications of such systems requiring an on-board robot manipulator. This configuration is seen in many applications including planetary exploration, hazardous materials removal, and marine research and is frequently referred to as the vehicle-manipulator system. The purpose of our study was to develop and implement a set of JAUS components that will allow for tele-operational (teleop) and autonomous control of a vehicle-manipulator platform. Teleop control is the control of a system by the direct input of a human or a computer. Autonomous control is a cooperative mode between the vehicle and the manipulator.

Testing and implementation of these components was performed on a 6-degree-of-freedom (6-DOF) Puma 762 robot manipulator (Unimation – Westinghouse, Danbury, Connecticut) outfitted with a commercially available Galil motion controller (Galil

Motion Control, Inc, Rocklin, California). Successful completion and adequate compatibility to the outlined JAUS performance specifications would ensure the inception of the above mentioned components to the latest version of the document's Reference Architecture. Even though this modular manipulator architecture is suitable for both autonomous and teleop control, testing and results were based solely on the input provided using a graphical user interface on a computer.

CHAPTER 1
INTRODUCTION

1.1 Introduction

In the past decade, significant technological breakthroughs have led to smarter and more-reliable unmanned systems. Reliability and robustness of these systems depend on many factors, most of which are defined in the early stages of design and development. With an increasing interest in this field and tremendous implications to both civilian and military use, there is a growing demand for a set of well-defined standards that could ensure safety, reliability, and interoperability. The Center for Intelligent Machines and Robotics (CIMAR) at the University of Florida has been involved in developing of one such standard, in cooperation with the Department of Defense.

The Joint Architecture for Unmanned Systems (JAUS) has successfully established a well-defined component interface for unmanned mobile systems, but has yet to address the implications of such systems requiring an on-board robot manipulator. This configuration is seen in many applications including planetary exploration, hazardous materials removal, and marine research; and is frequently referred to as a vehicle-manipulator system. The purpose of our study is to develop and implement a set of JAUS components that will allow for tele-operational (teleop) and autonomous control of a vehicle-manipulator platform. Teleop control is defined as the control of a system by the direct input of a human or a computer. Autonomous control is defined as a cooperative mode between the vehicle and the manipulator. The components focus on a serial manipulator comprising of any number of prismatic and revolute joints. Parallel

mechanisms are not addressed as a part of our study. The components are grouped according to function into the following categories:

- **Low-level manipulator control components:** The one component in this category allows for low-level command of the manipulator joint-actuation efforts. This is an open-loop command that could be used in a simple tele-operation scenario. The component in this category is listed as follows:
  - Primitive manipulator component
- **Manipulator sensor components:** These components, when queried, return instantaneous sensor data. Three components are defined that return respectively joint positions, joint velocities, and joint torques or forces. The components in this category are listed as follows:
  - Manipulator joint position sensor component
  - Manipulator joint velocity sensor component
  - Manipulator joint force/torque sensor component
- **Low-level position and velocity driver components:** These components take as inputs the desired joint positions, the desired joint velocities, the desired end-effector pose, or the desired end-effector velocity state. Closed-loop control is implied. No path information is specified. The components in this category are listed as follows:
  - Manipulator joint positions driver component
  - Manipulator end-effector pose driver component
  - Manipulator joint velocities driver component
  - Manipulator end-effector velocity state driver component
- **Mid-level position and velocity driver components:** Two components are grouped under this heading. The first takes as input the goal values for each joint parameter at several time values together with motion constraints (i.e. maximum joint velocity, maximum acceleration, and maximum deceleration). The second takes as input a series of end-effector poses at specified time values. Closed-loop control is implied. The components in this category are:
  - Manipulator Joint Move Driver Component
  - Manipulator End-Effector Discrete Pose Driver Component.

Testing and implementation of these components is performed on a 6-degree-of-freedom (6-DOF) Puma 762 robot manipulator outfitted with a commercially available Galil motion controller. Successful completion and adequate compatibility to the outlined JAUS performance specifications would ensure the inception of the above-mentioned components to the latest version of the document's Reference Architecture. Even though this modular-manipulator architecture is suitable for both autonomous and teleop control, testing and results are based solely on input provided using a graphical user interface on a computer.

1.2 Background

Robot manipulators are used in a wide variety of applications; but usually follow user-defined paths, thus requiring teleop control. In the case of a manipulator aboard an autonomous mobile system, there is a growing demand for the manipulator and the vehicle to be able to cooperate, allowing the system to autonomously complete more-complex tasks (such as sample acquisition, instrument placement, and mobility assistance). This problem was first addressed by NASA in the mid 1990s as the first rover missions to Mars were planned. Technology developed in 1998 [1] allowed robot manipulators aboard the vehicles to autonomously acquire small rock samples (designated by the user) within 1 meter away; and to place instruments on targets less than 5 meters away. These capabilities are accomplished with onboard vision sensors using stereo processing algorithms developed by Matthies [2]. This application showed the autonomous behavior of the manipulator but did not consider other high-level vehicle functions. The algorithms were designed to be portable, extendible, and reusable across many vehicle platforms. Testing was successfully completed on Rocky 7 [3].

Other inhospitable environments (such as oceans) are frequently explored using underwater vehicle-manipulator systems (UVMS). These systems are mostly used for inspection, drilling, mine countermeasures, and surveying [4]. In most applications, UVMS operates as a master-slave configuration, and as such is prone to a few deficiencies. Challenges include correctly modeling the nonlinear hydrodynamic effects of the environment; and improving the performance of motion control encompassing singularity avoidance, obstacle detection, and power optimization [5]. Recent research [6] in the field of UVMS uses a unified force control approach, which combines impedance control with hybrid position/force control by means of fuzzy switching to perform autonomous underwater manipulation.

1.3 Joint Architecture for Unmanned Systems (JAUS)

1.3.1 Overview

The Joint Architecture for Unmanned Systems (JAUS) [7] is being developed in conjunction with the Department of Defense and many other members of industry and academia for use in research, development and acquisition of unmanned systems. The current version of the document is divided into three large volumes; the JAUS Domain Model (Vol. I), JAUS Reference Architecture (RA-Vol. II), and JAUS Document Control Plan (Vol. III). The JAUS Working Group was chartered to reduce lifecycle costs, and integration and development time; to provide a framework for technology insertion; to accommodate expansion of existing systems with new capabilities.

Volume I defines known and prospective operational requirements of unmanned systems. Volume III defines the process used to identify and track requested changes to accepted JAUS documentation. Components developed as a part of our study affect all three parts of Volume II because it is concerned with aspects of component design. The

main purpose of the Reference Architecture (RA) is to describe all functions and messages that shall be used to design new components. Joint Architecture for Unmanned Systems defines components for all classifications of Unmanned Systems, from teleop to autonomous. As a particular system evolves, the architecture is already in place to support more-advanced capabilities. To meet this requirement, four technical constraints are imposed on JAUS:

- **Platform independence:** Since unmanned systems will be based on a variety of missions, no assumptions are made regarding the vehicle platform.
- **Mission isolation:** Joint Architecture for Unmanned Systems defines a mission as the ability to gather information about or to alter the state of the environment in which the platform is operating. This allows the developer to construct the system to support a variety of missions.
- **Computer hardware independence:** Advances in computer technology over the past couple of decades have seen rapid growth. The JAUS computer hardware constraint was put in place to ensure software and hardware portability as new systems are developed in the future.
- **Technology independence:** This constraint is similar to hardware independence, but focuses more on technical approach [7, 8]. In this particular application, the architecture makes no assumption regarding the method used to obtain joint positions. For example different manipulators could be outfitted by different position sensors that include encoders, potentiometers, or rotational variable differential transformers.

1.3.2 Standards

The Reference Architecture (RA) document defines the JAUS specific protocol for transmission of messages. Aside from computer code specifications, this section also standardizes mathematical notations. The notation used in this thesis is standard to many popular robotics textbooks; specifically, we use the definitions covered by Crane and Duffy [9]. To standardize the method of referencing manipulator links (or joints), Figures 1-1 through 1-3 illustrate the notation that will be used later in the document. Figure 1-1

shows the parameters used for link ij. Figure 1-2 and Figure 1-3 show additional parameters used to define rotational joints and prismatic joints, respectively.

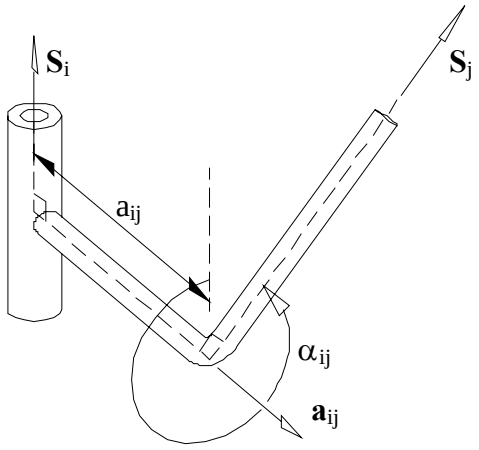

Figure 1-1. Manipulator linkage parameters for link ij (Source: Crane C., Duffy J., Kinematic Analysis of Robot Manipulators, Cambridge University Press, 1998, p. 21, Figure 3.2)

Link length a_{ij} is measured as the perpendicular distance between joint axis i and joint axis j along the unit vector $\mathbf{a}_{ij}$. Note that it can have a negative value. Twist angle α_{ij} is the angle between $\mathbf{S}_i$ (unit vector along joint axis i) and $\mathbf{S}_j$ (unit vector along joint axis j) measured in a right hand sense about $\mathbf{a}_{ij}$.

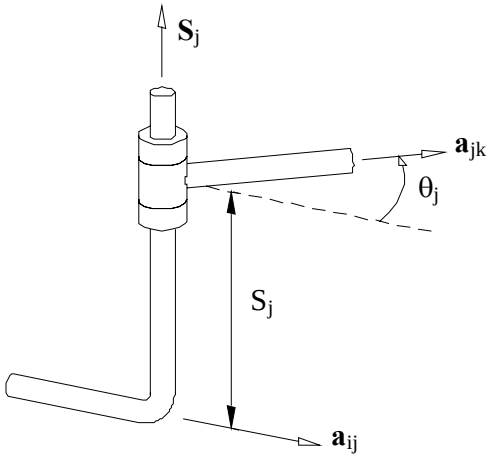

Figure 1-2. Manipulator linkage parameters for revolute joint j (Source: Crane C., Duffy J., Kinematic Analysis of Robot Manipulators, Cambridge University Press, 1998, p. 22, Figure 3.5)

Constant joint offset S_j is measured as the perpendicular distance between link a_{ij} and Link a_{jk} along the unit vector $\mathbf{S}_j$. Note that it can have a negative value. Variable joint angle θ_j is the angle between $\mathbf{a}_{ij}$ (unit vector along link ij) and $\mathbf{a}_{jk}$ (unit vector along link jk) measured in a right hand sense about $\mathbf{S}_j$.

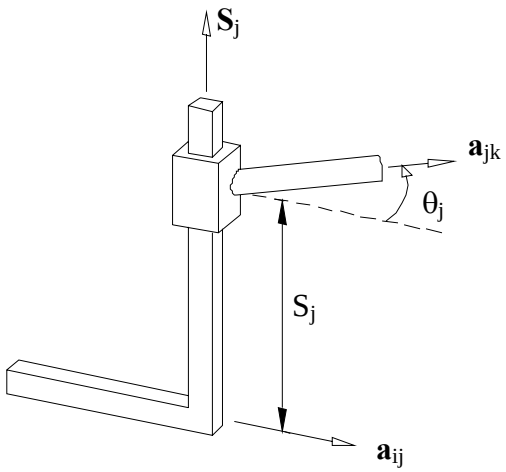

Figure 1-3. Manipulator linkage parameters for prismatic joint j (Source: Crane C., Duffy J., Kinematic Analysis of Robot Manipulators, Cambridge University Press, 1998, p. 22, Figure 3.6)

Variable joint offset S_j is measured as the perpendicular distance between link a_{ij} and link a_{jk} along the unit vector $\mathbf{S}_j$. Note that it can have a negative value. Joint angle θ_j is the angle between $\mathbf{a}_{ij}$ (unit vector along link ij) and $\mathbf{a}_{jk}$ (unit vector along link jk) measured in a right hand sense about $\mathbf{S}_j$.

1.3.3 System Topology

The Joint Architecture for Unmanned Systems hierarchy [7] is comprised of four elements: system, subsystem, node, and component/instance. Figures 1-4 and 1-5 show the interaction of the four hierarchical elements in general and with respect to the manipulator extension. A system is defined as a logical grouping of one or more subsystems allowing for cooperative advantage between the constituents. A subsystem is an independent unit consisting of a number of nodes supporting its functional

requirements and defining an operational unmanned system. The scope of our study will encompass the use of all but the system element, thus in this particular implementation, subsystem would correspond to the vehicle hosting the manipulator.

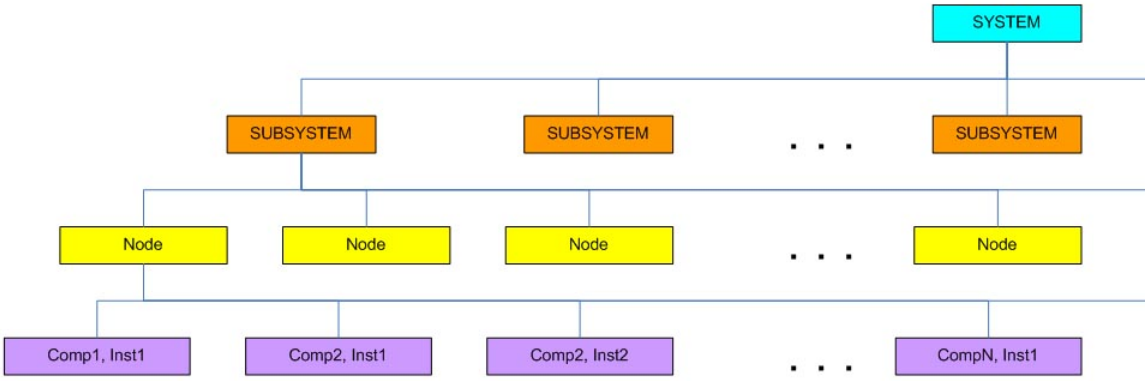

Figure 1-4. Architecture hierarchy

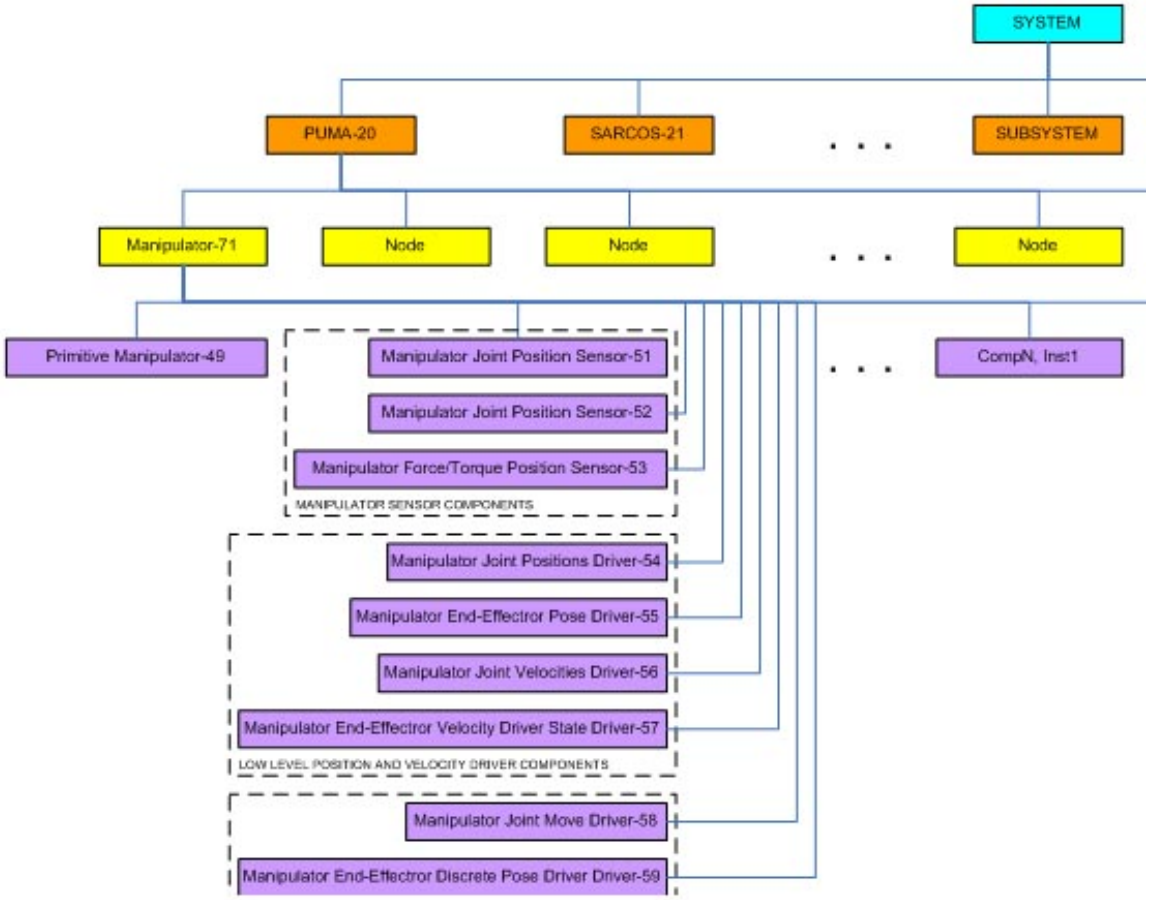

Figure 1-5. Architecture hierarchy of the manipulator control node

A node is defined as a distinct entity, composed of all the hardware and software necessary to support a well-defined computing capability. In the case of a vehicle-manipulator system, the manipulator control node contains all of the manipulator components whether they are a part of a single or multiple processors working together. Finally, a component or an instance is a cohesive software process [8]. An instance is a single occurrence running on a single node.

1.3.4 Component Definition

As it can be seen in Figure 1-5, JAUS is a hierarchical system of components with standardized interfaces. Each component has a distinct name and identification number and performs a single, cohesive function. Each component has to accept the core JAUS message set as well as the input and output messages specific to the component itself. The JAUS core message set [7] is shown in Table 1-1.

Table 1-1. The JAUS core message set

Code	Description
0x01	Set component authority
0x02	Shutdown
0x03	Standby
0x04	Resume
0x05	Reset
0x06	Set emergency
0x07	Clear emergency
0x08	Create service connection
0x09	Confirm service connection
0x0A	Activate service connection
0x0B	Suspend service connection
0x0C	Terminate service connection
0x0D	Request component control
0x0E	Release component control
0x0F	Confirm component control
0x10	Reject component control

1.3.5 Message Specification

JAUS can be defined as component based message passing architecture. As such, it uses a well defined set of messages that commence actions, exchange information, and cause events to occur. JAUS defines six classes of messages at the component level, each of which will be used in the manipulator component implementation. Table 1-2 lists the message classes [7].

Table 1-2. Segmentation of command codes by class

Message Class	Offset Range (0000h to FFFFh)
Command	0000h – 1FFFh
Query	2000h – 3FFFh
Inform	4000h – 5FFFh
Event Setup	6000h – 7FFFh
Event Notification	8000h – 9FFFh
Node Management	A000h – BFFFh
Reserved	C000h – CFFFh
User Defined Message	D000h – FFFFh

All messages are composed of a message header and the message data buffer. The header defines the message's destination node, component, instance, subsystem identification and the message's corresponding source information. The header also contains the JAUS command code, the number of bytes in the data buffer that the destination component can expect to receive, as well as the information pertaining to the message properties [8]. The header format is common to all messages as shown in Table 1-3, allowing JAUS to employ an embedded protocol providing specific information on how to handle encoding before and after the transmission. The message header, at a minimum, should be included in all messages. Figure 1-6 and Figure 1-7 show message header detail, header data format and header bit layout, respectively.

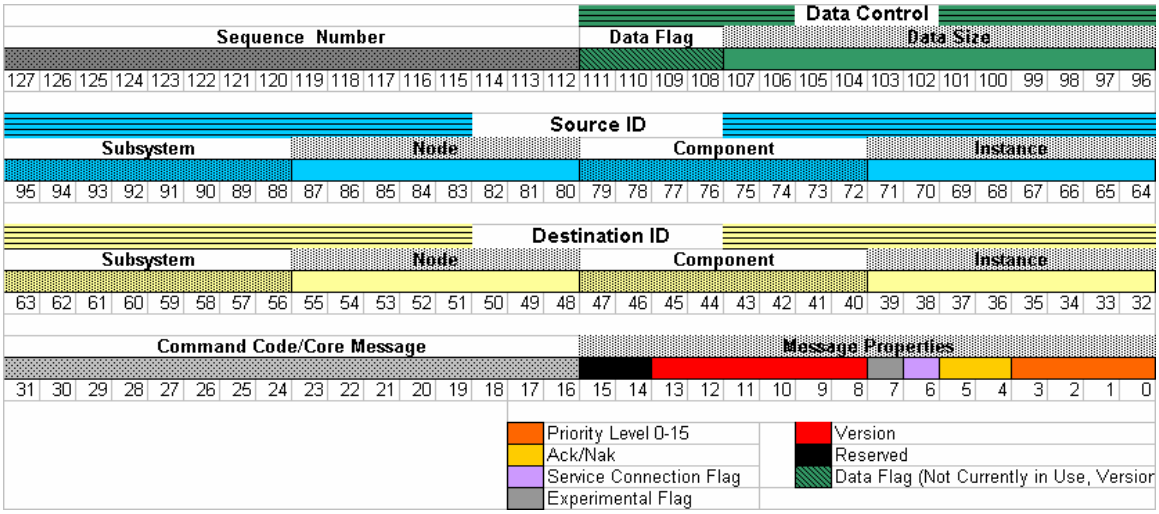

Figure 1-6. The JAUS message header detail (Source: JAUS Working Group, 2004, Joint Architecture for Unmanned Systems (JAUS): Reference Architecture Specification, Version 3.2, Volume II, The Joint Architecture for Unmanned Systems, http://www.jauswg.org, October 2004, Part 2, p.11, Figure 3.1)

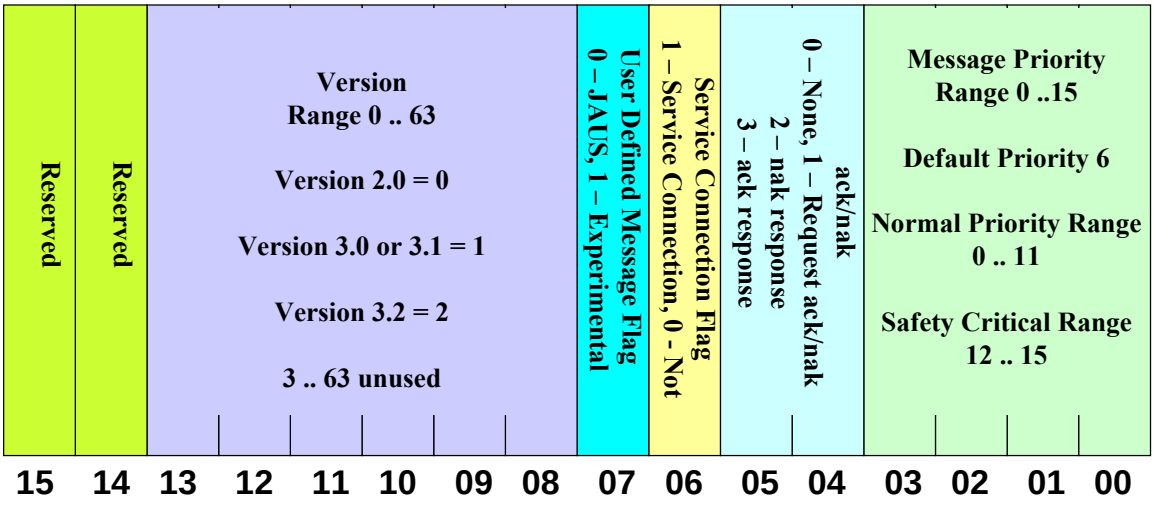

Figure 1-7. Bit layout of message properties (Source: JAUS Working Group, 2004, Joint Architecture for Unmanned Systems (JAUS): Reference Architecture Specification, Version 3.2, Volume II, The Joint Architecture for Unmanned Systems, http://www.jauswg.org, October 2004, Part 2, p.13, Figure 3.2)

The message data buffer contains packed JAUS control data, and each command code has control data used by the system to command component behavior. If a particular system or subsystem contains multiple components passing multiple messages simultaneously, it can potentially result in system delays due to bandwidth overload. In

order to prevent this, each component and its subsequent command codes must have the ability to pack and unpack (compress) the control data [8].

Each JAUS component and message is well constrained as described in the sections above. The next four chapters strive to provide adequate definitions of manipulator control components relative to the JAUS framework. Chapter 8 covers the aspects of software design and details of the component and message implementation onto the PumaA 762 robot. Finally, Chapter 9 provides the results in terms of the compatibility of the designed JAUS components and messages with the constraints outlined by JAUS. It also quantifies the performance specifications of the system using the new architecture.

Table 1-3. Message header data format

Field #	Field Description	Type	Size (Bytes)
1	Message Properties	Unsigned Short	2
2	Command Code	Unsigned Short	2
3	Destination Instance ID	Byte	1
4	Destination Component ID	Byte	1
5	Destination Node ID	Byte	1
6	Destination Subsystem ID	Byte	1
7	Source Instance ID	Byte	1
8	Source Component ID	Byte	1
9	Source Node ID	Byte	1
10	Source Subsystem ID	Byte	1
11	Data Control (bytes)	Unsigned Short	2
12	Sequence Number	Unsigned Short	2
	Total Bytes		16

CHAPTER 2
SYSTEM ANALYSIS AND OVERVIEW

Manipulator JAUS components and messages are designed to work on any serial manipulator regardless of the type or the number of joints. As a result, the choice of the test platform for implementation purposes is irrelevant. Due to the availability and its operational condition, a 6-DOF Puma 762 (Figure 2-1) robot arm was chosen for this task. Even though it was originally designed for industrial purposes, this particular manipulator can be found in many research laboratories due to its robustness and reliability.

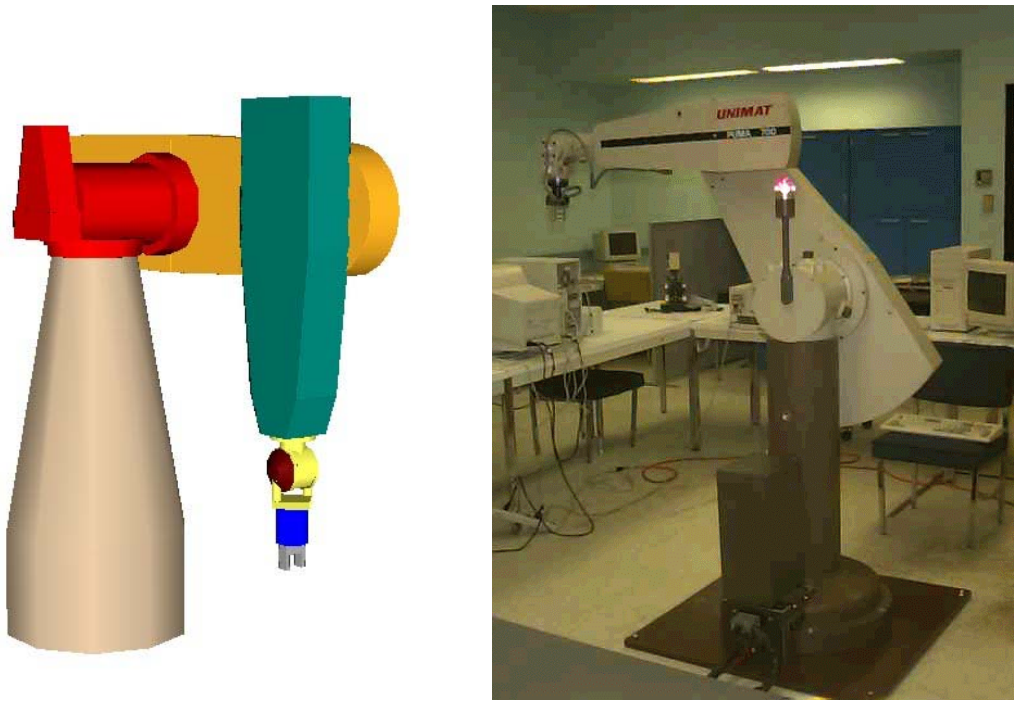

Figure 2-1. Puma 762, 6-DOF robot manipulator

The purpose of this chapter is to summarize the useful mechanical properties and describe the mathematics behind position and velocity analysis of the Puma 762 robot.

The theoretical approach and notations used in the following sections specifically follow definitions outlined by Crane and Duffy [9, 10].

2.1 Mechanism Overview

The Puma arm can be compared to a human torso, shoulder, and wrist. It consists of members connected by six revolute joints, each defining an axis about which the members rotate. The major joints are equipped with limit-switch-shutoff-systems positioned 2 degrees past the software stops providing additional safety. The Puma 762 weighs 590 kg and has a maximum static payload of 20 kg. Table 2-1 lists useful mechanical specifications of the system.

Table 2-1. Mechanical specifications of the Puma 762 robot

Joint #	1	2	3	4	5	6
Software Movement Limits (deg)	320	220	270	532	220	532
Joint Angular Resolution (deg * 10^{-3})	5.0	3.5	4.5	12.5	6.2	13.4
Encoder Index Resolution; equal to one motor revolution (deg)	4.0	2.78	3.57	6.26	6.26	13.4

2.2 Kinematic Analysis of the Puma 762 Robot

2.2.1 Notation

Figures 1-2 and 1-3 show detailed definitions of vectors and parameters used to label revolute and prismatic joints, respectively. The mechanism parameters listed in Table 2-2 are used to create a kinematic chain shown in Figure 2-2. The proper analysis of a robot manipulator mandates that a coordinate system is attached to each of the bodies. The coordinate system attached to link ij is called the ith coordinate system [9]. Its

origin is located at the intersection of vectors $\vec{S}_i$ and $\vec{a}_{ij}$; x-direction is along the vector $\vec{a}_{ij}$, and z-direction is along $\vec{S}_i$.

Table 2-2. Mechanism parameters of the Puma 762 robot

Link Length, mm	Twist Angle, deg	Jnt Offset, mm	Joint Angle, deg
$a_{12} = 0$	$\alpha_{12} = 90$		$\phi_1 =$ variable
$a_{23} = 650$	$\alpha_{23} = 0$	$S_2 = 190$	$\theta_2 =$ variable
$a_{34} = 0$	$\alpha_{34} = 270$	$S_3 = 0$	$\theta_3 =$ variable
$a_{45} = 0$	$\alpha_{45} = 90$	$S_4 = 600$	$\theta_4 =$ variable
$a_{56} = 0$	$\alpha_{56} = 90$	$S_5 = 0$	$\theta_5 =$ variable
			$\theta_6 =$ variable

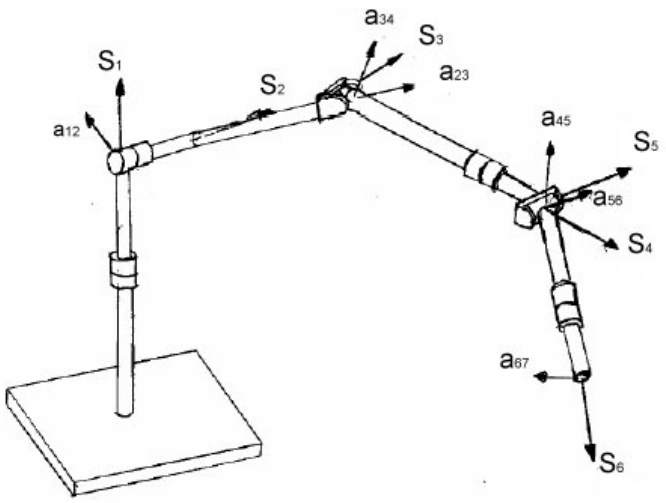

Figure 2-2. Labeled kinematic model of the Puma 762 manipulator

The transformation matrix relating two of these coordinate systems in a three dimensional space is given by Equation 2-1.

$$ {}^{i}_{j}T = \begin{bmatrix} c_i & -s_j & 0 & a_{ij} \\ s_j c_{ij} & c_j c_{ij} & -s_{ij} & -s_{ij} S_j \\ s_j s_{ij} & c_j s_{ij} & c_{ij} & c_{ij} S_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2\text{-}1) $$

The inverse of this transformation is very frequently used and is given by the following 4×4 transformation:

$$ {}^{j}_{i}T = \begin{bmatrix} c_j & s_j c_{ij} & s_j s_{ij} & -c_j a_{ij} \\ -s_j & c_j c_{ij} & c_j s_{ij} & s_j a_{ij} \\ 0 & -s_{ij} & c_{ij} & -S_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-2} $$

where a_{ij} represents a link length of a serial manipulator, s_i and c_i represent the sine and cosine of θ_i, respectively. Furthermore, s_{ij} and c_{ij} represent the sine and cosine of α_{ij}. Finally, a fixed coordinate system is defined as having its origin coincident with the origin of the first coordinate system and its axis along the vector $\vec{S}_1$. The transformation that relates the first coordinate system and the fixed is given in Equation 2-3.

$$ {}^{F}_{1}T = \begin{bmatrix} \cos\phi_1 & -\sin\phi_1 & 0 & 0 \\ \sin\phi_1 & \cos\phi_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-3} $$

General expressions determining the directions of joint vectors with respect to the first coordinate system are given in Equations 2-4 and 2-5.

$$ {}^{1}\vec{S}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad {}^{1}\vec{S}_2 = \begin{bmatrix} 0 \\ -s_{12} \\ c_{12} \end{bmatrix} \quad {}^{1}\vec{S}_3 = \begin{bmatrix} \overline{X}_2 \\ \overline{Y}_2 \\ \overline{Z}_2 \end{bmatrix} \quad {}^{1}\vec{S}_n = \begin{bmatrix} X_{n-1,n-2,\ldots,2} \\ Y_{n-1,n-2,\ldots,2} \\ Z_{n-1,n-2,\ldots,2} \end{bmatrix} \tag{2-4} $$

$$ {}^{1}\vec{a}_{12} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad {}^{1}\vec{a}_{23} = \begin{bmatrix} c_2 \\ s_2 c_{12} \\ s_2 s_{12} \end{bmatrix} \quad {}^{1}\vec{a}_{nm} = \begin{bmatrix} W_{n-1,m-1,\ldots,2} \\ -U^{*}_{n-1,m-1,\ldots,2} \\ U_{n-1,m-1,\ldots,2} \end{bmatrix} \tag{2-5} $$

where the definitions of the X, Y, Z and W, U*, U are presented in Appendix A. Using the coordinate transformation given in Equation 2-3 and expressions from Equations 2-4

and 2-5, direction vectors can be obtained in terms of the fixed coordinate system and are shown in Equations 2-6 and 2-7.

$$ {}^{F}\vec{S}_i = {}^{F}T_1 \cdot {}^{1}\vec{S}_i \tag{2-6} $$

$$ {}^{F}\vec{a}_{ij} = {}^{F}T_1 \cdot {}^{1}\vec{a}_{ij} \tag{2-7} $$

2.2.2 Forward and Reverse Position Analysis

The most basic problem in the analysis of serial manipulators is to determine the position and orientation of the end-effector for a specified set of joint angles. Following the method outlined by Crane and Duffy [9], the first step to the solution determines the transformation that relates the end-effector coordinate system to the fixed coordinate system. In the case of a 6-axis robot, the transformation in question is obtained as follows:

$$ {}^{F}_{6}T = {}^{F}_{1}T \cdot {}^{1}_{2}T \cdot {}^{2}_{3}T \cdot {}^{3}_{4}T \cdot {}^{4}_{5}T \cdot {}^{5}_{6}T \tag{2-8} $$

The coordinates of the end-effector in the fixed coordinate system can be found from Equation 2-9.

$$ {}^{F}P_{tool} = {}^{F}_{6}T \cdot {}^{6}P_{tool} \tag{2-9} $$

All of the terms in Equation 2-8, are obtained from known mechanism parameters that are listed in Table 2-2 for the Puma 762 manipulator.

A more complicated problem arises when one is trying to determine all possible joint-angle configurations for a specific end-effector position and orientation. This non-trivial solution is clearly more difficult, however due to a favorable geometry of most of the industrial manipulators, much of the analysis is simplified. The approach is referred to as the reverse kinematic analysis and begins by closing the link-loop with a hypothetical member. For a 6-R (6-revolute-joint) manipulator such as the Puma 762, this

results in a 1-degree-of-freedom 7-R spatial mechanism with the angle θ_7 known [9]. Detailed derivation of the solution to this problem is provided by Crane and Duffy [9], thus only the final equations used to obtain the values of the 6-close-the-loop parameters are shown below. The twist angle α_{71} can be calculated using Equations 2-10 and 2-11.

$$c_{71} = {}^F\vec{S}_7 \cdot {}^F\vec{S}_1 \tag{2-10}$$

$$s_{71} = \left({}^F\vec{S}_7 \times {}^F\vec{S}_1\right) {}^F\vec{S}_1 \tag{2-11}$$

Next, joint angle θ_7 is found using Equations 2-12 and 2-13.

$$c_7 = {}^F\vec{a}_{67} \cdot {}^F\vec{a}_{71} \tag{2-12}$$

$$s_7 = \left({}^F\vec{a}_{67} \times {}^F\vec{a}_{71}\right) {}^F\vec{a}_{71} \tag{2-13}$$

Parameter γ_1 is defined as the angle between the vector $\vec{a}_{71}$ and the x-axis of the fixed coordinate system, thus using a similar approach as outlined above, this angle is obtained using Equations 2-14 and 2-15.

$$\cos\gamma_1 = {}^F\vec{a}_{71} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{2-14}$$

$$\sin\gamma_1 = \left({}^F\vec{a}_{71} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right) \cdot {}^F\vec{S}_1 \tag{2-15}$$

Finally, the values of S_7, a_{71} and S_1 can be obtained and are given by Equations 2-16, 2-17, and 2-18 respectively.

$$S_7 = \frac{\left({}^F\vec{S}_1 \times {}^F\vec{P}_{6otig}\right) {}^F\vec{a}_{71}}{s_{71}} \tag{2-16}$$

$$a_{71} = \frac{\left({}^F\vec{P}_{6orig} \times {}^F\vec{S}_1\right) \cdot {}^F\vec{S}_7}{s_{71}} \tag{2-17}$$

$$S_1 = \frac{\left({}^F\vec{P}_{6orig} \times {}^F\vec{S}_7\right) \cdot {}^F\vec{a}_{71}}{s_{71}} \tag{2-18}$$

Table 2-3 shows the mechanism parameters for the newly formed closed-loop spatial mechanism.

Table 2-3. Closed-loop mechanism parameters of the Puma 762 robot

Link Length, mm	Twist Angle, deg	Jnt Offset, mm	Joint Angle, deg
$a_{12} = 0$	$\alpha_{12} = 90$	$S_1 = C.L.$	$\phi_1 =$ variable
$a_{23} = 650$	$\alpha_{23} = 0$	$S_2 = 190$	$\theta_2 =$ variable
$a_{34} = 0$	$\alpha_{34} = 270$	$S_3 = 0$	$\theta_3 =$ variable
$a_{45} = 0$	$\alpha_{45} = 90$	$S_4 = 600$	$\theta_4 =$ variable
$a_{56} = 0$	$\alpha_{56} = 90$	$S_5 = 0$	$\theta_5 =$ variable
$a_{67} = 0$	$\alpha_{67} = 90$	$S_6 = 129$	$\theta_6 =$ variable
$a_{71} = C.L.$	$\alpha_{71} = C.L.$	$S_7 = C.L.$	$\theta_7 =$ C.L.

Once all of the closed-loop parameters have been calculated, the vector-loop equation for the closed-loop Puma mechanism can be constructed and is represented using Equations 2-19:

$$S_1\vec{S}_1 + S_2\vec{S}_2 + a_{23}\vec{a}_{23} + S_4\vec{S}_4 + S_6\vec{S}_6 + S_{71}\vec{S}_{71} + a_{71}\vec{a}_{71} = \vec{0} \tag{2-19}$$

Expanding these vectors and expressing them in terms of the Set 14 of the table of direction cosines for the spatial heptagon (Appendix A) results in:

$$S_1\begin{bmatrix} 0 \\ -s_{12} \\ c_{12} \end{bmatrix} + S_2\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + a_{23}\begin{bmatrix} c_2 \\ -s_2 \\ 0 \end{bmatrix} + S_4\begin{bmatrix} X_{5671} \\ Y_{5671} \\ Z_{5671} \end{bmatrix} + S_6\begin{bmatrix} X_{71} \\ Y_{71} \\ Z_{71} \end{bmatrix} + S_7\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} + a_{71}\begin{bmatrix} c_1 \\ s_1c_{12} \\ U_{12} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2-20}$$

Further simplification is obtained by using subsidiary spatial and polar-sine, sine-cosine, and cosine laws from Equation 2-21.

$$\begin{bmatrix} X_{5671} \\ Y_{5671} \\ Z_{5671} \end{bmatrix} = \begin{bmatrix} X_{32} \\ -X_{32}^{*} \\ \overline{Z}_3 \end{bmatrix} \tag{2-21}$$

The representation in Equation 2-20 yields three equations. Further substitution of the known terms and simplification of the Z component of the equation yields Equation 2-22.

$$[S_6 Y_7 - S_7 s_{71}]c_1 + [S_6 X_7 + a_{71}]s_1 + [S_2] = 0 \tag{2-22}$$

This is an equation of the form $Ac_1 + Bs_1 + D = 0$, where A, B, and D are constants. The solution provides two possible values labeled as θ_{1a} and θ_{1b}. Corresponding values of the angle ϕ_1 can be calculated as $(\theta_{1a} - \gamma_1)$ and $(\theta_{1b} - \gamma_1)$.

Substituting the known values into the X and Y components of the Equation 2-20 yields Equations 2-23 and 2-24:

$$a_{23}c_2 + S_4 X_{32} + S_6 X_{71} + S_7 X_1 + a_{71}c_1 = 0 \tag{2-23}$$

$$-S_1 - a_{23}s_2 + a_{34}V_{32} - S_4 X_{32}^{*} + S_6 Y_{71} + S_7 Y_1 = 0 \tag{2-24}$$

Moving the terms that do not contain the unknown variables and expanding the direction-cosines results in Equations 2-25 and 2-26:

$$a_{23}c_2 + S_4 s_{2+3} = A \tag{2-25}$$

$$-a_{23}s_2 - S_4 c_{2+3} = B \tag{2-26}$$

where $A = -S_6 X_{71} - S_7 X_1 - a_{71}c_1$, $B = S_1 - S_6 Y_{71} - S_7 Y_1$, while s_{2+3} and c_{2+3} represent sine and cosine of $(\theta_2 + \theta_3)$, respectively. Squaring both sides of both Equations 2-25 and 2-26 and then adding them yields Equation 2-27.

$$a_{23}^2 + S_4^2 + 2a_{23}S_4[s_2 c_{2+3} - c_2 s_{2+3}] = A^2 + B^2 \tag{2-27}$$

where $[s_2 c_{2+3} - c_2 s_{2+3}] = \sin[\theta_2 - (\theta_2 + \theta_3)] = \sin(-\theta_3) = -s_3$. Substituting this term back into the Equation 2-27 results in Equation 2-28.

$$s_3[-2a_{23}S_4] + [a_{23}^2 + S_4^2 - A^2 - B^2] = 0 \tag{2-28}$$

This equation contains only θ_3 as the unknown. For each value of θ_1, 2 corresponding values of θ_3 are determined. Regrouping Equations 2-25 and 2-26 yields the following relationships:

$$c_2[a_{23} - S_4 s_3] + s_2[-S_4 c_3] = A \tag{2-29}$$

$$s_2[-a_{23} + S_4 s_3] + c_2[-S_4 c_3]] = B \tag{2-30}$$

Equations 2-29 and 2-30 provide a system of 2 equations and 2 unknowns. Thus a unique value of θ_2 is solved for each corresponding set of θ_1 and θ_3. The next step is to solve for θ_5 using Equation 2-31:

$$Z_{7123} = \overline{Z}_5 \tag{2-31}$$

With corresponding substitutions and expansions, this yields two values of θ_5 for each set of θ_7, θ_1, θ_3, and θ_5 as follows:

$$c_5 = -Z_{7123} \tag{2-32}$$

Solution of θ_4 is obtained from a system of 2 subsidiary spherical Equations 2-33 and 2-34.

$$X_{54} = X_{7123} \tag{2-33}$$

$$X_{54}^* = -Y_{7123} \tag{2-34}$$

where $X_{54} = \overline{X}_5 c_4 - \overline{Y}_5 s_4$ and $X_{54}^* = \overline{X}_5 s_4 + \overline{Y}_5 c_4$. Final form of the solution shown in

Equations 2-35 will result in a unique value of θ_4 for each of the 8 sets of corresponding angle values.

$$c_4 = \frac{X_{7123}}{s_5} \tag{2-35}$$

$$s_4 = -\frac{Y_{7123}}{s_5} \tag{2-36}$$

The solution of the last remaining joint angle θ_6 is obtained using the following two fundamental spherical sine and sine-cosine laws defined by Equations 2-37 and 2-38.

$$X_{43217} = s_{56} s_6 \tag{2-37}$$

$$Y_{43217} = s_{56} c_6 \tag{2-38}$$

Since α_{56} is known, the equations reduce to $s_6 = X_{43217}$ and $c_6 = Y_{43217}$. The final solution tree for the Puma 762 robot is shown below in Figure 2-3.

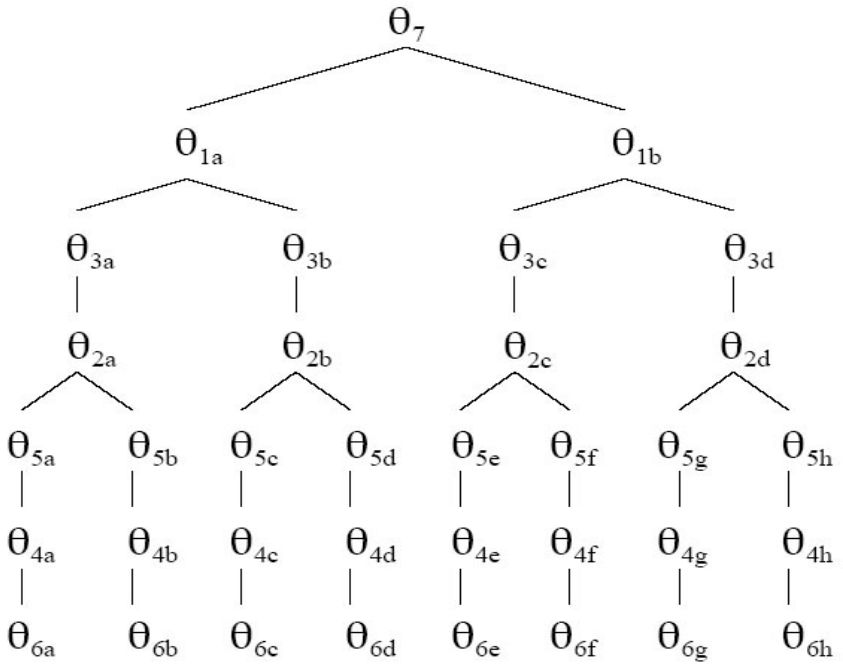

Figure 2-3. Reverse analysis solution tree for Puma 762 robot

2.2.3 Forward and Reverse Velocity Analysis

Similar to the forward position analysis, the velocity analysis assumes that the angular velocities of each of the joints are known, and as a result it determines the velocity state of the last link of the manipulator as shown in Equations 2-39.

$$\begin{bmatrix} {}^0\vec{\omega}^6 \\ {}^0\vec{v}^6 \end{bmatrix} = {}^0\omega^6 \begin{bmatrix} {}^0\vec{S}^6 \\ {}^0\vec{S}_O^6 \end{bmatrix} = {}^0\omega^6\,{}^0\$^6 = {}^1\omega^2\,{}^1\$^2 + {}^2\omega^3\,{}^2\$^3 + {}^3\omega^4\,{}^3\$^4 + {}^4\omega^5\,{}^4\$^5 + {}^5\omega^6\,{}^5\$^6 \quad (2\text{-}39)$$

where ${}^i\$^j$ are the unitized line coordinates for each of the joint-axes. From Duffy and Crane[10], this equation can be rewritten in the matrix format yielding Equation 2-40.

$$\begin{bmatrix} {}^0\vec{\omega}^6 \\ {}^0\vec{v}^6 \end{bmatrix} = {}^0\omega^6\,{}^0\$^6 = J\omega \quad (2\text{-}40)$$

where J, the Jacobian matrix, is the 6×6 matrix formed from the unitized screw coordinates, which can be written using Equation 2-41.

$$J = \begin{bmatrix} {}^0\$^1 & {}^1\$^2 & {}^2\$^3 & {}^3\$^4 & {}^4\$^5 & {}^5\$^6 \end{bmatrix} = \begin{bmatrix} {}^0\vec{S}^1 & {}^1\vec{S}^2 & {}^2\vec{S}^3 & {}^3\vec{S}^4 & {}^4\vec{S}^5 & {}^5\vec{S}^6 \\ {}^0\vec{S}_O^1 & {}^1\vec{S}_O^2 & {}^2\vec{S}_O^3 & {}^3\vec{S}_O^4 & {}^4\vec{S}_O^5 & {}^5\vec{S}_O^6 \end{bmatrix} \quad (2\text{-}41)$$

where ω is the vector of joint velocities defined by Equation 2-42.

$$\omega = \begin{bmatrix} {}^0\omega^1 & {}^1\omega^2 & {}^2\omega^3 & {}^3\omega^4 & {}^4\omega^5 & {}^5\omega^6 \end{bmatrix}^T \quad (2\text{-}42)$$

Assuming that the position analysis is completed, all members of the Jacobian are known, thus the forward velocity analysis is completed using Equation 2-39. Just as was the case with the position analysis, the reverse velocity problem becomes more complicated and individual joint velocities can be determined by simple arithmetic yielding Equation 2-43.

$$\omega = J^{-1} \cdot {}^0\omega^6 \cdot {}^0\$^6 \quad (2\text{-}43)$$

It is possible that the manipulator is driven to a configuration in which the Jacobian matrix is singular and the inverse cannot be obtained [10]. Such configurations are the topic of the following section.

2.2.4 Singularity Determination

A robot is defined to be in a singular configuration when the determinant of the Jacobian is equal to 0. Duffy and Crane [10] also define this special configuration when 1 or more of the joint-axis lines as defined in the previous section become linearly dependent to the others. This section identifies those configurations in which such conditions occur. Such configurations will cause joint velocities to rapidly increase, and there will be no feasible solutions of the manipulator to move in a desired direction. Many computational methods are developed to optimize singularity avoidance.

For practical purposes, the line coordinates of the Jacobian matrix shown in Equation 2-44 are expressed in terms of the third coordinate system. This is arithmetically the most favorable selection and allows for easy identification of singular configurations.

$$J = \begin{bmatrix} s_{2+3} & 0 & 0 & 0 & s_4 & s_5 c_4 \\ c_{2+3} & 0 & 0 & 1 & 0 & -c_5 \\ 0 & 1 & 1 & 0 & c_4 & -s_5 s_4 \\ S_2 c_{2+3} & a_{23} s_3 & 0 & 0 & S_4 c_4 & -S_4 s_5 s_4 \\ -S_2 s_{2+3} & a_{23} c_3 & 0 & 0 & 0 & 0 \\ -a_{23} c_2 & 0 & 0 & 0 & -S_4 s_4 & -S_4 s_5 c_4 \end{bmatrix} \tag{2-44}$$

The determinant of this Jacobian yields Equation 2-45.

$$\det[J] = -a_{23} S_4 (s_5)(c_3)[-a_{23} c_2 + S_4 s_{2+3}] \tag{2-45}$$

The left side of the Equation 2-45 equals zero if:

- **Wrist singularity ($s_5 = 0$):** In this case vectors S_4 and S_6 become collinear.

- **Forearm boundary singularity [11] ($c_3 = 0$):** In this case vectors S_2 and a_{23} are parallel and the elbow is fully extended.

- **Forearm interior singularity [11] ($[-a_{23}c_2+S_4s_{2+3}]=0$):** The interpretation of this term is rather complicated, but it can be said that the singularity occurs when vector S_6 is very close to the S_1 vector.

Figures 2-4, 2-5, and 2-6 show the wrist, forearm interior, and boundary singularities, respectively.

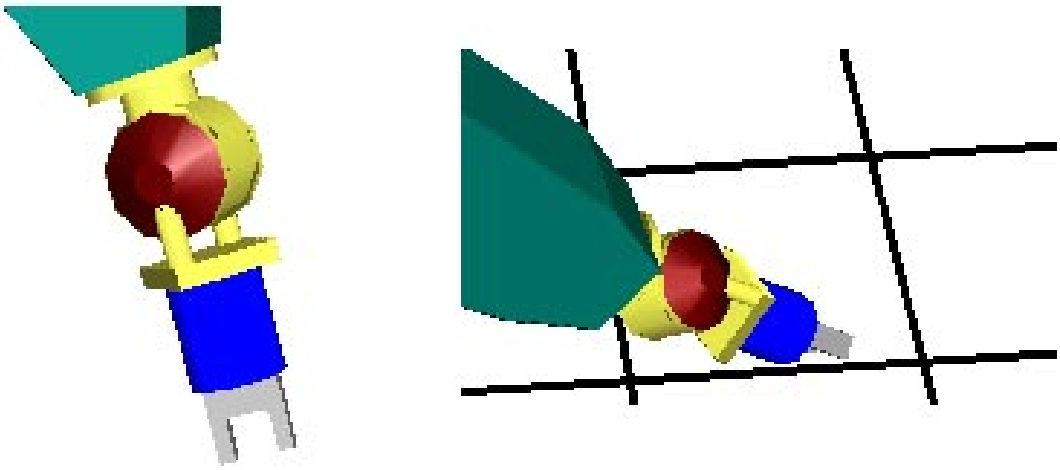

Figure 2-4. Wrist singularity of Puma 762 robot

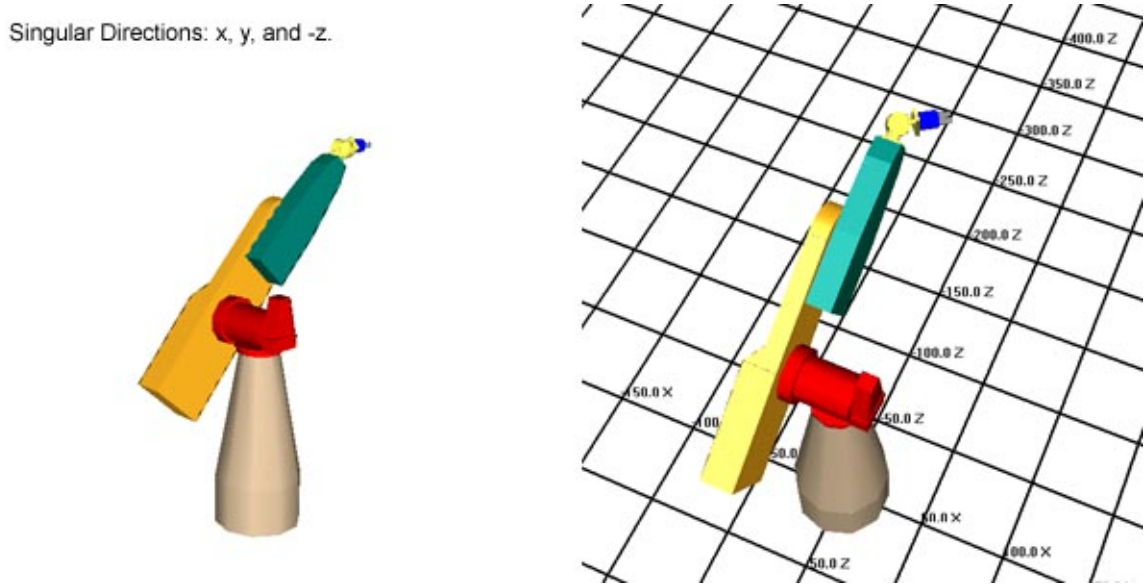

Figure 2-5. Forearm boundary singularity of Puma 762 robot

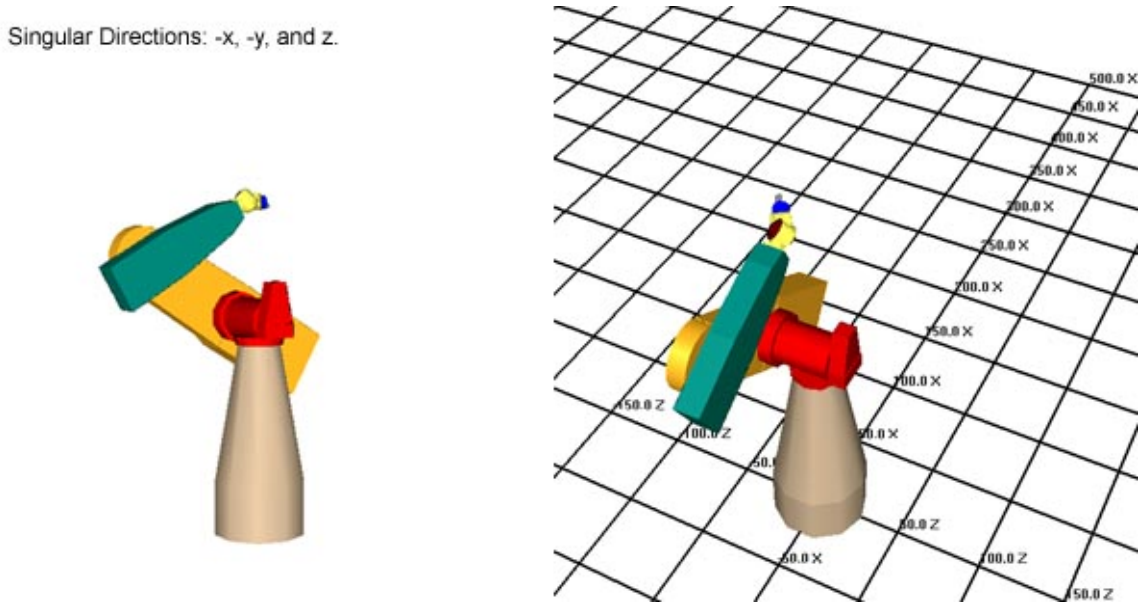

Figure 2-6. Forearm interior singularity of Puma 762 robot

The above 3 configurations require special attention and will be a matter of interest in the upcoming chapters.

2.3.5 Quaternion Representations

Crane and Duffy [9] define a real quaternion as a set of 4 real numbers written in definite order as shown in Equation 2-46.

$$q = (d, a, b, c) \tag{2-46}$$

Chapter 6 will use quaternion representation for the position and orientation of the end-effector. For that reason, this section defines those aspects of the quaternion algebra. The unit quaternion q is defined in Equation 2-47.

$$q = \cos(\theta) + \sin\theta\left(s_x i + s_y j + s_z k\right) \tag{2-47}$$

The equivalent rotation matrix is defined by Crane and Duffy [9] as the following:

$${}^{i}_{j}R = \begin{bmatrix} s_x^2(1-\cos 2\theta) + \cos 2\theta & s_x s_y(1-\cos 2\theta) - s_z \sin 2\theta & s_x s_z(1-\cos 2\theta) + s_y \sin 2\theta \\ s_x s_y(1-\cos 2\theta) + s_z \sin 2\theta & s_y^2(1-\cos 2\theta) + \cos 2\theta & s_y s_z(1-\cos 2\theta) \_ s_x \sin 2\theta \\ s_x s_z(1-\cos 2\theta) - s_y \sin 2\theta & s_y s_z(1-\cos 2\theta) + s_x \sin 2\theta & s_z^2(1-\cos 2\theta) + \cos 2\theta \end{bmatrix}$$

In summary, this chapter provides all the necessary analysis sufficient for tasks mandated by the proposed JAUS components. Equations derived in Sections 2.3.3 and 2.3.4 will become essential for component development outlined in Chapters 5, 6 and 7. Detailed information on theory covered in this chapter and more advanced robot manipulator analysis is presented by Crane and Duffy [9] and Duffy [10].

CHAPTER 3
PUMA 762 CONTROLLER SYSTEM

This chapter addresses the capabilities of an onboard controller system and ties them into the requirements outlined by the JAUS Compliance Specification (JCS) document. Before modular architecture implementation, the platform was in operational condition, allowing the user to control the motion of the individual joints using commercially available software. Such conditions required no additional changes to the current hardware configuration. For completion purposes, this chapter provides a brief summary of modifications to the original system configuration. Further emphasis is put on the functions available in the C/C++ Application Programming Interface (API) and its use in this implementation.

3.1 Overview

Native to the Puma 762 is the Val II high level programming language. Aside from being a sophisticated development tool, Val II is a complete control system. Joint Architecture for Unmanned Systems standard currently dictates the use of a C programming language for development purposes. Due to some deficiencies in the older VAL controller and lack of an adequate API, the Puma was outfitted with a Galil DMC-2100 motion controller. Unfortunately, the use of a commercially available motion controller dictates operational constraints. The JCS document mentioned above defines three levels of interoperability. This implementation satisfies inter-nodal, or level II compliance, supporting interoperation between nodes [12]. Figure 3-1 shows the simplest

representation of the overall system. Connectivity between Val and Galil is addressed in the following section.

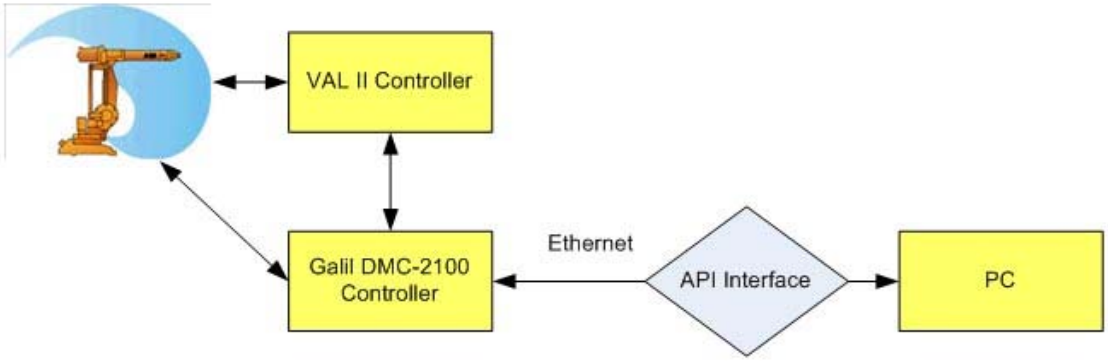

Figure 3-1. Schematic representation entire manipulator system

3.2 Reverse Engineering the Puma 762 Robot

Hardware interfacing between the Val and Galil controllers was done by Prof. Robert Bicker of the University of Newcastle upon Tyne in June of 2001. This section summarizes all the necessary modifications made to effectively outfit the system with a newer Galil controller while maintaining some of the basic Val functionality.

3.2.1 Existing Architecture

Val II is a hierarchical controller based on the LSI 11-73 microcomputer that provides a high-level trajectory and program control of the robot using the native programming language. Joint proportional-integral-derivative (PID) control is provided by dedicated digital axis servo-boards (using an Intel 8748 microcontroller) running at 1 kHz. The 12-bit digital-to-analog converter (DAC) output of the digital servo-boards is input to the individual PWM power amplifiers (PAs). Three major PAs are located on the rear-door of the controller, while three minor ones are located in the control rack. As mentioned earlier, each joint of the robot is driven by a direct-current (DC) servo-motor with integral electro-magnetic brake. Mounted directly on the end of the motor shaft are

an incremental encoder (250 lines) and a geared servo-potentiometer providing position feedback. The initial calibration process requires each joint to be driven through a small angle, sufficient to locate the nearest encoder index. This non-absolute position is then compared to the values stored in a look-up table via the corresponding analog potentiometer reading [13].

3.2.2 Encoder and Potentiometer Val Interface

The board edge connector located in the J56 slot directs encoder quadrature (A & Q), index and potentiometer outputs to respective digital servo-boards (J45-50) via a J-bus backplane. Wire-wrap terminals are provided at the rear of the backplane only on the J56, and wire-wrap connections are made to a 50-way IDC wire-wrap connector. A 50-way ribbon cable runs to a break-out box, and then single core cables are bundled to the respective connections on the Galil ICM-2900 modules, designated 1 and 2 (Figure 3-2).

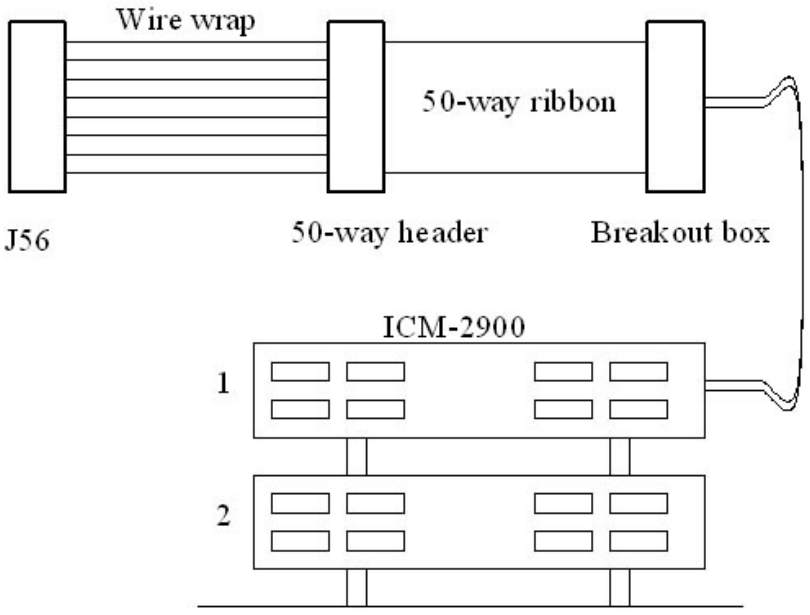

Figure 3-2. Arm signal interconnects between Val and ICM-2900 modules

3.2.3 Amplifier Digital-to-Analog Converter Signals and Control Lines

The DAC outputs on the Val controller for each axis are derived by the digital servo boards. The DAC (+/-) signals are presented to a 34-way IDC connector on the control backplane, and are linked to the power amplifier backplane via a 34-way ribbon cable (J103 => J79). Additional control lines are also present. Table 3-1 shows the pin connections between the Val controller and Galil's interconnect modules.

Table 3-1. Pin connections between Val and Galil ICM-2900 interconnect modules

VAL Signal	34/50 way pin	Wire color	ICM-2900
_teach-error	1	white	
_stop	2	white	1/14/INPUT1
Hnd-sig1	3	white	
Hnd-sig2	4	white	
Brk-on-hi	5	white	
Deadman-sw	6	white	
E-stop b	7	white	
_ox-inhibit	8	white	1/14/INPUT3
Servo E-stop	9	white	
DAC+1	11	orange	1/3/MOCMDX
DAC-1	12	blue	1/3/GND
DAC+2	13	orange	1/4/MOCMDY
DAC-2	14	blue	1/4/GND
DAC+3	15	orange	1/1/MOCMDZ
DAC-3	16	blue	1/1/GND
DAC+4	17	orange	1/2/MOCMDW
DAC-4	18	blue	1/1/GND
DAC+5	19	orange	2/3/MOCMDX
DAC-5	20	blue	2/3/GND
DAC+6	21	orange	1/4/MOCMDY
DAC-6	22	blue	1/4/GND
Remote +ve	33	white	
_Encoder fault	34	white	1/14/INPUT2

In order to retain Val safety functionality, all control lines are connected straight through via the 34-50-50-34 interconnection cabling. Stop, Encoder-fault and Ox-inhibit signals are linked to digital inputs 1-3, respectively of the ICM module 1. The amplifier enable signal is provided by a single control line BRK-ON-HI (brake on high). This is a TTL open-collector output from the LSI-11 controller, which is pulled low on VAL being initialized. A 7406 hex-inverter (with open-collector outputs) is used to drive this line low via the Galil digital output 1 as is shown in Figure 3-3.

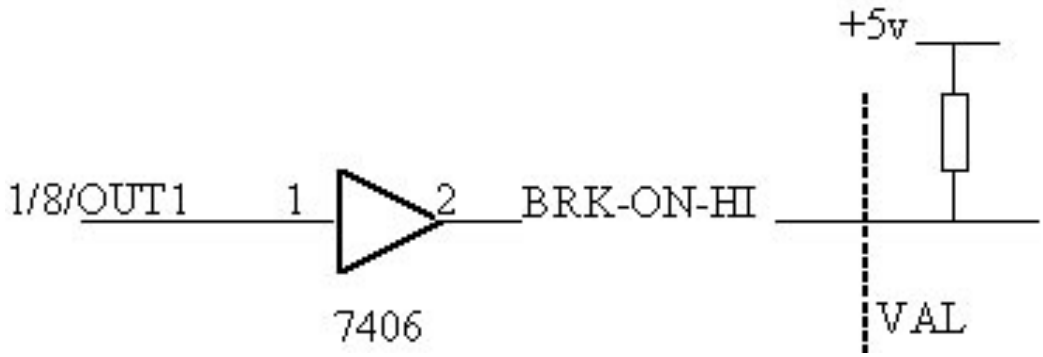

Figure 3-3. BRK-ON-HI connection diagram

3.2.4 Safety

The power amplifiers are enabled by pulling-low BRK-ON-HI using the Galil command OP1 (OutPut 1 set). Arm power is applied using the remote ON/OFF switch box (if motion control set to REMOTE) or via the front panel (if set to LOCAL). The arm will attempt to go into closed loop at this time, and it is essential to apply either the SH or MO (ServoHere, MotorOff) Galil commands prior to powering up the arm to prevent any run-away.

3.2.5 Tuning

Axis tuning was carried out sequentially, initially in the digital feedback mode (AFn=0), and subsequently in the analog feedback mode (AFn=1). Step response tests were carried out using the Servo-Design Kit software. Small and large steps (100 and

1000 counts – using encoder feedback) as well as manual tuning and auto-tuning features were used, although manual tuning was preferred. Proportional (KPn), Derivative (KDn) and Integral action gains (KIn) are set as shown in Table 3-2.

Table 3-2. Controller gain values

Axis/Joint #	KP	KD	KI
A/1	80	500	1
B/2	60	500	1
C/3	100	200	1
D/4	100	500	1
E/5	100	500	1
F/6	100	500	1

Once adequately tuned, this configuration provides a user friendly computer interface and allows for simple control of the manipulator.

3.3 Galil DMC-2100 Functionality

DMC-2100 is one of Galil's highest performance stand-alone controllers outfitted with many enhanced features including high-speed communications, non-volatile program memory, faster encoder speeds and improved cabling for noise reduction. It is designed to solve complex problems involving jogging, point-to-point positioning, vector positioning, electronic gearing, multiple move sequences, and contouring [14].

3.3.1 Command Modes

There are several of ways to command motion using the Galil controller. The requirements of this project are simple and only take advantage of two motion modes. Independent axis positioning generates an independent trajectory profile for each of the axes. The user commands either absolute or relative position and selects the values of maximum speed, acceleration and deceleration. The second, and more relevant, is independent jogging. This mode is very flexible and allows the user to change all of the

above-mentioned values during motion and specify the direction. The controller operates as a closed-loop position controller while in the jog mode. It converts the velocity profile into a position trajectory and a new position target is generated with every sample period [14].

3.3.2 Theory of Operation

This section only offers a brief overview of the operation of a motion control system. Detailed information can be found in many control theory books and controller manuals. The operation of a system (such as the DMC-2100) could be divided in three levels as follows:

- Closing the loop
- Motion profiling
- Motion programming.

Figure 3-4 shows the elements of the servo system. On the lowest level, the controller ensures that the motor follows the commanded position. This is accomplished using a feedback provided by a sensor. In the case of the Puma system, position feedback is provided by an incremental encoder.

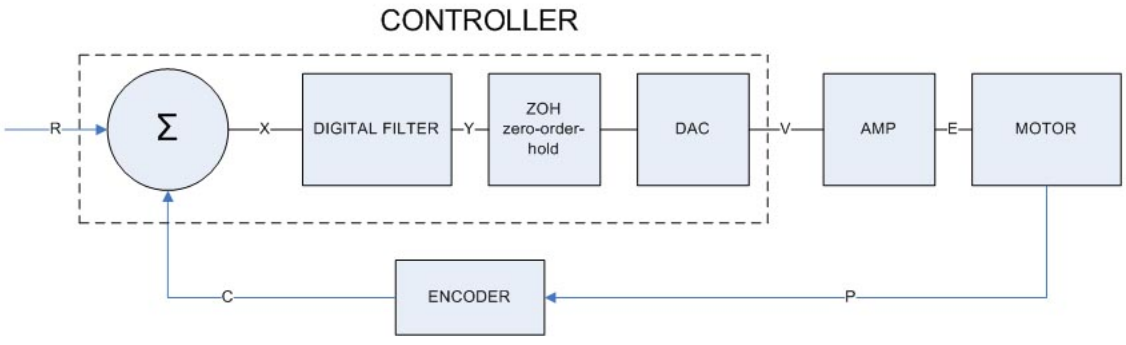

Figure 3-4. Functional elements of a motion control system

Most of the commercially available motion controllers use trajectory generators. This function describes where the motor should be at every sampling period. Finally, at

the highest level of control, the program describes the tasks such as desired distances or speed [14]. A detailed mathematical model of each of the various components presented below can be found in [14].

3.4 Galil C/C++ Application Programming Interface (API)

Engineers at Galil Motion Control Inc. have developed a set of API function calls that can be used in software development on various computer platforms. These calls provide an effective link between the Galil machine code and standard C or C++ programming languages. Manipulator JAUS implementation more specifically requires libraries running on the Linux operating system. This section lists only the calls and corresponding prototypes used in our study (Figure 3-5 through 3-9). Additional function calls and installation procedures are outlined in the interface document [15].

Function Prototype: extern LONG FAR GALILCALL DMCInitLibrary(void);
Function Description: This function must be called prior to using the library.

Figure 3-5. Initializing the Galil libraries

Function Prototype: extern LONG FAR GALILCALL DMCOpen(PCONTROLLERINFO pcontrollerinfo, PHANDLE phdmc);
Function Description: The handle to the controller is returned in the argument phdmc. Pcontrollerinfo: Structure holding information about the controller. Phdmc: Buffer to recelive the to the Galil controller to be used in all subsequent calls.

Figure 3-6. Establishing communications with the Galil Controller

Function Prototype: extern LONG FAR GALILCALL DMCClose(HANDLEDMC hdmc);
Function Description: This function closes the Ethernet connection to the controller.

Figure 3-7. Closing the connection to the Galil Controller

Function Prototype: extern LONG FAR GALILCALL DMCCommand(HANDLEDMC hdmc, PSZ pszCommand, PCHAR pchResponse, ULONG cbResponse);
Function Description: This function is used to send commands to the controller. pszCommand: The command send to the controller in the machine language. pchResponse: Buffer receiving the response data. cbResponse: Length of the buffer.

Figure 3-8. Sending commands to the Galil Controller

Function Prototype: extern LONG FAR GALILCALL DMCReset(HANDLEMC hdmc);
Function Description: This function is used to reset the controller.

Figure 3-9. Resetting the controller

The use of these calls will become more apparent in Chapter 9 as the aspects of software design on a node level are introduced.

CHAPTER 4
LOW-LEVEL MANIPULATOR CONTROL COMPONENT

This chapter defines the Primitive Manipulator (PM) component in terms of the JAUS architecture. For the purposes of easy insertion, the format used to present the information in the first section of this and the upcoming chapters, follows the latest format outlined in version 3.2 of the JAUS Reference Architecture Document. The chapter further details the application of the PM to the Puma system.

4.1 Primitive Manipulator Component

The one component in this category (JAUS ID# 49) allows for low-level command of the manipulator joint efforts. This is an open-loop command that could be used in a simple tele-operated scenario. This component is currently in version 3.2 of the JAUS Reference Architecture.

4.1.1 Definition of Coordinate Systems

A variety of platform configurations on which manipulator components can be implemented might require data to be interpreted in terms of different coordinate systems either to adequately define the geometry or allow for easier task completion.

4.1.1.1 Global coordinate system

Points that are defined in this coordinate system are defined in units of longitude, latitude, and elevation. The x-axis points North and the z-axis points downward.

4.1.1.2 Vehicle coordinate system

This coordinate system is attached to the vehicle frame. The x-axis points in the forward direction and the z-axis points downward. The y-axis is defined so as to have a

right-handed coordinate system (i.e. $\mathbf{i} \times \mathbf{j} = \mathbf{k}$ where **i**, **j**, and **k** are unit vectors along the x, y, and z coordinate axes).

4.1.1.3 Manipulator base coordinate system

In most cases, this coordinate system is attached to the vehicle base just like the vehicle coordinate system. The origin is located at the intersection of the line along the first joint axis, $\mathbf{S}_1$, and the line along the first link, $\mathbf{a}_{12}$. The z-axis is along $\mathbf{S}_1$ and the direction of the x-axis is user defined, but fixed with respect to the vehicle frame. Since the manipulator base coordinate system and the vehicle coordinate system are both attached to the vehicle base, the transformation matrix that describes the relative position and orientation of these two coordinate systems will be constant. In some cases where one manipulator may be attached to the end of another manipulator, this transformation matrix would vary.

4.1.1.4 End-effector coordinate system

This coordinate system is attached to the last link of the serial manipulator. The origin is located at the point that is a distance S_n (S_6 for a six axis manipulator) along the last joint axis vector ($\mathbf{S}_6$ for a six axis manipulator) from the intersection of the lines along the last joint axis and the preceding link axis ($\mathbf{S}_6$ and $\mathbf{a}_{56}$ for a six axis manipulator). The Z axis is along the $\mathbf{S}_n$ vector. The direction of the x-axis is user defined, but of course must be perpendicular to the z-axis. The y-axis is defined by the right-hand rule. Note that the distance S_n is returned by the Primitive Manipulator component via the Report Manipulator Specifications Message.

4.1.2 Component Function

This component is concerned only with the remote operation (open-loop control) of a single manipulator system. The manipulator may or may not have joint measurement

sensors that would provide joint position and joint velocity information. Without joint measurement sensors, the operator can only send joint motion efforts as commands to the manipulator. This component does not use any joint angle or velocity feedback from the manipulator. It only receives the desired percentage of maximum joint effort for each joint as an input.

4.1.3 Associated Messages

The PM accepts the core input and output messages listed in Table 1-1. The set of user defined input and output messages specific to this component are listed below. Detailed definitions are provided in Section 4.2.6.

- Set Joint Effort
- Query Manipulator Specifications
- Query Joint Efforts
- Report Manipulator Specifications
- Report Joint Efforts

4.1.4 Component Description

This component is the low-level interface to a manipulator arm and is in many respects similar to the Primitive Driver component for mobility of the platform. When queried, the component will reply with a description of the manipulator's specification parameters, axes range of motion, and axes velocity limits. The notations used to describe these data are documented in many popular text books on robotics and were previously presented in Chapter 2. The mechanism specification parameters as reported by the Report Manipulator Specifications Message consist of the number of joints, the type of each joint (either revolute or prismatic), the link description parameters for each link (link length and twist angle, Figures 1-1 to 1-3), the constant joint parameter value

(offset for a revolute joint, Figure 1-2), and joint angle for a prismatic joint (Chapter 1, Figure 1-3). The minimum and maximum allowable value for each joint and the maximum velocity for each joint follow this information. Motion of the arm is accomplished via the Set Joint Effort message (Figure 4-1). In this message, each actuator is commanded to move with a percentage of maximum effort.

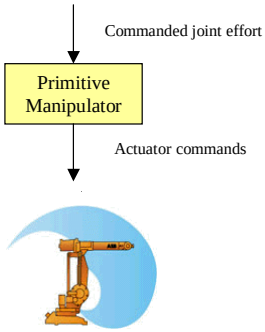

Figure 4-1. Joint effort provides basic manipulator mobility

4.1.5 Input and Output Messages

This section provides detailed specifications of user defined messages. It covers data types used for each of the parameters, as well as the upper and lower limits. This information becomes very important for software development purposes and is further covered in Chapter 8.

4.1.5.1 Code 0601h: Set Joint Effort

Field #1 in Table 4-1 indicates the number of joint effort commands contained in this message. This message sets the desired joint effort values.

Table 4-1. Set Joint Effort message parameters

Field #	Name	Type	Units	Interpretation
1	Num Joints	Byte	N/A	1 ... 255; 0 is Reserved
2	Joint 1 –effort	Short int	Percent	Scaled integer Lower Limit = -100% Upper Limit = +100%
3 ... n	…			
n + 1	Joint n –effort	Short int	Percent	see field 2

4.1.5.2 Code 2600h: Query Manipulator Specifications

This message shall cause the Primitive Manipulator component to reply to the requestor with a Code 4600h: Report Manipulator Specifications message.

4.1.5.3 Code 2601h: Query Joint Effort

This message shall cause the receiving component to reply to the requestor with a Code 4601h: Report Joint Efforts message.

4.1.5.4 Code 4600h: Report Manipulator Specifications

This message provides the specifications of the manipulator including the number of joints, the link lengths, twist angles, offset or joint angles, minimum and maximum value for each joint, and minimum and maximum speed for each joint.

4.1.5.5 Code 4601h: Report Joint Effort

This message provides the receiver with the current values of the commanded joint effort. The message data and mapping of the presence vector for the Report Joint Efforts message are identical to Code 0601h: Set Joint Effort.

Table 4-2. Report Manipulator Specifications parameters

Field #	Name	Type	Units	Interpretation
1	Number of Joints	Byte	N/A	1 … 255 0 is Reserved For a value of n the message data area size in bytes is: (36 + 13(n-1)) bytes.
2	Joint n Type	Byte	N/A	Joint type of the last joint of the manipulator. 1 = revolute, 2 = prismatic
3	Joint n – Offset/Joint Angle	Unsigned Short	rad or mm	Prismatic joint, value × 10^{-3} radians. Revolute joint, mm
4	Joint n – Min value	Unsigned Short	rad or mm	Prismatic joint, mm Revolute joint, value × 10^{-3} radians.
5	Joint n – Max value			
6	Joint n – Max velocity	Unsigned Short	rad/s or mm/s	Prismatic joint, mm/sec Revolute joint, value × 10^{-3} radians/sec.
7	manipulator coordinate sys. x	Integer	m	x coordinate of origin of manipulator coordinate system measured with respect to vehicle coordinate system Scaled integer: Lower limit = -30 m Upper limit = +30 m
8	manipulator coordinate sys. y	Integer	m	y coordinate of origin of manipulator coordinate system measured with respect to vehicle coordinate system
9	manipulator coordinate sys. z	Integer	m	z coordinate of origin of manipulator coordinate system measured with respect to vehicle coordinate system
10	d component of unit quaternion q	Integer	N/A	quaternion q defines the orientation of the manipulator coordinate system measured with respect to the vehicle coordinate system Scaled integer: Lower limit = -1 Upper limit = +1

Table 4-2. Continued

Field #	Name	Type	Units	Interpretation
12	b component of unit quaternion q	Integer	N/A	see field 10
13	c component of unit quaternion q	Integer	N/A	see field 10
14	Joint 1 Type	Byte	N/A	1 = revolute, 2 = prismatic
14a	Link a_{12} – Link Length	Unsigned Short	mm	Link Length
14b	Link a_{12} – Twist Angle	Unsigned Short	rad	Twist Angle - value × 10^{-3} radians
14c	Joint 1 – Offset/Joint Angle	Unsigned Short	rad or mm	Prismatic joint, value × 10^{-3} radians. Revolute joint, mm
14d	Joint 1 – Min value	Unsigned Short	rad or mm	Prismatic joint, mm
14e	Joint 1 – Max value			Revolute joint, value × 10^{-3} radians.
14f	Joint 1 – Max velocity	Unsigned Short	rad/s or mm/s	Prismatic joint, mm/sec Revolute joint, value × 10^{-3} radians/sec.
…				
n-1	Joint (n-1) Type	Byte	N/A	1 = revolute, 2 = prismatic
(n-1)a	Link $a_{(n-1)n}$ – Link Length	Unsigned Short	mm	Link Length
(n-1)b	Link $a_{(n-1)n}$ – Twist Angle	Unsigned Short	rad	Twist Angle - value × 10^{-3} radians
(n-1)c	Joint (n-1) – Offset/Joint Angle	Unsigned Short	rad or mm	Prismatic joint, value × 10^{-3} radians. Revolute joint, mm
(n-1)d	Joint (n-1) – Min value	Unsigned Short	rad or mm	Prismatic joint, mm
(n-1)e	Joint (n-1) – Max value			Revolute joint, value × 10^{-3} radians.
(n-1)f	Joint (n-1) – Max velocity	Unsigned Short	rad/s	Prismatic joint, mm/sec Revolute joint, value × 10^{-3} radians/sec.

4.2 Primitive Manipulator Applications to the Puma system

As mentioned earlier, the primary function of this component is to allow for tele-op commands of joint efforts. This implementation interprets joint efforts as the percentages of maximum velocity. Thus, a Set Joint Effort message commanding values of [50, 50, 50, 50, 50, 50] would in effect rotate each of the joints at fifty percent of the maximum velocity in the positive direction. Note that the interpretation of joint efforts can vary depending on the type of the controller provided. Since the component assumes that no sensor feedback is provided, it is possible to drive the robot into an unfavorable state triggering a fatal Val II error. Either one of the joints reaching a hardware limit, or driving a manipulator to a singular configuration would cause Val to respond in such a way. In order to prevent this from happening during the tasks requiring closed-loop control, this implementation takes advantage of the onboard position ad velocity sensor components covered in the following chapter. Computer logic and software design of this and all other components will be the topic of Chapter 8.

CHAPTER 5
MANIPULATOR SENSOR COMPONENTS

This chapter describes the components that when queried, return instantaneous joint position, velocity, and force or torque information. Note that a particular manipulator might only have one or two measurement sensors onboard. Consequently, the implementation of all three components covered in this chapter is seldom necessary. Moreover, the JAUS Reference Architecture is not concerned with the type of the sensors used, but rather with the format in which the measured information is exchanged with the component requesting it.

5.1 Manipulator Joint Position Sensor Component

5.1.1 Component Function

The function of the Manipulator Joint Position Sensor Component (MJPS) is to report the values of manipulator joint parameters when queried.

5.1.2 Associated Messages

The MJPS (JAUS ID# 51) accepts the core input and output messages (Chapter 1, Table 1-1). The set of user defined input and output messages specific to this component are:

- Query Joint Positions
- Report Joint Positions.

Detailed definitions are provided in Section 5.1.4.

5.1.3 Component Description

The Report Joint Positions message provides the instantaneous joint positions. The positions are given in radians for revolute and in meters for prismatic joints. The component is shown in Figure 5-1.

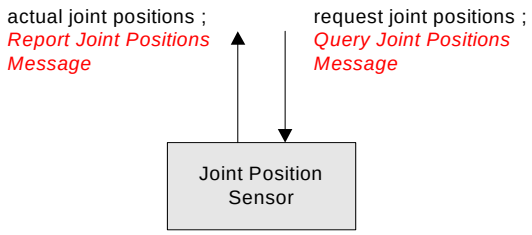

Figure 5-1. Joint position sensor component

5.1.4 Input and Output Messages

Similar to Section 4.1.5, this section and Sections 5.2.4 and 5.3.4, are used to provide detailed specifications of user defined messages. They cover data types and value limits for each of the parameters.

5.1.4.1 Code 0602h: Set Joint Positions message

Set Joint Positions message is not defined by this component but it is included as a reference.

Table 5-1. Set Joint Positions message parameters

Field #	Name	Type	Units	Interpretation
1	Number of Joints	Byte	N/A	1 ... 255 0 is Reserved
2	Joint 1 –position	Int	rad or m	Scaled integer: If revolute joint, Lower limit = -8π rad Upper limit = +8π rad If prismatic joint, Lower limit = -10 m Upper limit = +10 m
3 ... n	…			
n + 1	Joint n –position	Int	rad or m	see field 2

5.1.4.2 Code 2602h: Query Joint Positions message

This message shall cause the receiving component to reply to the requestor with a Code 4602h: Report Joint Positions message.

5.1.4.3 Code 4602h: Report Joint Positions message

This message provides the receiver with the current values of the joint positions. The message data for the Report Joint Positions message is identical to Code 0602h: Set Joint Positions.

5.2 Manipulator Joint Velocity Sensor Component

5.2.1 Component Function

The function of the Manipulator Joint Velocity Sensor (MJVS) is to report the values of instantaneous joint velocities when queried.

5.2.2 Associated Messages

The MJVS (JAUS ID# 52) accepts the core input and output messages (Chapter 1, Table 1-1). The set of user defined input and output messages specific to this component are:

- Query Joint Velocities
- Report Joint Velocities.

Detailed definitions are provided in Section 5.2.4.

5.2.3 Component Description

The Report Joint Velocities message provides the instantaneous joint velocities. The velocities are given in radians/sec for revolute and in meters/sec for prismatic joints. The component is depicted in Figure 5-2.

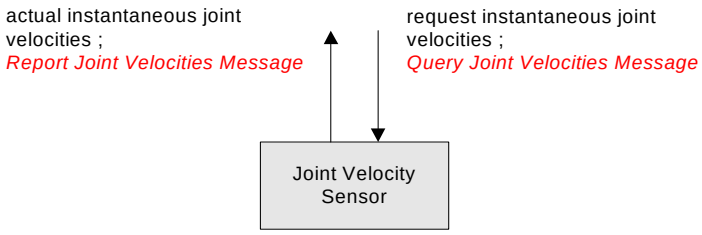

Figure 5-2. Joint velocity sensor component

5.2.4 Input and Output Messages

5.2.4.1 Code 0603h: Set Joint Velocities message

Set Joint Velocities message is not defined by this component but it is included for reference purposes.

Table 5-2. Set Joint Velocities message parameters

Field #	Name	Type	Units	Interpretation
1	Num Joints	Byte	N/A	1 ... 255 0 is Reserved
2	Joint 1 –velocity	int	rad/sec or m/sec	If revolute joint, Lower limit = -10π rad/sec Upper limit = +10π rad/sec If prismatic joint, Lower limit = -5 m/sec Upper limit = +5 m/sec
3 ... n	…			
n + 1	Joint n –velocity	int	rad/sec or m/sec	see field 2

5.2.4.2 Code 2603h: Query Joint Velocities message

This message shall cause the receiving component to reply to the requestor with a Code 4603h: Report Joint Velocities message.

5.2.4.3 Code 4603h: Report Joint Velocities message

This message provides the receiver with the current values of the joint velocities. The message data for the Report Joint Velocities message is identical to Code 0603h: Set Joint Velocities.

5.3 Manipulator Joint Force/Torque Sensor Component

5.3.1 Component Function

The function of the Manipulator Joint Force/Torque Sensor (MJFTS) is to report the values of instantaneous torques (for revolute joints) and forces (for prismatic joints) that are applied at the individual joints of the manipulator kinematic model when queried.

5.3.2 Associated Messages

The MJFTS (JAUS ID# 53) accepts the core input and output messages (Chapter 1, Table 1-1). The set of user defined input and output messages specific to this component are listed below, while detailed definitions are provided in Section 5.3.4:

- Query Joint Force/Torques
- Report Joint Force/Torques

5.3.3 Component Description

The Joint Force/Torque Sensor component provides the instantaneous joint forces or torques acting on each joint of the manipulator kinematic model. Forces are returned for prismatic joints in units of Newton's (N). Torques is returned for revolute joints in units of Newton-meters (Nm).

5.3.4 Input and Output Messages

5.3.4.1 Code 2605: Query Joint Force/Torques

This message shall cause the receiving component to reply to the requestor with a Code 4605h: Report Joint Force/Torques message.

5.3.4.2 Code 4605h: Report Joint Force/Torques

This message replies to the requestor with the values of current joint forces or torques for prismatic and revolute joints, respectively.

Table 5-3. Report Joint Force/Torque message parameters

Field #	Name	Type	Units	Interpretation
1	Num Joints	Byte	N/A	1 ... 255 0 is Reserved
2	Joint 1 –force or torque	int	N or Nm	If revolute joint Scaled integer: Lower limit = -1000 Nm Upper limit = +1000 Nm If prismatic joint Scaled integer: Lower limit = -500 N Upper limit = +500 N
3 ... n	…			
n + 1	Joint n –force or torque	int	N or Nm	see field 2

5.4 Sensor Component Applications to the Puma System

Each joint onboard the Puma 762 manipulator uses an incremental encoder for position feedback (Chapter 3, Section 3.2.1). This information is obtained from the Galil controller in encoder counts and is converted to radians. Similarly, the velocity is reported in encoder counts per second and converted into radians per second to meet JAUS specifications. In many instances, the joint data is converted back to either encoder counts or degrees for the purposes of more meaningful interpretation.

The position sensor component implemented on the Puma system has additional functionality (Section 5.1.1). This involves monitoring joint movement and assuring that no set software limits have been breached. It further uses the acquired information to monitor joint angle values approaching singularity configurations (Chapter 2, Section 2.3.4). Actions taken if either occurs are discussed in Chapter 8. The use of MJPS and MJVS components will become more apparent in the next chapter dealing with closed-loop control of joint positions and velocities.

CHAPTER 6
MANIPULATOR LOW-LEVEL POSITION AND VELOCITY DRIVER COMPONENTS

These components take as inputs the desired joint positions or velocities, or the desired end-effector pose or velocity state. They further use platform specific information provided by the PM component, as well as the sensor information obtained from the MJPS and MJVS components to perform closed-loop position and velocity control. It should be noted that in most implementations, these components and the Primitive Manipulator component would be embedded in the same node that will facilitate the control process. The implementation of the low level drivers to the Puma platform becomes nontrivial and is addressed in Section 6.5.

6.1 Manipulator Joint Positions Driver Component

6.1.1 Component Function

The function of the Manipulator Joint Positions Driver (MJPD) is to perform closed-loop joint position control.

6.1.2 Associated Messages

The MJPD (JAUS ID# 54) accepts the core input and output messages (Chapter 1, Table 1-1). User defined messages that are received or sent by this component were defined in previous chapters and are:

- Set Joint Positions
- Report Manipulator Specifications
- Report Joint Efforts

- Report Joint Positions
- Set Joint Effort.

6.1.3 Component Description

The inputs are the desired joint values, current joint angles, and the manipulator specifications report. The output is the joint effort level that is sent to the Primitive Manipulator component. Figure 6-1 shows the MJPD component.

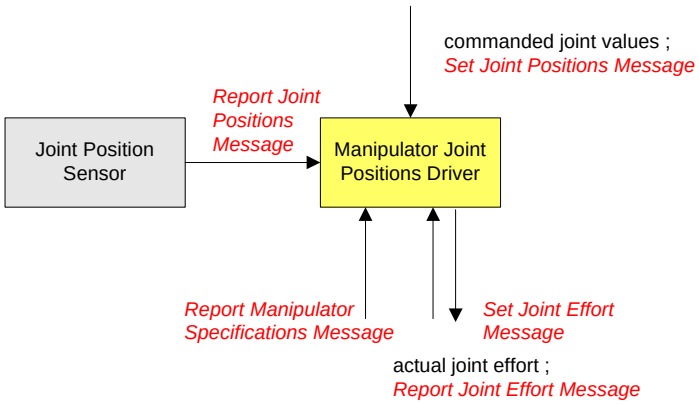

Figure 6-1. Manipulator Joint Positions Driver component

6.2 Manipulator End-Effector Pose Driver Component

6.2.1 Component Function

The function of the Manipulator End-Effector Pose Driver (MEEPD) is to perform closed-loop position and orientation control of the end-effector.

6.2.2 Associated Messages

The MEEPD (JAUS ID# 55) accepts the core input and output messages (Chapter 1, Table 1-1). Some of the user defined messages that are received or sent by this component were defined in previous chapters, while others are defined in Section 6.2.4. The following messages are associated with this component:

- Set Tool Point

- Set End-Effector Pose
- Query Tool Point
- Report Manipulator Specifications
- Report Joint Efforts
- Report Joint Positions
- Set Joint Effort
- Report Tool Point.

6.2.3 Component Description

This component performs closed-loop position and orientation control of the end-effector. The input is the desired position and orientation of the end-effector specified in the vehicle coordinate system, the current joint angles, and the data from the manipulator specification report. The output is the joint effort level that is sent to the Primitive Manipulator component. The component is shown in Figure 6-2.

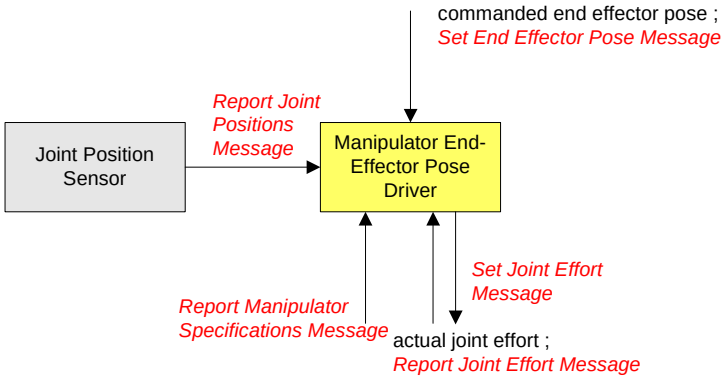

Figure 6-2. Manipulator End-Effector Pose Driver component

6.2.4 Input and Output Messages

6.2.4.1 Code 0604h: Set Tool Point message

This message specifies the coordinates of the end-effector tool point in terms of the coordinate system attached to the end-effector. For a 6-axis robot, this coordinate system

is defined by having its origin located at the intersection of the $\mathbf{S}_6$ joint axis vector and the user defined link vector $\mathbf{a}_{67}$. The z-axis of the coordinate system is along $\mathbf{S}_6$ and the –x-axis is along the $\mathbf{a}_{67}$ vector.

Table 6-1. Set Tool Point message parameters

Field #	Name	Type	Units	Interpretation
1	x coordinate of tool point	Int	m	Scaled integer: Lower limit = -15 m Upper limit = +15 m
2	y coordinate of tool point	Int	m	See field 1
3	z coordinate of tool point	Int	m	See field 1

6.2.4.2 Code 0605h: Set End-Effector Pose message

This message defines the desired end-effector position and orientation. The coordinates of the tool point are defined in terms of the vehicle coordinate system. The orientation of the end-effector is defined by a unit quaternion (d ; a, b, c) which specifies the axis and angle of rotation that was used to establish the orientation of the end-effector coordinate system with respect to the vehicle coordinate system.

Table 6-2: Set End-Effector Pose message parameters

Field #	Name	Type	Units	Interpretation
1	x component of tool point	int	m	Scaled integer: Lower limit = -30m Upper limit = +30m
2	y component of tool point	int	m	see field 1
3	z component of tool point	int	m	see field 1
4	d component of unit quaternion q	int	N/A	Scaled integer: Lower limit = -1 Upper limit = +1
5	a component of unit quaternion q	int	N/A	see field 4
6	b component of unit quaternion q	int	N/A	see field 4
7	c component of unit quaternion q	int	N/A	see field 4

6.2.4.3 Code 2604h: Query Tool Point

This message shall cause the receiving component to reply to the requestor with a Code 4604h: Report Tool Point message.

6.2.4.4 Code 4604h: Report Tool Point

This message provides the receiver with the current values of the joint positions. The message data for the Report Joint Positions message is identical to Code 0604h: Set Tool Point.

6.3 Manipulator Joint Velocities Driver Component

6.3.1 Component Function

The function of the Manipulator Joint Velocities Driver (MJVD) is to perform closed-loop joint velocity control.

6.3.2 Associated Messages

The MJVD (JAUS ID# 56) accepts the core input and output messages (Chapter 1, Table 1-1). User defined messages that are received or sent by this component were defined in previous chapters and are:

- Set Joint Velocities
- Report Manipulator Specifications
- Report Joint Effort
- Report Joint Velocities
- Set Joint Effort

6.3.3 Component Description

The input consists of the desired instantaneous joint velocities, the current joint velocities (rad/s for revolute and m/s for prismatic joints, respectively), and the data from the manipulator specification report (Puma platform specific parameters, including the orientation of the manipulator base coordinate system). The output is the joint effort level that is sent to the Primitive Manipulator component. The Manipulator Joint Velocities Driver component is shown in Figure 6-3.

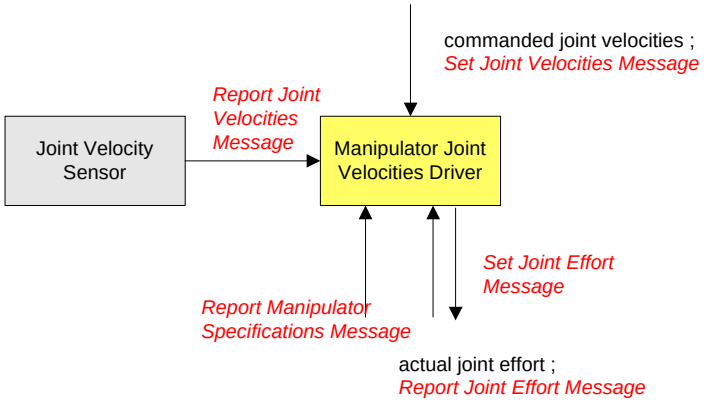

Figure 6-3. Manipulator Joint Velocities Driver component

6.4 Manipulator End-Effector Velocity State Driver Component

6.4.1 Component Function

The function of the Manipulator End-Effector Velocity State Driver (MEEVD) is to perform closed-loop velocity control of the end effector.

6.4.2 Associated Messages

The MEEPD (JAUS ID# 57) accepts the core input and output messages (Chapter 1, Table 1-1). Some of the user defined messages that are received or sent by this component were defined in previous chapters, while Set End-Effector Velocity State message is defined in Section 6.4.4. The following messages are associated with this component:

- Set End-Effector Velocity State
- Report Manipulator Specifications
- Report Joint Effort
- Report Joint Positions
- Report Joint Velocities
- Set Joint Effort

6.4.3 Component Description

The input is the desired end-effector velocity state, specified in the vehicle coordinate system or the end-effector coordinate system, the current joint positions and joint velocities, and the data from the manipulator specifications report. The output is the joint effort level that is sent to the Primitive Manipulator component. Figure 6-4 depicts this component.

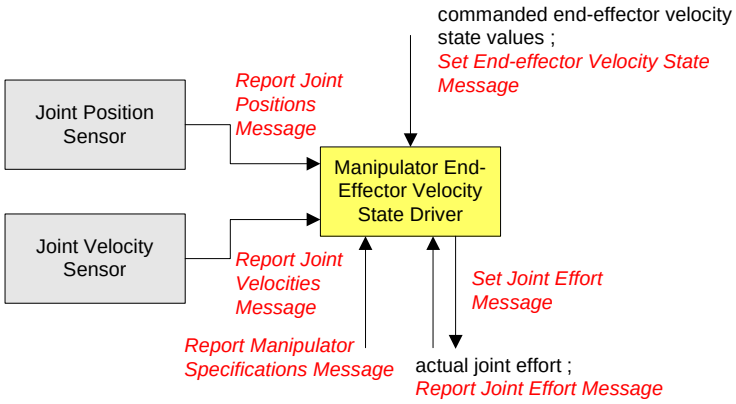

Figure 6-4. Manipulator End-Effector Velocity State Driver component

6.4.4 Code 0606h: Set End-Effector Velocity State message

The velocity state of body B measured with respect to body A is defined as the angular velocity of body B with respect to body A, ${}^{A}\boldsymbol{\omega}^{B}$, and the linear velocity of the point in body B that is coincident with the origin of the reference frame measured with respect to body A, ${}^{A}\mathbf{v}_{0}^{B}$. From these parameters, the velocity of any point in body B can be determined from the equation ${}^{A}\mathbf{v}_{P}^{B} = {}^{A}\mathbf{v}_{0}^{B} + {}^{A}\boldsymbol{\omega}^{B} \times \mathbf{r}_{0\to P}$. Here ${}^{A}\mathbf{v}_{P}^{B}$ is the velocity of some point P, that is embedded in body B, measured with respect to body A and $\mathbf{r}_{0\to P}$ represents the vector from the origin of the reference frame to point P, i.e. the coordinates of point P. In this application, body B is the end-effector, and body A is ground. The

reference coordinate system is embedded in ground, but is aligned with either the end-effector coordinate system or the vehicle coordinate system at this instant.

Table 6-3. Set End-Effector Velocity State message parameters

Field #	Name	Type	Units	Interpretation
1	Coordinate system definition	byte	N/A	1 = reference coord. system aligned with vehicle coord. sys. 2 = reference coord. system aligned with end-effector coord. sys.
2	x component of angular velocity ${}^{A}\boldsymbol{\omega}^{B}$	int	rad/sec	Scaled integer: Lower limit = -20 π rad/sec Upper limit = +20 π rad/sec
3	y component of angular velocity ${}^{A}\boldsymbol{\omega}^{B}$	int	rad/sec	see field 2
4	z component of angular velocity ${}^{A}\boldsymbol{\omega}^{B}$	int	rad/sec	see field 2
5	x component of the linear velocity ${}^{A}\mathbf{v}_{0}^{B}$	int	m/sec	Scaled integer: Lower limit = -10 m/sec Upper limit = +10 rad/sec
6	y component of the linear velocity ${}^{A}\mathbf{v}_{0}^{B}$	int	m/sec	see field 5
7	z component of the linear velocity ${}^{A}\mathbf{v}_{0}^{B}$	int	m/sec	see field 5

6.5 Applications of the Low-Level Driver Components to the Puma System

The complexity of low-level component implementation revolves around the concept of joint effort. Even though the Galil controller is inherently capable of position and velocity control, the interpretation of effort as velocity mandates the use of independent jogging mode to perpetuate joint motion (Chapter 3, Section 3.3.1). Therefore this section strives to relate this interpretation to each individual component's requirements.

The Manipulator Joint Positions Driver component receives commanded joint positions and constantly queries the MJPS for current joint values. As this information is rapidly updated, the corresponding joint efforts are determined using Equation 6-1.

$$effort[n] = \frac{\left(position_{des}[n] - position_{curr}[n]\right) \cdot k_p[n]}{velocity_{\max}[n]} \tag{6-1}$$

where n is the joint number and k_p is a proportional constant whose values are determined experimentally for each of the joints and depend on the chosen values of maximum speed, acceleration and deceleration. Since joint efforts can vary from -100% to 100%, computed efforts are clipped when received by the Primitive Manipulator. For instance, if commanded joint position is -3000 encoder counts, and the current value is 6000 encoder counts, then the PM will cause that joint to move at a 100% of the maximum velocity in the negative direction. As the position gap closes, the difference becomes smaller causing the effort to decrease. When the desired position is reached, the resulting effort is zero. This implementation thus adds another control loop to the system. Since the independent jogging mode uses trajectory generator, there is a possibility of overshoot. Equation 6-1 is set up to quickly fix the error and return the joint back to the desired position.

The Manipulator End-Effector Pose Driver component receives the desired position and orientation of the end-effector as its inputs. It then uses the reverse position analysis (Section 2.3.2) to compute the corresponding joint values. Just as with the MJPD, this component constantly queries for current joint positions. Once this information is available, the identical approach (Equation 6-1) is used to complete the task.

Definition of the joint effort for the two components commanding velocity is much simpler and is given by Equation 6-2.

$$effort[n] = \frac{velocity_{des}[n]}{velocity_{\max}[n]} \tag{6-2}$$

The Manipulator Joint Velocities Driver component receives the desired joint velocities, while the MEEVD uses reverse velocity analysis from Section 2.3.3 to determine these values given the desired velocity state of the end-effector. Once computed, these values are passed on to the Primitive Manipulator. The effort values should never exceed -100% or 100% (e.g, commanding a -2500 encoder count per second velocity would be converted to -50%, given the maximum velocity of 5000 encoder counts per second). It is entirely up to the Galil controller to close the loop and ensure joint motion at the desired speed.

This chapter explored all the core functionality of the JAUS manipulator implementation. The following chapter further takes the above capabilities and allows the system to handle multiple sets of joint positions and end-effector positions and orientations.

CHAPTER 7
MID-LEVEL POSITION AND VELOCITY DRIVER COMPONENTS

This chapter defines the last 2 components with respect to the scope of our study. Up to this point, this modular architecture was only concerned with single-goal completion. The components presented in this chapter will extend this interface to allow the manipulator to receive multiple joint position configurations or a tool-point path profile. It should be noted that in most implementations, these components and the Primitive Manipulator will be embedded in the same node which will facilitate the control process. Finally, the last section will briefly discuss how these 2 components are implemented on the Puma platform.

7.1 Manipulator Joint Move Driver Component

7.1.1 Component Function

The function of the Manipulator Joint Move Driver (MJMD) is to perform closed-loop joint level control of the manipulator where motion parameters for each joint are specified. The specified motion parameters are the desired values, maximum velocity, maximum acceleration, and maximum deceleration.

7.1.2 Associated Messages

The MJMD (JAUS ID# 58) accepts the core input and output messages (Chapter 1, Table 1-1). Some of the user defined messages that are received or sent by this component were defined in previous chapters, while Set Joint Motion message is defined in Section 7.1.4. The following messages are associated with this component:

- Set Joint Motion

- Report Manipulator Specifications
- Report Joint Effort
- Report Joint Positions
- Report Joint Velocities
- Set Joint Effort

7.1.3 Component Description

The inputs are the desired joint values at specified time values together with data to define a trapezoidal velocity profile (i.e. the maximum joint velocity, maximum joint acceleration, and maximum joint deceleration). No explicit path is defined, only the values of the joint angles at distinct times. The time values are measured in units of seconds and are relative to the time that the movement to the first joint angle set is started. Additional inputs are the current joint values, joint velocities, and the data from the Report Manipulator Specifications message. The output is the joint effort level that is sent to the Primitive Manipulator component. The MJMD is shown in Figure 7-1.

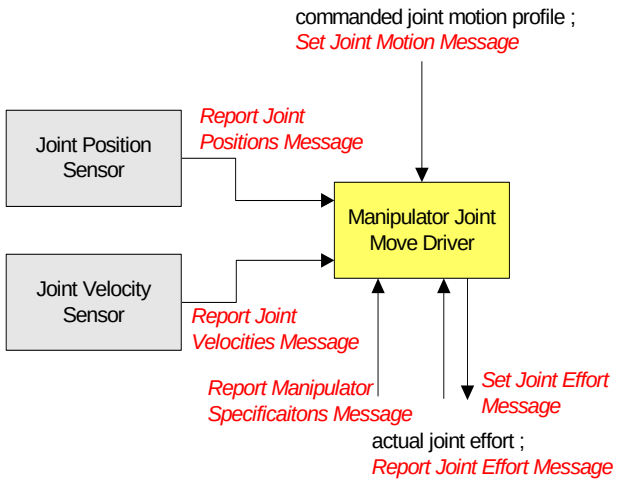

Figure 7-1. Manipulator Joint Move Driver component

7.1.4 Code 0607: Set Joint Motion

Note that this message specifies the number of manipulator joints and the number of poses (different trapezoidal profile sets).

Table 7-1. Set Joint Motion message parameters

Field #	Name	Type	Units	Interpretation
1	Num Joints, n	Byte	N/A	1 ... 255 0 is Reserved
2	number of poses, p	Byte	N/A	1…255 0 is Reserved
3	pose 1 time	int	Sec	Scaled integer: Lower limit = 0 sec Upper limit = 6000 sec
4	Joint 1 – position at pose 1	int	rad or m	Scaled integer: If revolute joint, Lower limit = -8π rad Upper limit = +8π rad If prismatic joint, Lower limit = -10 m Upper limit = +10 m
5	Joint 1 – max velocity	int	rad/s or m/s	If revolute joint, Lower limit = -10π rad/s Upper limit = +10π rad/s If prismatic joint, Lower limit = -5 m/s Upper limit = +5 m/s
6	Joint 1 – max acceleration	int	rad/s^2 or m/s^2	If revolute joint, Lower limit = -10π rad/s^2 Upper limit = +10π rad/s^2 If prismatic joint, Lower limit = -20 m/s^2 Upper limit = +20 m/s^2
7	Joint 1 – max deceleration	int	rad/s^2 or m/s^2	If revolute joint, Lower limit = -10π rad/s^2 Upper limit = +10π rad/s^2 If prismatic joint, Lower limit = -20 m/s^2 Upper limit = +20 m/s^2
8	Joint 2 – position at pose 1	int	rad or m	see field 4
9	Joint 2 – max velocity	int	rad/s or m/s	see field 5

Table 7-1. Continued

Field #	Name	Type	Units	Interpretation
10	Joint 2 – max acceleration	int	rad/s^2 or m/s^2	see field 6
11	Joint 2 – max deceleration	int	rad/s^2 or m/s^2	see field 7
…				
4n	Joint n – position at pose 1	int	rad or m	see field 4
4n+1	Joint n – max velocity	int	rad/s or m/s	see field 5
4n+2	Joint n – max acceleration	int	rad/s^2 or m/s^2	see field 6
4n+3	Joint n – max deceleration	int	rad/s^2 or m/s^2	see field 7
…				
(p-1) 4n+4	pose p time	int	s	see field 3
(p-1) 4n+5	Joint 1 – position at pose p	int	rad or m	see field 4
(p-1) 4n+6	Joint 1 – max velocity	int	rad/s or m/s	see field 5
(p-1) 4n+7	Joint 1 – max acceleration	int	rad/s^2 or m/s^2	see field 6
(p-1) 4n+8	Joint 1 – max deceleration	int	rad/s^2 or m/s^2	see field 7
…				
(p-1)8n	Joint n – position at pose p	int	rad or m	see field 4
(p-1)8n+1	Joint n – max velocity	int	rad/s or m/s	see field 5
(p-1)8n+2	Joint n – max acceleration	int	rad/s^2 or m/s^2	see field 6
(p-1)8n+3	Joint n – max deceleration	int	rad/s^2 or m/s^2	see field 7

7.2 Manipulator End-Effector Discrete Pose Driver Component

7.2.1 Component Function

The function of the Manipulator End-Effector Discrete Pose Driver (MEEDPD) is to perform closed-loop control of the end-effector pose through a series of specified positions and orientations.

7.2.2 Associated Messages

The MEEDPD (JAUS ID# 59) accepts the core input and output messages (Chapter 1, Table 1-1). Some of the user defined messages that are received or sent by this component were defined in previous chapters, while Set End-Effector Path Motion message is defined in Section 6.4.4. The following is the list of all the messages associated with this component:

- Set End-Effector Path Motion
- Report Manipulator Specifications
- Report Joint Effort
- Report Joint Positions
- Report Joint Velocities
- Set Joint Effort

7.2.3 Component Description

This component performs closed-loop control of the end-effector pose as measured with respect to the vehicle coordinate system. The inputs are a path motion description (discrete end-effector position and orientation at time t measured in the vehicle coordinate system which is defined by a point and a quaternion at time t), the current

joint values, the current joint velocities, and the data from the Report Manipulator Specifications message. The output is the joint effort level that is sent to the Primitive Manipulator component. The functionality of the MEEDPD is shown in Figure 7-2.

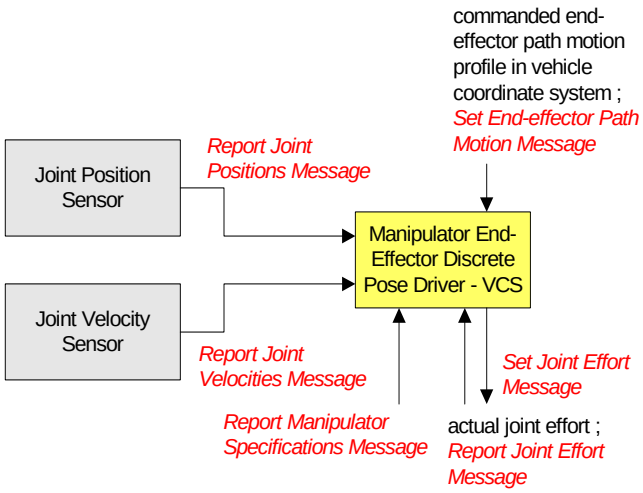

Figure 7-2. Manipulator End-Effector Discrete Pose Driver component

7.2.4 Code 0608h: Set End-Effector Path Motion

A series of end-effector poses are defined in terms of the vehicle coordinate system at various times. The time is a relative time and is defined in seconds where time equals 0 is the moment that the motion towards the first pose begins.

7.3 Applications of the Mid-Level Driver Components to the Puma System

The Section 3.3.1 described the independent axis positioning mode inherent to the Galil controller. The parameters defining joint motion in this mode are identical to those defined by Set Joint Motion message. Furthermore, the independent jogging mode used in previous component definitions provides closed-loop position control generating a trapezoidal trajectory based on these parameters. Thus, the MJMD can be defined as a superset of the MJPD component, in that it redefines the maximum speed, acceleration and deceleration with each new pose instead of using the default values.

Table 7-2. Set End-Effector Path Motion message parameters

Field #	Name	Type	Units	Interpretation
1	number of poses, n	Byte	N/A	1 ... 255 0 is Reserved
2	time 1	int	sec	time for pose 1 Scaled integer: Lower limit = 0 sec Upper limit = 6000 sec
3	X component of tool point for pose 1	int	m	Scaled integer: Lower limit = -30 m Upper limit = +30 m
4	Y component of tool point for pose 1	int	m	see field 3
5	Z component of tool point for pose 1	int	m	see field 3
6	d component of unit quaternion q for pose 1	int	N/A	Scaled integer: Lower limit = -1 Upper limit = +1
7	a component of unit quaternion q for pose 1	int	N/A	see field 6
8	b component of unit quaternion q for pose 1	int	N/A	see field 6
9	c component of unit quaternion q for pose 1	int	N/A	see field 6
…				
8n-6	time n	int	sec	see field 2
8n-5	X component of tool point for pose n	int	m	see field 3
8n-4	Y component of tool point for pose n	int	m	see field 3
8n-3	Z component of tool point for pose n	int	m	see field 3
8n-2	d component of unit quaternion q for pose n	int	N/A	see field 6
8n-1	a component of unit quaternion q for pose n	int	N/A	see field 6
8n	b component of unit quaternion q for pose n	int	N/A	see field 6
8n+1	c component of unit quaternion q for pose n	int	N/A	see field 6

An identical approach is taken to determine the corresponding joint efforts (Equation 6-1). Note however, that the proportional constant k_p should change with the

magnitude of these parameters. Since this implementation is used to test rather then optimize manipulator performance, previously determined values of k_p are adequate for operating speeds below 10000 encoder counts per second. Thus after each pose is completed a new set of parameters is applied to the system.

Unlike the MJMD, the functionality of the MEEDPD component is identical to that of the MEEPD. It just allows for a single message to define multiple tool-point positions and orientations to be executed autonomously at a particular time. Note that this implementation uses generic timing functions provided with the Linux kernel. More precise operation would require the use of a real-time operating system. All aspects of software design are addressed in the following chapter.

CHAPTER 8
OVERVIEW OF SOFTWARE DESIGN

The purpose of our study was to design and implement the JAUS manipulator components. The design aspects were covered in Chapters 4, 5, 6, and 7, while the software implementation is addressed here. Keep in mind that the chapter covers design and logistics aspects rather than computer science technicalities. The source code is rather complicated but well documented and could be easily read. For completion purposes, this chapter gives a brief overview of the node manager and its role in inter-component communications.

8.1 Overview

The scope of computer programming associated with each of the components and their corresponding messages reflects the importance of software design in this and any other large scale project. In order to allow for easy and clear understanding behind the process control and data flow in each of the functional elements, this chapter begins its analysis at the lowest level and moves up hierarchically. More specifically, it first addresses the logistics behind the interface to the Galil controller, then goes into the component and message development, and finally describes the node level functionality. Due to a high level of redundancy, only one component out of the first three groups will be analyzed in detail. Furthermore, this development takes advantage of multithreading, a powerful tool described in many popular computer science books. From a software development standpoint, each component is treated as a separate thread, and could be

compared to a program running a while-loop. An additional thread is used in the interface to the controller.

8.2 The Interface to the Galil Controller

The Section 3.4 gave an overview of the C function prototypes that are the core of this development. Only a single program/thread is responsible for communications with the Galil controller eliminating the need for multithread synchronization. The file galilInterface.c is responsible for managing startup and shutdown routines of this particular thread. The interface has a well defined structure that is shown in Figure 8-1:

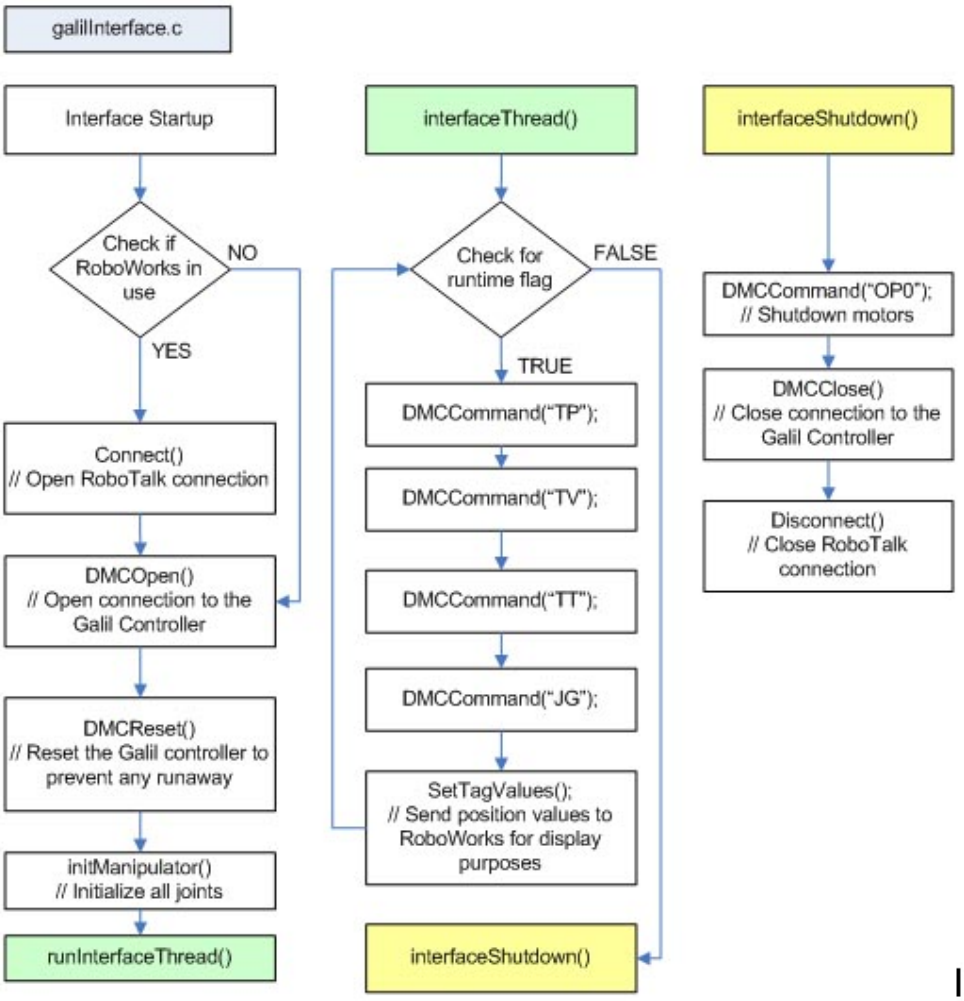

Figure 8-1. The galilInterface.c logic flow diagram

On startup, the code checks whether the RoboWorks 3-D software tool is used. This package, developed at the University of Texas, was slightly modified to work with the Linux operating system for both simulation and real-time 3D display purposes. Its application will become more apparent in the next chapter. RoboTalk is the program that handles Ethernet communications between the computer hosting RoboWorks and the computer providing position data. The file RoboTalk.h contains function prototypes for the Connect(), SetTagValues(..), and Disconnect() functions shown in Figure 8-1. This is followed by establishing a connection to the Galil controller, resetting it for safety purposes, and calibrating all the joints using stored potentiometer values.

After the state thread is started, position, velocity and force/torque values are queried using “TP”, “TV”, and “TT” commands, respectively. Regardless of the fact that the Puma 762 does not have any torque sensors, this component was implemented for testing purposes. Commanding an independent jogging motion is accomplished using the “JG” command (e.g. if the Primitive Manipulator commands a 50% effort, this is converted to -2500 encoder counts per second velocity using Equation 6-2, assuming the maximum velocity is 5000 encoder counts per second). In addition to the core functionality, galilInterface.c is used to set up maximum speed, acceleration, and deceleration values using “SP”, “AC”, and “DC” commands generating a trapezoidal profile. The function is set as a sleeper outside the thread and will change the values only if called. Only the Manipulator Joint Motion component is responsible for changes to the default parameters. The file also contains a homing function that could take the manipulator to its home position on startup. This capability should be used after recalibration of joint values. The source code for galilInterface.c, RoboTalk.c, and

corresponding header files is listed in Appendix B, Section B.1. The Primitive Manipulator component and all of the sensor components acquire data from the interface file using external access functions eliminating the need for the use of global variables.

8.3 Component Level Software Development

Components are based on standardized interfaces (Section 1.3.4). From a software development standpoint, this means that the code structure of the files defining these unique JAUS entities is very similar. Figure 8-2 shows the functional elements of a generic component file.

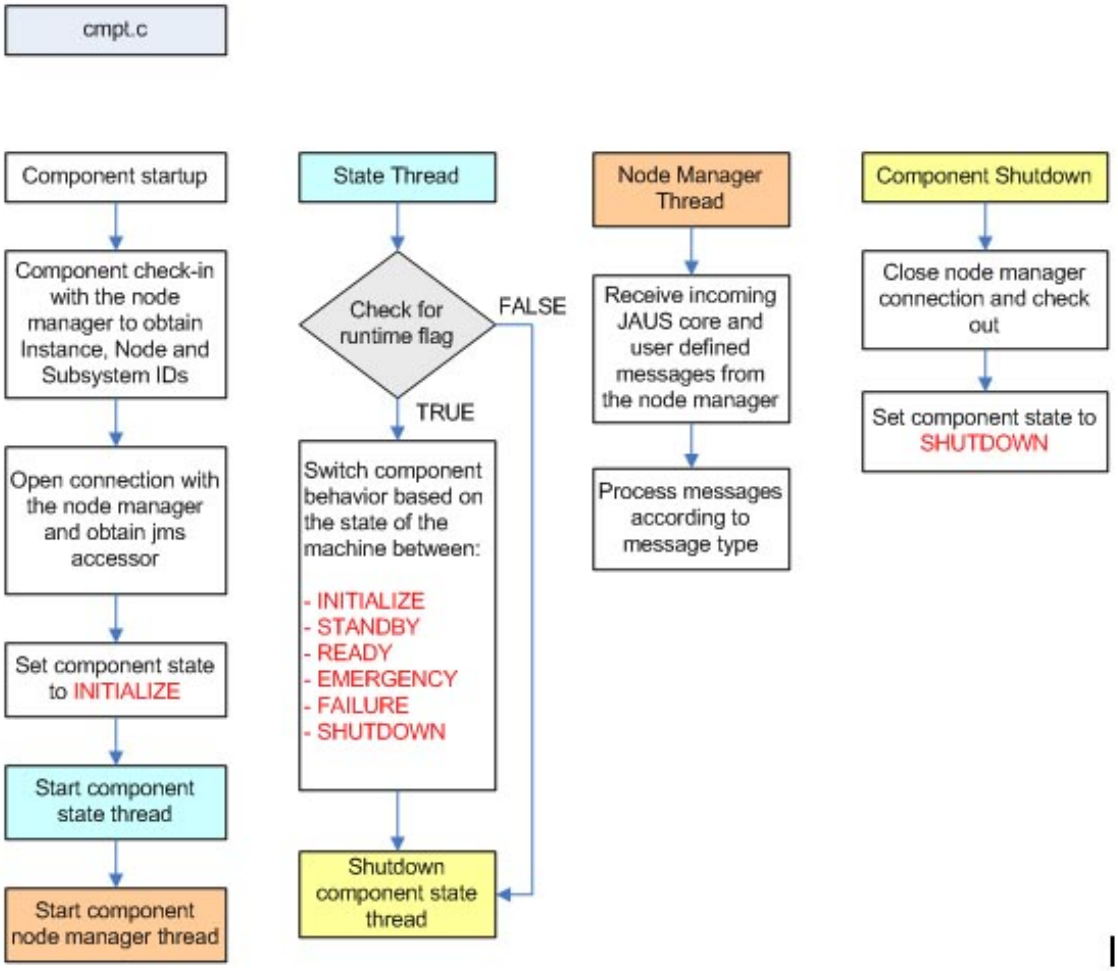

Figure 8-2. Generic component logic flow diagram

Each component follows a standard process flow defining thread startup and shutdown routines. As it is shown in Figure 8-2, each component consists of two threads. The first is responsible for performing component function while the other handles the incoming core and user-defined messages. Therefore it is in the state thread that the particular component's behavior is defined. This implementation defines Emergency, Failure, and Shutdown states identically across the entire node. In order to better understand the inner workings of these JAUS elements, the state thread of one component from each of the Chapters 4, 5, and 6 is analyzed.

8.3.1 Primitive Manipulator State Thread

The best way to represent the processes within a particular file is using a flow chart. Therefore, Figure 8-3 shows the PM state thread.

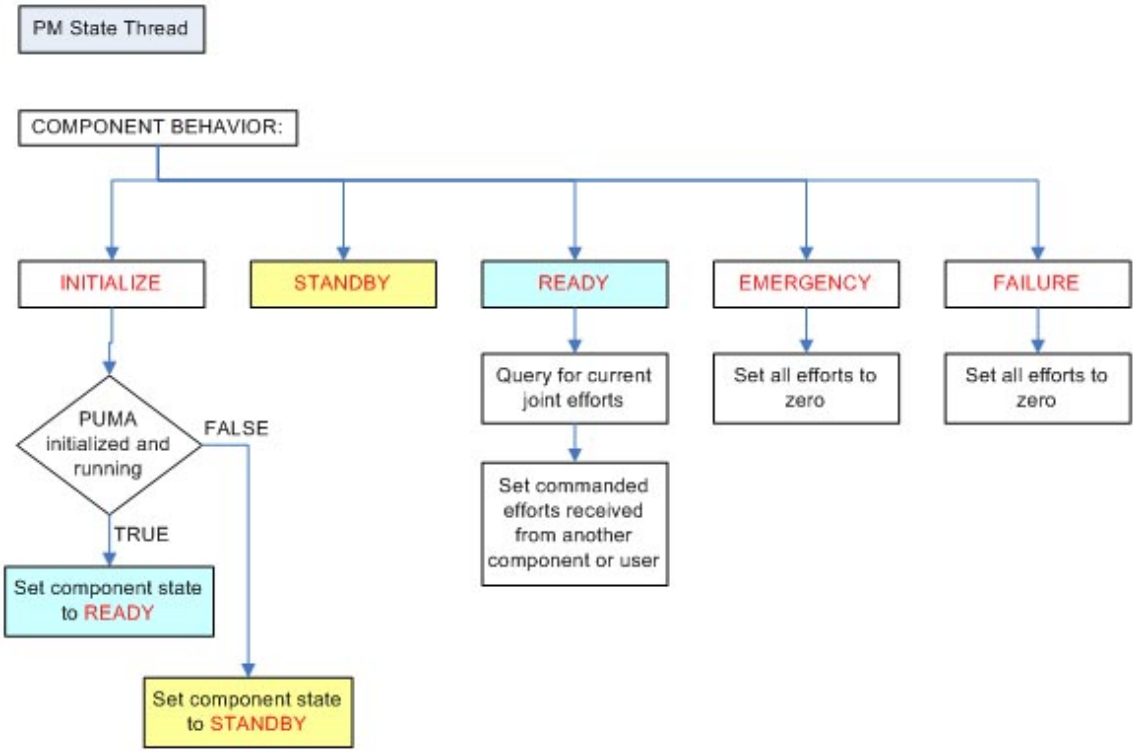

Figure 8-3. The Primitive Manipulator state thread logic flow diagram

Each component is set to the Initialize state by default during the startup routine. While at this state, the PM accesses the information provided by the interface to the Galil controller requesting the status on initialization procedures, ensuring all integrity checks are complete and the system is up and running. Once the component enters the Ready state, it talks to the interface file again, but this time accessing the function directly responsible for commanding motion, given the joint effort values. The source code for the file pm.c and its corresponding header is documented in Appendix B.2.

8.3.2 Manipulator Sensor Components

The state thread of Manipulator Joint Position Sensor is described in this section as it provides more complex functionality than the other sensor components. Definitions of Initialize, Standby and Ready states are shown in Figure 8-4.

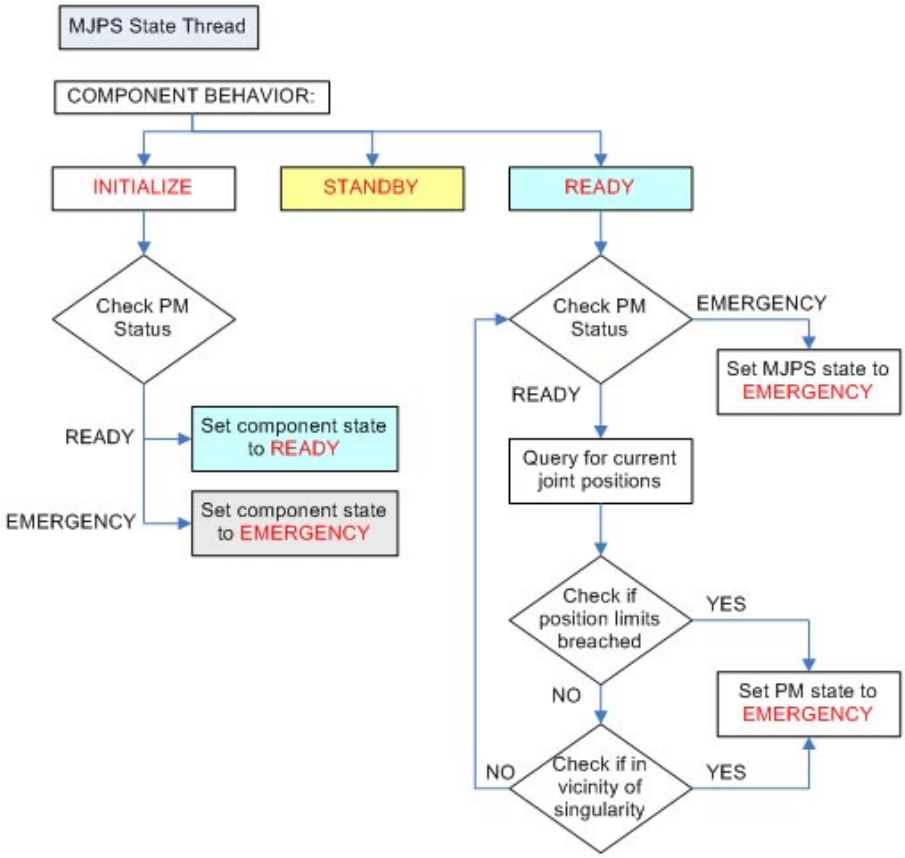

Figure 8-4. Manipulator Joint Position Sensor logic flow diagram

The definition of the initialize state in the above diagram is applied to the remainder of the components on this node. If the query status message to the Primitive Manipulator returns Ready, it is assumed that the initialization procedure was successful. Once in the Ready state, the MJPS accesses the Galil interface file and queries it for position. Next, it checks those values ensuring that none of the joints are crossing position limits, or getting close to a singularity configuration. In either case, the MJPS changes the status of the Primitive Manipulator to the Emergency state. This instantly causes all the other components to go into the Emergency state as well. The system would now have to be reinitialized. The source code for the mjps.c file and the corresponding header is listed in the Appendix B.3.

8.3.3 Low-Level Position and Velocity Drivers

It should be noted, that all states, except Ready, are defined in the same manner as covered in previous sections across all of the remaining components. Figure 8-5 shows the MEEPD state thread (Ready state only).

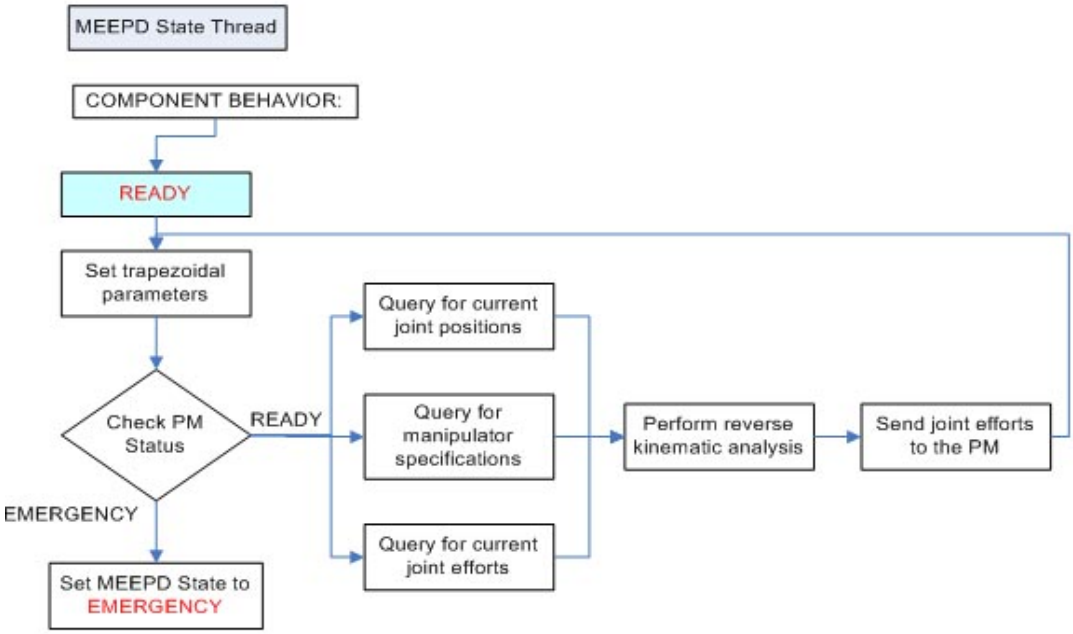

Figure 8-5. Manipulator End-Effector Pose Driver logic flow diagram

The Figure 8-5 simply states that this component performs its unique function as long as there is no change in the status of the Primitive Manipulator. At this point all of the software functionality is defined, and since the timing functionality of the mid-level components is a trivial addition, it is not discussed in detail. However, for completion purposes, Appendix B.4 and Appendix B.5 list the meepd.c and the meedpd.c files and their corresponding headers, respectively.

8.4 Message Level Software Development

Each manipulator component sends or receives user defined messages. Prior to messages being exchanged, they are compressed (packed) and eventually uncompressed (unpacked) once they reach the destination (Section 1.3.5). Packing and unpacking algorithms are well defined and easy to implement. They are based on converting a desired parameter into a byte stream, thus using a correct byte counter is critical for preserving correct values. These functions can further clip passing parameters if they lay outside the specified limits. Information being passed is defined as a structure type so that it is easily accessed by the components requesting it. For example, joint velocities are sent by the MJVD component as a part of a jointVelocity_t structure containing six velocity values and a parameter defining the total number of joints. Message files and their corresponding headers are documented in the Appendix C.

8.5 Node Level Software Development

The Manipulator Control node is responsible for startup and shutdown of all the manipulator components. In the case of the Puma system, it also ensures that both the Val and Galil controllers, as well as the arm itself are powered up before starting the interface thread. The order in which the components are started up is irrelevant, but should be

consistent with the order in which they are shutdown. Two files, mc.c and mc.h, are listed in the Appendix B.6.

8.6 Node Manager and Communicator

Two JAUS components responsible for all inter-component communications are the Communicator and the Node Manger. The Communicator allows for a single point of message entry into a subsystem while maintaining the data link between the subsystems. The Node Manger component is responsible for routing JAUS messages from one node to another. When a component sends a message [7], it first goes to the Node Manager where the information regarding the destination component is interpreted and passed onto the communicator, which handles the transmission of the message to the proper nod.

Finally, at the highest level, the main file is responsible for starting the node itself. This file further uses a “curses” text-based window environment for display purposes. How the information is being displayed for each of the components is discussed in the next chapter. The main.c file is listed in the Appendix B.7.

CHAPTER 9
TESTING AND RESULTS

This chapter covers the methods used to test 2 aspects of this JAUS implementation. The first ensures that the functionality of developed messages and components meets the JAUS specification requirements. The latter looks at how well this architecture handles manipulator control from the view point of accuracy and task completion. In order to adequately test this implementation, an additional component mimicking a Subsystem Commander (SSC) was created. The manipulator sensor components are self tested as the information they provide becomes critical to components responsible for closed-loop position and velocity control. Essentially 7 different cases or scenarios pertaining to the functionality of the driver components are evaluated by the SSC. User input is required when choosing the scenario, however the current setup mandates that any parameter change has to be done in the source code.

9.1 Subsystem Commander Component Overview

The Subsystem Commander component is defined in the JAUS Reference Architecture document. Its function is to coordinate all activity within a given subsystem. The SSC (JAUS ID# 32) has the responsibility of performing mission planning, issuing commands, and querying status for the subsystem operation. The document states that functions of this component may be performed by humans, or by the computer or both [7]. The capabilities of the RoboWorks software package are integrated with this component to display real-time position data of the manipulator. This could be useful when the commander is located off site. Figure 9-1 shows this operator control setup.

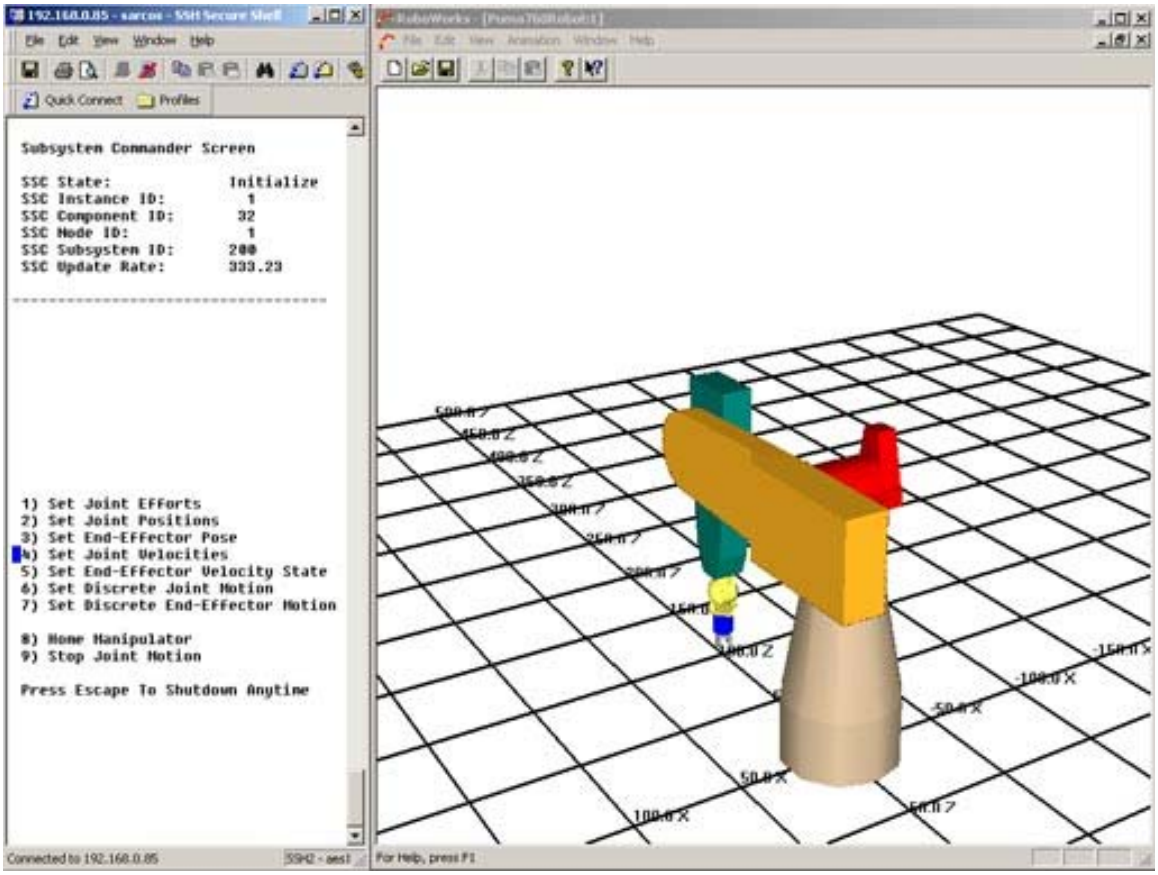

Figure 9-1. Subsystem Commander operator graphical user interface

9.2 Case 1: Set Joint Effort

The first scenario is used to test the Primitive Manipulator component without any sensor feedback. In order to ensure that no other components are commanding joint efforts, their status is changed to the Standby state. Note that the case numbers correspond to the items listed in Figure 9-1. Set Joint Effort Message commands the following set of values:

$$effort = \begin{bmatrix} 15 & -15 & 15 & -12 & 12 & -12 \end{bmatrix} \tag{9-1}$$

Given the default maximum velocity value of 5000 encoder counts per second (enc/s) for the first three joints and 1000 enc/s for the last three joints, these values are converted back to [750, -750, 750, -120, 120, -120] enc/s as they are passed on to the

Galil controller using Equation 6-2. Figure 9-2 shows a snapshot of the Primitive Manipulator screen during the execution of this command.

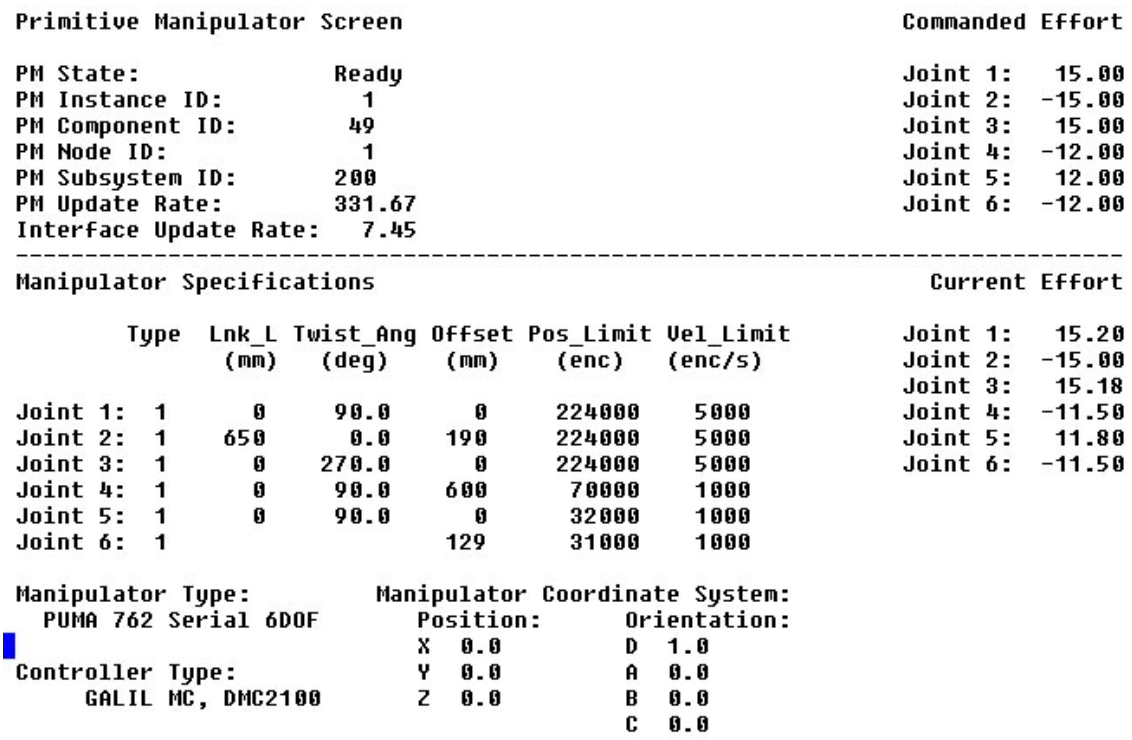

Figure 9-2. Primitive Manipulator Screen as it responds to the Set Joint Effort message

Note that the current effort values are slightly different from the commanded ones. This error ranging between 1% and 4% is inherent to the underlying theory behind the independent jogging mode covered in Chapter 3. Figure 9-2 also shows that the PM screen is used to display the contents of the Manipulator Specifications message. Mechanism and controller information is also available. It is useful to realize that in this and the upcoming sections, any information regarding position and velocity information is displayed in units of encoder counts (enc) and encoder counts per second (enc/s), respectively. This was done for practical purposes to allow for quick interpretation of the corresponding positions with the actual ones. Table 1-1 lists the conversion factors used to go from specified JAUS standard units to enc and enc/s and vice versa:

Table 9-1. The Puma 762 platform specific conversion factors

Joint #	Radians to Encoder Counts	Encoder Counts to Degrees
1	45836.6	0.00125
2	65951.9	0.00086875
3	51357.5	0.001115625
4	9152.6	0.00626
5	9152.6	0.00626
6	4285.3	0.01337

9.3 Case 2: Set Joint Position

This case tests the capabilities of the Manipulator Joint Position Driver. The SSC simply commands six joint values corresponding to the desired joint positions. That information is then used to determine the corresponding joint efforts (Equation 6-1) that are passed to the PM. Unlike the PM, MJVD, and MEEVD components, this and the other position drivers use the independent jogging mode to command position. This requires an introduction of an additional loop to the existing control system as shown in Figure 9-3.

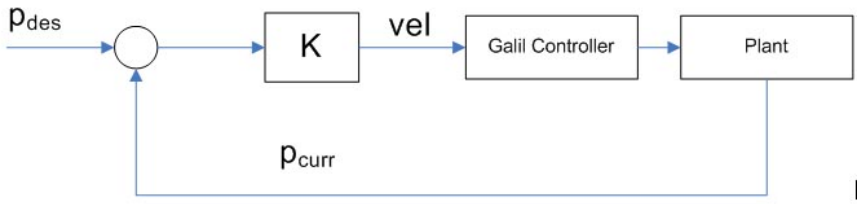

Figure 9-3. External closed-loop control diagram

The value of K is determined experimentally and its values are tuned to provide stable, no-steady-state-error operation of each of the joints within a particular range of motion parameters. The Puma 762 system is capable of high-speed operation with great accuracy; however such implementation would require a different set of K values and potentially a more complicated control loop. The values of K and motion parameters used in this project are summarized in Table 9-2.

Table 9-2. Values of the K constant and corresponding range of motion parameters

Joint #	K	Max Speed (end/s)	Max Acceleration (enc/s^2)	Max Deceleration (enc/s^2)
1	34	1000-10000	28000-256000	28000-256000
2	34	1000-10000	28000-256000	28000-256000
3	34	1000-10000	28000-256000	28000-256000
4	34	250-1000	28000-256000	28000-256000
5	15	250-1000	28000-256000	28000-256000
6	14	250-1000	28000-256000	28000-256000

To further illustrate the performance of the manipulator system within these ranges, the MJPS is commanded the same position, [35000, -35000, 50000, 5500, 6500, -4000], with three different sets of motion parameters. Keep in mind that when this or any other driver component enters the Ready state, all other components except the Primitive Manipulator and the sensors are set to Standby.

9.3.1 The "Average" Set

Table 9-3 shows the set of "average" motion parameters used to test the performance of the MJPD.

Table 9-3. Values of the K constant and "average" set of motion parameters

Joint #	K	Max Speed (end/s)	Max Acceleration (enc/s^2)	Max Deceleration (enc/s^2)
1	34	5000	128000	128000
2	34	5000	128000	128000
3	34	5000	128000	128000
4	34	1000	128000	128000
5	15	1000	128000	128000
6	14	1000	128000	128000

The step responses of each of the joints as they are commanded motion are shown in Figure 9-4.

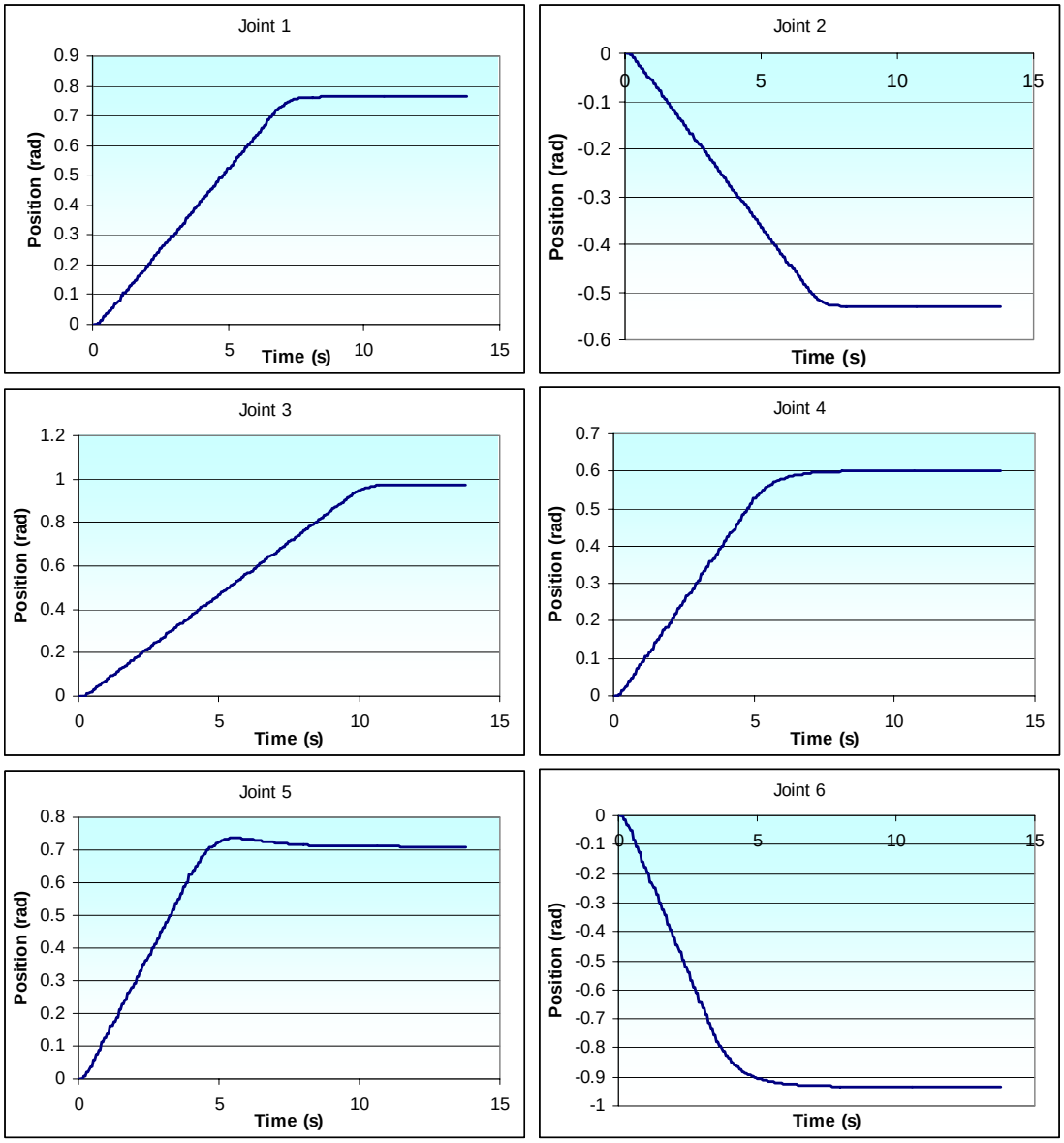

Figure 9-4. Joint step responses using "average" motion parameters

```
Manipulator Joint Positions Driver Screen                    Commanded Position

MJPD State:            Ready                                  Joint 1:    35000
MJPD Instance ID:        1                                    Joint 2:   -35000
MJPD Component ID:      54                                    Joint 3:    50000
MJPD Node ID:            1                                    Joint 4:     5500
MJPD Subsystem ID:     200                                    Joint 5:     6500
MJPD Update Rate:      330.16                                 Joint 6:    -4000

-------------------------------------------------------------------------------
Queried Information:     effort   position                     Resulting Effort

           Joint 1:       0.00      35000                     Joint 1:     0.00
           Joint 2:       0.00     -35000                     Joint 2:     0.00
           Joint 3:       0.00      50000                     Joint 3:     0.00
           Joint 4:       0.00       5499                     Joint 4:     0.10
           Joint 5:       0.00       6501                     Joint 5:    -0.14
           Joint 6:       0.00      -3999                     Joint 6:    -0.09
```

Figure 9-5. The MJPD screen as motion is completed under the "average" set

Note that only Joint 5 has a slight overshoot, and that all the joints arrive within 1 encoder count of the desired position as shown in Figure 9-5. This error falls within performance specifications of the Galil controller.

9.3.2 The "Low" Set

Table 9-4 shows the set of "low" motion parameters used to test the performance of the Manipulator Joint Positions Driver. The step responses are very similar to those in Figure 9-4 with Joint 5 having less overshoot. However low acceleration and deceleration values yield low response time, thus the resulting position falls within a larger margin of error. As Figure 9-6 shows, that magnitude of this error is +/-6 encoder counts. The step responses of each of the joints as they are commanded motion are shown in Figure 9-7.

Table 9-4. Values of the K constant and "low" set of motion parameters

Joint #	K	Max Speed (end/s)	Max Acceleration (enc/s^2)	Max Deceleration (enc/s^2)
1	34	1000	28000	28000
2	34	1000	28000	28000
3	34	1000	28000	28000
4	34	250	28000	28000
5	15	250	28000	28000
6	14	250	28000	28000

```
Manipulator Joint Positions Driver Screen                    Commanded Position

MJPD State:            Ready                                  Joint 1:    35000
MJPD Instance ID:        1                                    Joint 2:   -35000
MJPD Component ID:      54                                    Joint 3:    50000
MJPD Node ID:            1                                    Joint 4:     5500
MJPD Subsystem ID:     200                                    Joint 5:     6500
MJPD Update Rate:      19.24                                  Joint 6:    -4000

-------------------------------------------------------------------------------
Queried Information:     effort   position                     Resulting Effort

           Joint 1:       0.00      34996                     Joint 1:     0.12
           Joint 2:       0.00     -34996                     Joint 2:    -0.12
           Joint 3:       0.80      49954                     Joint 3:     1.40
           Joint 4:       0.00       5494                     Joint 4:     0.58
           Joint 5:       0.00       6495                     Joint 5:     0.47
           Joint 6:       0.00      -3994                     Joint 6:    -0.54
```

Figure 9-6. The MJPD screen as motion is completed under the "low" set

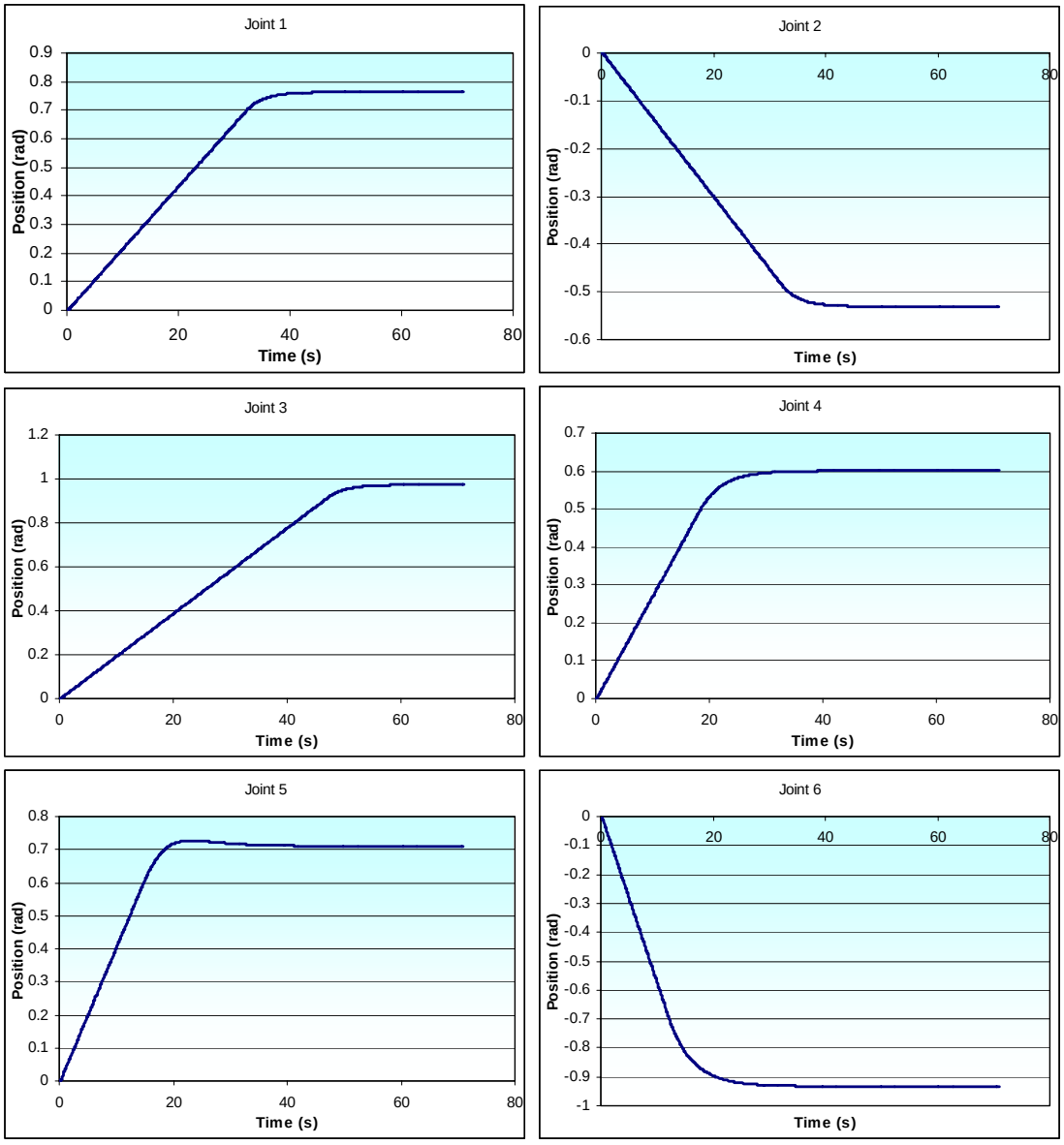

Figure 9-7. Joint step responses using “low” motion parameters

9.3.3 The “High” Set

Table 9-5 shows the set of “high” motion parameters used to test the performance of the MJPD. The step responses of each of the joints as they are commanded motion are shown in Figure 9-8. High values of maximum speed, acceleration and deceleration govern the quick response to the commanded position.

Table 9-5. Values of the K constant and “high” set of motion parameters

Joint #	K	Max Speed (end/s)	Max Acceleration (enc/s^2)	Max Deceleration (enc/s^2)
1	34	10000	256000	256000
2	34	10000	256000	256000
3	34	10000	256000	256000
4	34	2500	256000	256000
5	15	2500	256000	256000
6	14	2500	256000	256000

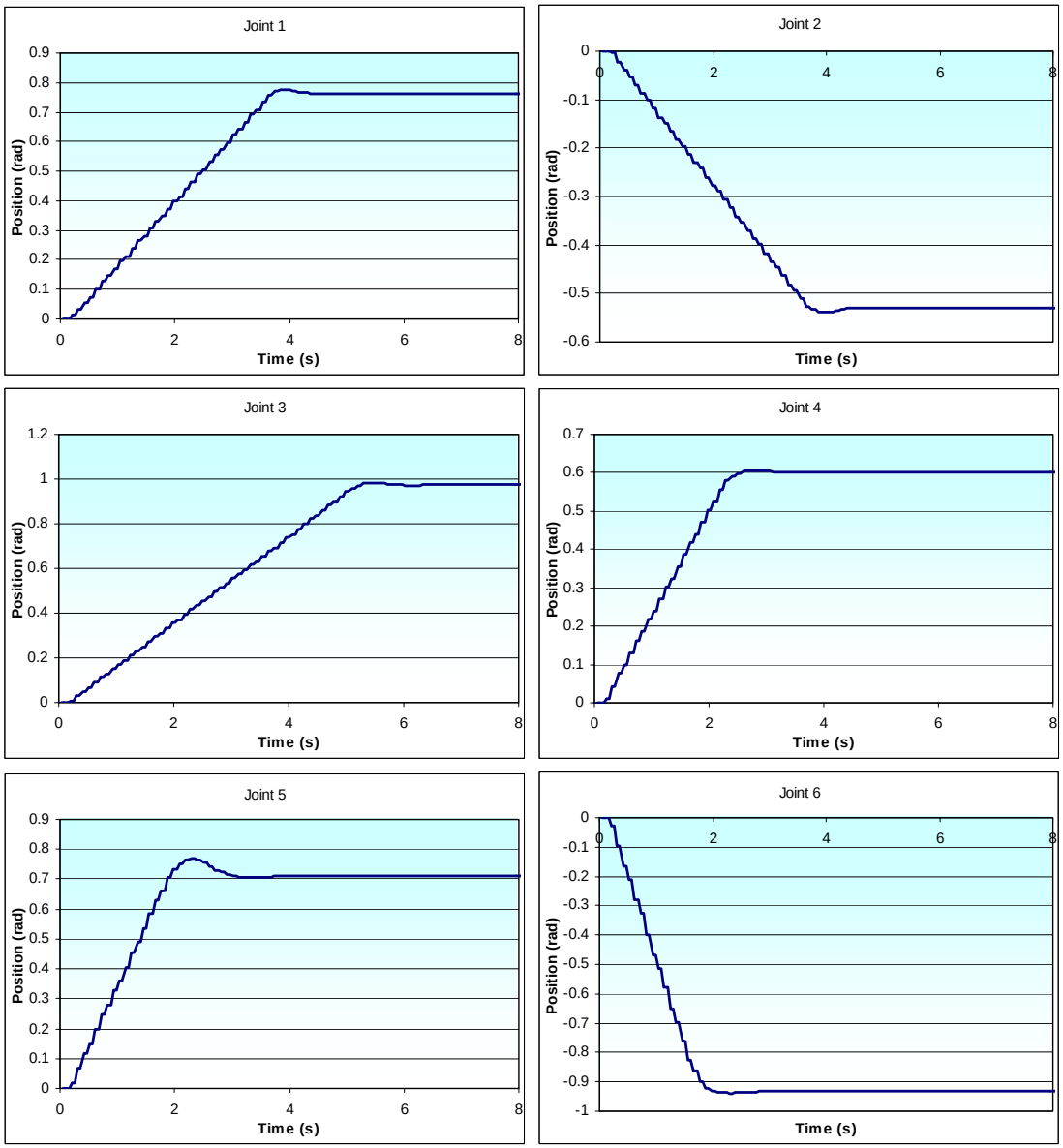

Figure 9-8. Joint step responses using “high” motion parameters

Unlike the other two cases, this “high” case starts to show some signs of instability in the initial part of the motion, but quickly recovers with resulting position error within the Galil controller specifications as shown in Figure 9-9.

```
Manipulator Joint Positions Driver Screen                    Commanded Position

MJPD State:            Ready                                  Joint 1:    35000
MJPD Instance ID:        1                                    Joint 2:   -35000
MJPD Component ID:      54                                    Joint 3:    50000
MJPD Node ID:            1                                    Joint 4:     5500
MJPD Subsystem ID:     200                                    Joint 5:     6500
MJPD Update Rate:    19.23                                    Joint 6:    -4000

-------------------------------------------------------------------------------
Queried Information:     effort   position                     Resulting Effort

            Joint 1:       0.00      34998                     Joint 1:     0.06
            Joint 2:       0.00     -35000                     Joint 2:     0.00
            Joint 3:       0.00      50000                     Joint 3:     0.00
            Joint 4:       0.00       5500                     Joint 4:     0.00
            Joint 5:       0.00       6500                     Joint 5:     0.00
            Joint 6:       0.00      -4000                     Joint 6:     0.00
```

Figure 9-9. The MJPD screen as motion is completed under the “high” set

Figure 9-10 shows the final configuration of the PUMA 762 manipulator as seen by the Subsystem Commander component.

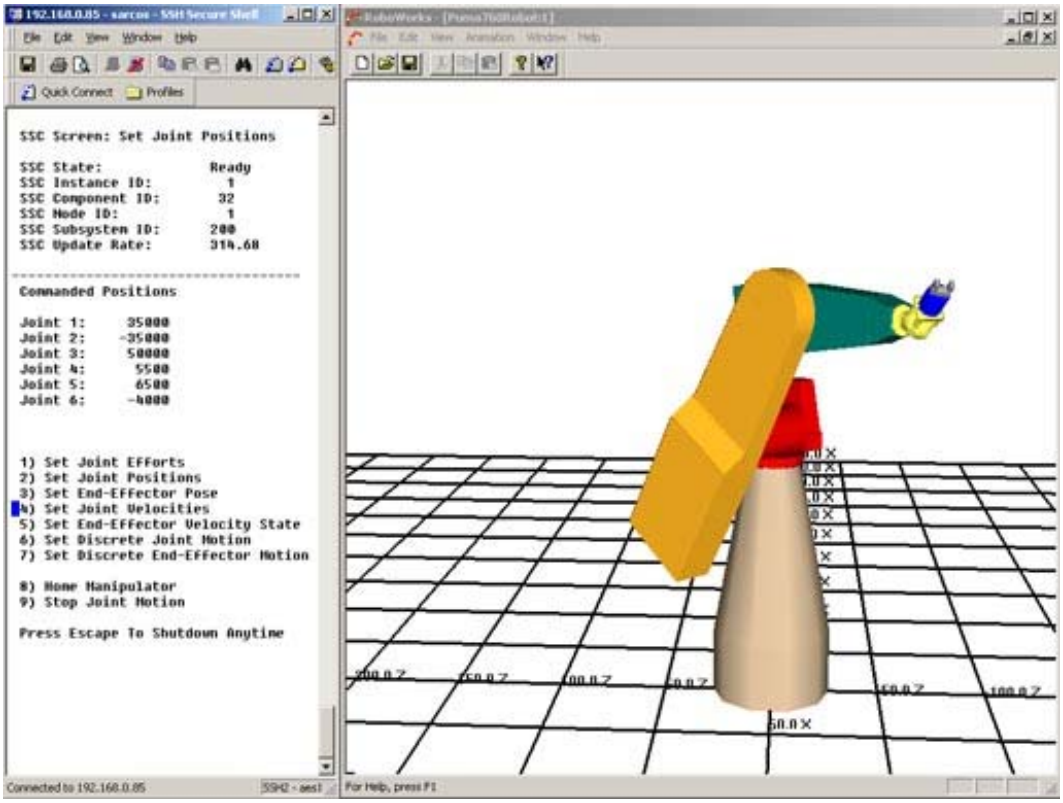

Figure 9-10. Subsystem Commander screen showing the final configuration of Puma 762

9.4 Case 3: Set End-Effector Pose

This section shows the performance of the Manipulator End-Effector Pose Driver Component. As described in Chapter 8, this component is responsible for computation of independent joint efforts based on the commanded position and orientation of the end-effector.

In order to adequately perform the forward or reverse position analysis, the joint angles that correspond to the manufacturer's specifications need to be converted to the kinematic values or vice versa using the following relationship:

$$\theta_{i,kinematic} = K_1 \theta_{i,manufacturer} + K_2 \tag{9-2}$$

where values of K_1 correspond to the conversion factors shown in column 3 of Table 9-1, and values of K_2 are [0, 90, 90, 180, 180, 0] for Joints 1, 2, 3, 4, 5, and 6 respectively. The orientation of the end-effector is expressed using a unit quaternion. Simple approach outlined in Section 2.3.5 is used to convert the four values defining a quaternion into the ${}^F a_{67}$ and ${}^F S_6$ vectors. Once the analysis is complete, the function returns up to eight possible solution sets. An optimization technique is used to pick one solution by defining a total cost function given in Equation 9-3:

$$\min_{0-8}\left(Cost = \frac{(p_{i,des} - p_{i,curr})}{K_{1,i}} \right) \tag{9-3}$$

Thus, the solution requiring the least amount of encoder counts to get to the desired position and orientation is chosen. Those angles are then converted back to the manufacturer's values required in the calculation of the desired joint efforts. Hence, the results of the commanded position [908, -602, 512] mm and orientation [0.4650,-0.1810, -0.8030,-0.3320] are shown in Figure 9-11.

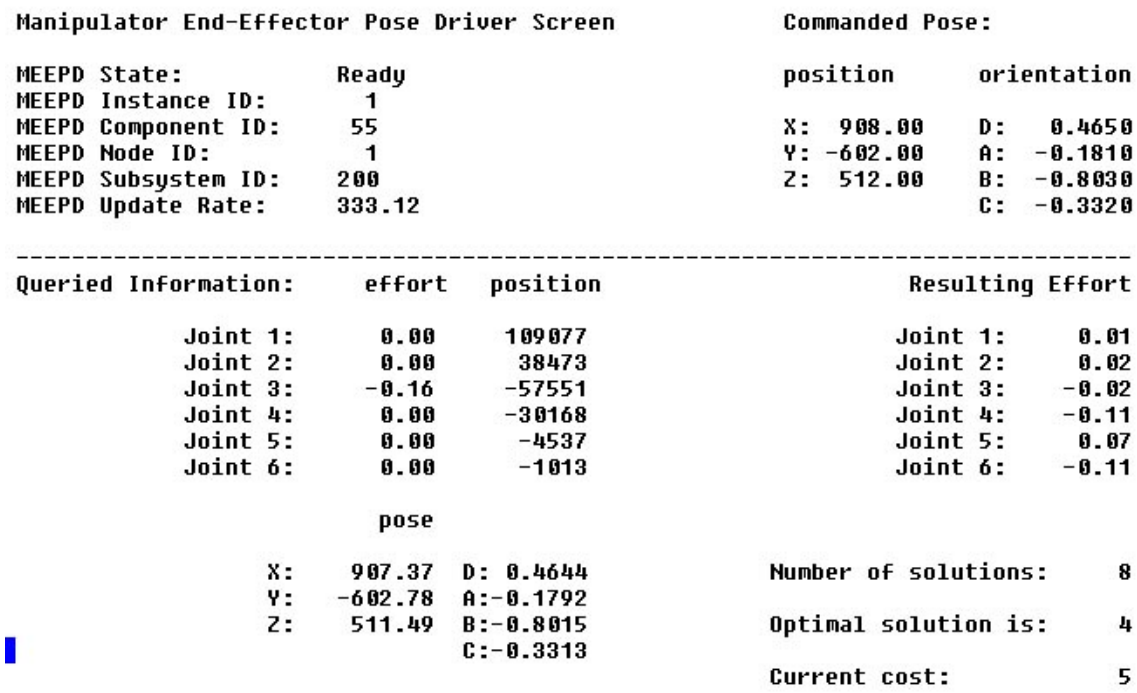

Figure 9-11. The MEEPD screen as it responds to Set End-Effector Pose message

The computation of errors associated with the resulting position and orientation is non-trivial and is not covered in this thesis. However, it should be noted that the resulting joint angle positions are within 2 encoder counts with respect to those computed using the reverse position analysis.

9.5 Case 4: Set Joint Velocities

This scenario is very similar to that covered in Section 9.2 since joint efforts are interpreted as joint velocities. The only difference is that joint velocities are now converted back to joint efforts and then back to velocities at the interface level. Figure 9-12 shows the Manipulator Joint Velocities Driver screen during the commanded motion. The individual joint velocities are calculated using Equation 6-1 with “average” parameter set defining maximum joint velocity. Therefore, a Set Joint Velocities message commanding [-3000, 2500, -3500, 500, -300, 250] for joints 1, 2, 3, 4, 5, and 6 respectively yields [-60, 50, -70, 50, -30, 25] joint efforts as shown in Figure 9-12.

Manipulator Joint Velocities Driver Screen

MJVD State:	Ready	Joint 1:	-3000
MJVD Instance ID:	1	Joint 2:	2500
MJVD Component ID:	56	Joint 3:	-3500
MJVD Node ID:	1	Joint 4:	500
MJVD Subsystem ID:	200	Joint 5:	-300
MJVD Update Rate:	333.01	Joint 6:	250

Queried Information:	effort	velocity	Resulting Effort	
Joint 1:	-59.66	-2983	Joint 1:	-60.00
Joint 2:	49.94	2497	Joint 2:	50.00
Joint 3:	-69.94	-3497	Joint 3:	-70.00
Joint 4:	50.30	503	Joint 4:	50.00
Joint 5:	-29.60	-296	Joint 5:	-30.00
Joint 6:	25.20	252	Joint 6:	25.00

Figure 9-12. The MJVD Screen as it responds to Set Joint Velocities message

It is apparent that very close values of velocity are obtained with the largest error of 0.5% occurring in motion of joint 1. This error is inherent to the theory behind the independent jogging motion, since the “JG” command is directly passed to the Galil controller.

9.6 Case 5: Set End-Effector Velocity State

This section is supposed to evaluate the performance of the Manipulator End-Effector Velocity State Driver. This component takes as input the desired velocity state of the end-effector and computes the resulting joint velocities. Any commanded motion will reach a singularity configuration if not terminated in time; hence the motion has to be constantly monitored and carefully selected. In addition, since the resulting velocities vary as the end-effector moves in a commanded direction, the “high” set of maximum parameters is used in for this application allowing for a wide range of possible solutions. Once determined, the joint velocities are converted into efforts and passed to the PM. The collection of results for this section is still in progress and might not be included in this document due to the time constraints.

9.7 Case 6: Set Joint Motion

This case is in many ways similar to Case 2, in that it uses the same approach to compute the resulting joint efforts. However, the major difference is seen as this component sets the parameters defining the trapezoidal trajectory profile with each pose. As a result, neither of the “average”, “low” or “high” sets are used in this implementation, but rather a custom set is created with each pose. In this example, the SSC sends a message to the Manipulator Joint Move Driver containing three different poses. The results as well as the resulting trajectory profiles are covered in the following sections.

9.7.1 Set Joint Motion: Pose 1

The snapshot of the MJMD screen after the completion of the first pose is shown in Figure 9-13.

```
Manipulator Joint Move Driver Screen                          Commanded Motion

MJMD State:            Ready                                  Joint 1:    45000
MJMD Instance ID:        1                                    Joint 2:    45000
MJMD Component ID:      58                                    Joint 3:    45000
MJMD Node ID:            1                                    Joint 4:     5000
MJMD Subsystem ID:     200                                    Joint 5:     5000
MJMD Update Rate:      334.34                                 Joint 6:     5000

-------------------------------------------------------------------------------
Commanded Joint Values:  max_vel   max_acc   max_dec
                                                    Current pose:          1
            Joint 1:      4886      135718    135718
            Joint 2:      4886      135718    135718    Number of poses:       3
            Joint 3:      4886      135718    135718
            Joint 4:       977      135718    135718    Next pose time:       30
            Joint 5:       977      135718    135718
            Joint 6:       977      135718    135718    Relative time:     28.06

Queried Information:     effort   position   velocity          Resulting Effort

            Joint 1:       0.00      44999          0          Joint 1:     0.03
            Joint 2:       0.00      44999          0          Joint 2:     0.03
            Joint 3:       0.00      45001          0          Joint 3:    -0.03
            Joint 4:       0.00       5001          0          Joint 4:    -0.10
            Joint 5:       0.00       5001          0          Joint 5:    -0.05
            Joint 6:       0.00       4999          0          Joint 6:     0.09
```

Figure 9-13. The MJMD screen showing component status upon completion of pose 1

The desired trapezoidal motion parameters are listed above. Figure 9-14 shows the velocity performance plot displaying the actual motion parameters for each of the joints.

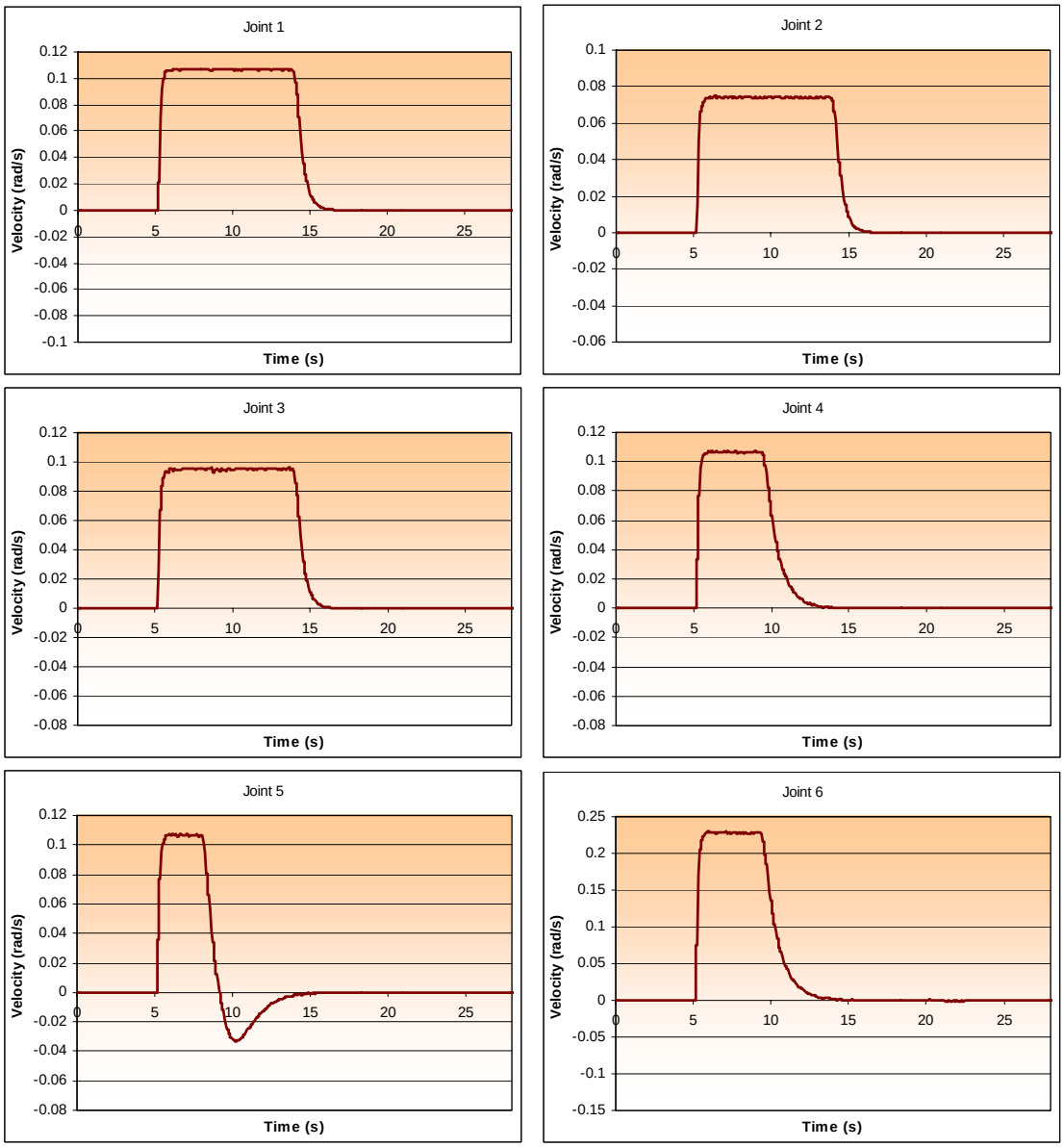

Figure 9-14. Joint performance plots for motion parameters set in pose 1

It is apparent that these plots closely follow the desired parameters with the exception to the deceleration stage which is affected by the additional control loop that was introduced in Section 9-3. In this stage, the component constantly reduces commanded velocity values that are then passed to the Galil controller which then once

again uses a trajectory generator to close the loop. The incremental position parameter used in the Galil “JG” function is not specified, thus the controller uses very small values to set the path by default. The combination of the two control loops provides additional damping. Similar behavior can be observed in the upcoming sections. Referring back to Figure 9-13, it can be seen that the resulting positions are within +/-1 encoder counts.

9.7.2 Set Joint Motion: Pose 2

The snapshot of the MJMD screen after the completion of the second pose is shown in Figure 9-15.

```
Manipulator Joint Move Driver Screen                          Commanded Motion

MJMD State:            Ready                                  Joint 1:        0
MJMD Instance ID:        1                                    Joint 2:        0
MJMD Component ID:      58                                    Joint 3:        0
MJMD Node ID:            1                                    Joint 4:        0
MJMD Subsystem ID:     200                                    Joint 5:        0
MJMD Update Rate:    19.23                                    Joint 6:        0

-------------------------------------------------------------------------------
Commanded Joint Values:  max_vel   max_acc   max_dec
                                                    Current pose:          2
            Joint 1:      3257      21715     21715
            Joint 2:      3257      21715     21715   Number of poses:       3
            Joint 3:      3257      21715     21715
            Joint 4:       543      21715     21715   Next pose time:       60
            Joint 5:       543      21715     21715
            Joint 6:       543      21715     21715   Relative time:     53.83

Queried Information:     effort   position   velocity          Resulting Effort

            Joint 1:       0.00          1          0          Joint 1:    -0.03
            Joint 2:       0.00          1          0          Joint 2:    -0.03
            Joint 3:       0.00          1          0          Joint 3:    -0.03
            Joint 4:       0.00          2          0          Joint 4:    -0.19
            Joint 5:       0.00         -2          0          Joint 5:     0.19
            Joint 6:       0.00          3          0          Joint 6:    -0.27
```

Figure 9-15. The MJMD screen showing component status upon completion of pose 2

The second set of joint positions goes in effect after 30 seconds of the component startup. The Puma was commanded to go back to its home position, which was successfully accomplished in a little over 23 seconds. The desired trapezoidal motion parameters for pose 2 are listed in Figure 9-15. Figure 9-16 depicts the velocity performance plot showing the actual motion parameters for each of the joints.

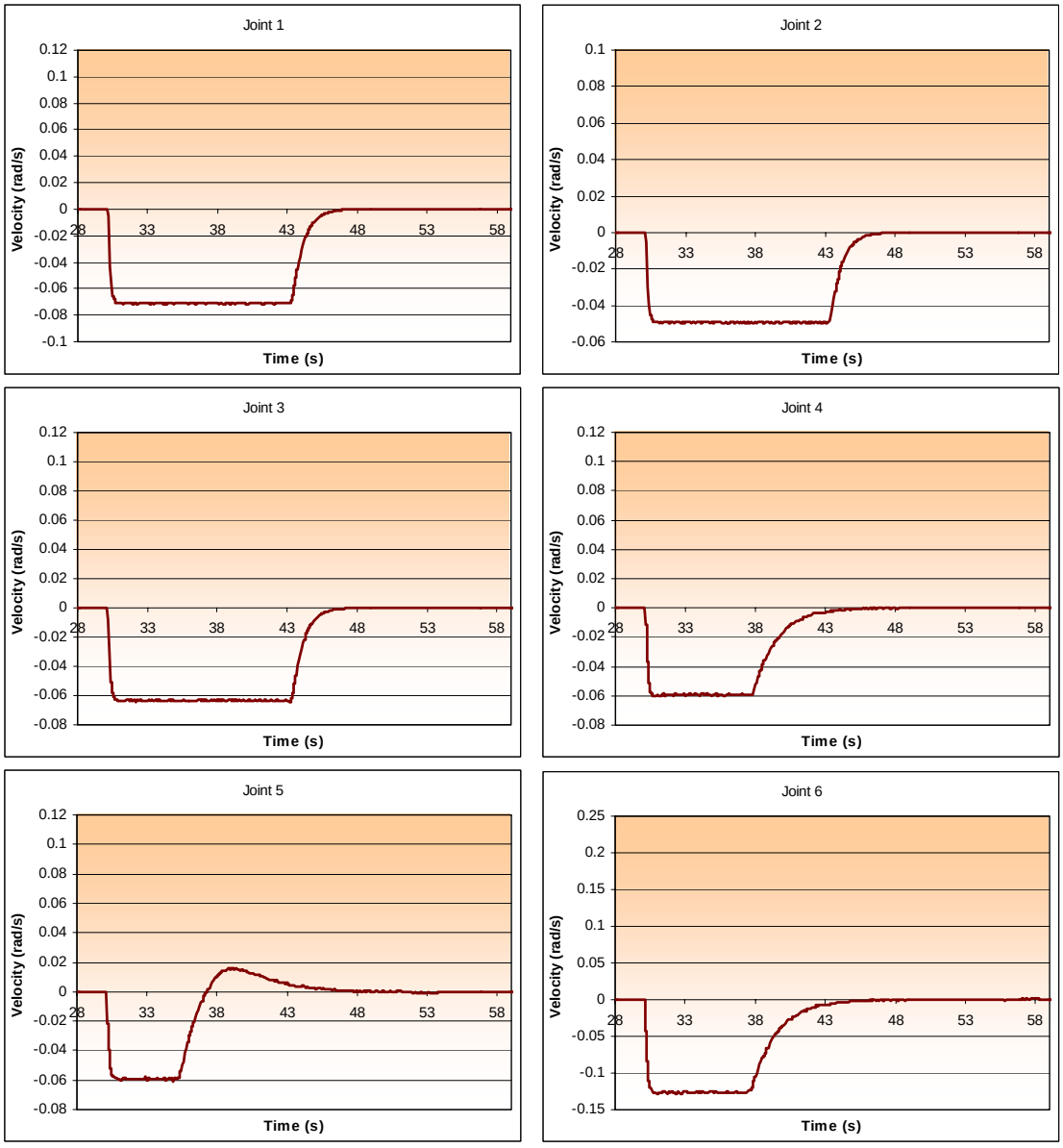

Figure 9-16. Joint performance plots for motion parameters set in pose 2

The commanded parameters resulting in the above plots are very similar to those defining the “low” set, thus implying a higher position error of +/-3 encoder counts.

9.7.3 Set Joint Motion: Pose 3

The snapshot of the MJMD screen after the completion of pose 3 is shown in Figure 9-17. The set of joint positions goes in effect after 60 seconds of the component startup. The desired trapezoidal parameters provide a mix of values (Figure 9-16).

```
Manipulator Joint Move Driver Screen                              Commanded Motion

MJMD State:              Ready                                  Joint 1:   -62000
MJMD Instance ID:          1                                    Joint 2:   -65000
MJMD Component ID:        58                                    Joint 3:    65000
MJMD Node ID:              1                                    Joint 4:     5000
MJMD Subsystem ID:       200                                    Joint 5:     5000
MJMD Update Rate:      19.33                                    Joint 6:    -5000

-------------------------------------------------------------------------------
Commanded Joint Values:  max_vel   max_acc   max_dec
                                                    Current pose:          3
            Joint 1:      1086      217148     92288
            Joint 2:      1086      217148     92288   Number of poses:       3
            Joint 3:      1086      217148     92288
            Joint 4:       326      217148     92288   Next pose time:       60
            Joint 5:       326      217148     92288
            Joint 6:       326      146518     92288   Relative time:    140.66

Queried Information:     effort   position   velocity          Resulting Effort

            Joint 1:       0.00     -61996          0          Joint 1:    -0.12
            Joint 2:       0.00     -64997          0          Joint 2:    -0.09
            Joint 3:       0.00      64996          0          Joint 3:     0.12
            Joint 4:       0.00       4996          0          Joint 4:     0.39
            Joint 5:       0.00       5004          0          Joint 5:    -0.37
            Joint 6:       0.00      -4995          0          Joint 6:    -0.45
```

Figure 9-17. The MJMD screen showing component status upon completion of pose 3

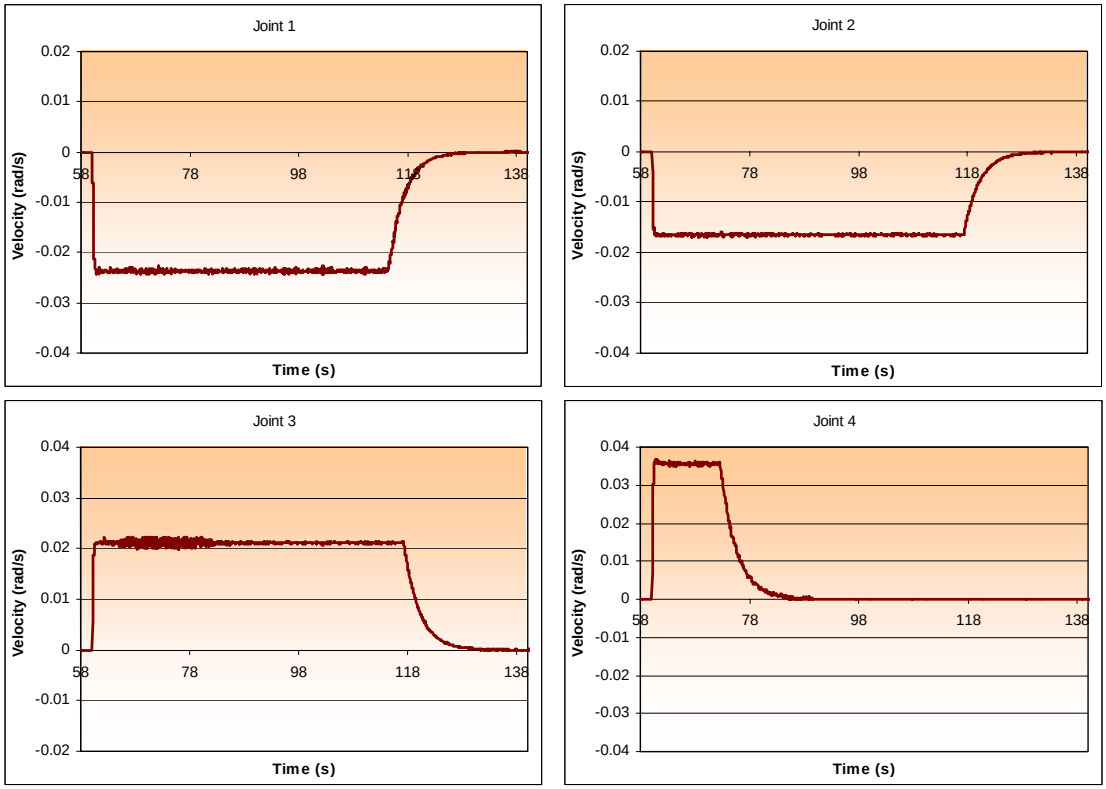

Figure 9-18a. Joint performance plots for motion parameters set in pose 3

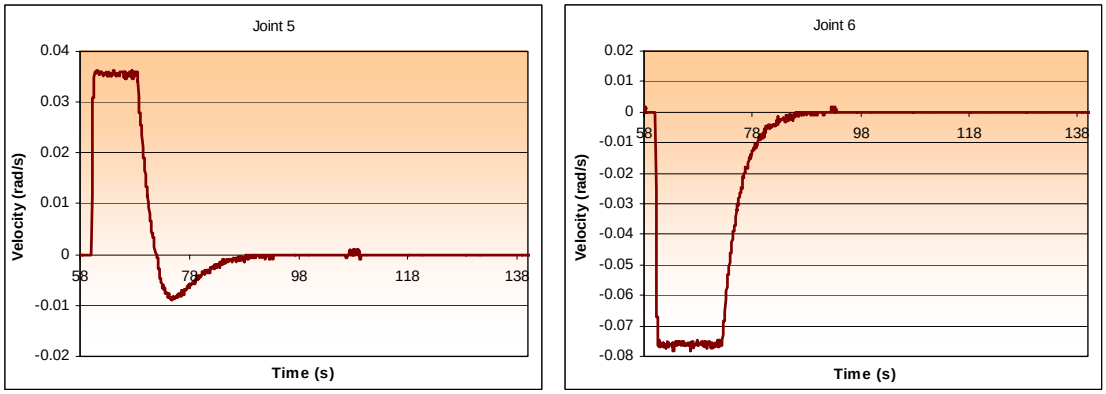

Figure 9-18b. Joint performance plots for motion parameters set in pose 3

Commanded joint positions [-62000, -65000, 65000, 5000, 5000, -5000] are reached in approximately 80 seconds with the accuracy of +/-5 encoder counts. Figures 9-18a and 9-18b show the velocity performance plot showing the actual motion parameters for each of the joints.

9.8 Case 7: Set End-Effector Path Motion

This section tests the capabilities of the Manipulator End-Effector Discrete Pose Driver. This component can be considered a superset of the MEEPD in that it fundamentally performs the same function with the ability of the user to specify the end-effector path or profile. In other words, this component takes as input the message that specifies multiple positions and orientations of the end-effector at different times. A test sample message contains five different end-effector pose sets, the two of which are shown in Figure 9-19 and Figure 9-20. The “average” set of motion parameters is used for this component by default and thus the accuracy of +/- 3 encoder counts was obtained with all the results. This chapter has successfully shown that all the information communicated between the JAUS components is done correctly. It has also illustrated that each component properly performs its required tasks, outlined in Chapters 4-7.

```
Manipulator End-Effector Discrete Pose Driver Screen   Commanded Pose:

MEEDPD State:          Ready                           position      orientation
MEEDPD Instance ID:      1
MEEDPD Component ID:    59                             X:  1055.00   D:   0.7684
MEEDPD Node ID:          1                             Y:  172.00    A:   0.3045
MEEDPD Subsystem ID:   200                             Z:  580.00    B:  -0.5537
MEEDPD Update Rate:    328.73                                        C:  -0.1012

-------------------------------------------------------------------------------
Queried Information:     effort   position                     Resulting Effort

            Joint 1:      0.00      15001                     Joint 1:    -0.01
            Joint 2:      0.00     -34945                     Joint 2:    -0.03
            Joint 3:      0.00      50090                     Joint 3:     0.02
            Joint 4:      0.00     -30258                     Joint 4:    -0.10
            Joint 5:      0.00      -2491                     Joint 5:     0.06
            Joint 6:      0.00      11466                     Joint 6:    -0.01

                          pose
                                                   Current pose:            1
                  X:    1055.11 D: 0.7684          Number of poses:         5
                  Y:    171.66  A: 0.3044          Next pose time:         45
                  Z:    579.99  B:-0.5537          Relative time:       40.18
                                C:-0.1012
                                                   Number of solutions:     8
                                                   Optimal solution is:     2
                                                   Current cost:            4
```

Figure 9-19. The MEEDPD screen after completion of pose 1

```
Manipulator End-Effector Discrete Pose Driver Screen   Commanded Pose:

MEEDPD State:          Ready                           position      orientation
MEEDPD Instance ID:      1
MEEDPD Component ID:    59                             X:  1008.00   D:   0.5244
MEEDPD Node ID:          1                             Y:  362.00    A:   0.0685
MEEDPD Subsystem ID:   200                             Z:  678.00    B:  -0.8251
MEEDPD Update Rate:    333.44                                        C:   0.1986

-------------------------------------------------------------------------------
Queried Information:     effort   position                     Resulting Effort

            Joint 1:      0.00      23971                     Joint 1:     0.01
            Joint 2:      0.00     -36053                     Joint 2:    -0.01
            Joint 3:      0.00      45952                     Joint 3:     0.01
            Joint 4:      0.00      -2582                     Joint 4:     0.14
            Joint 5:      0.00      -2395                     Joint 5:    -0.06
            Joint 6:      0.00       2489                     Joint 6:     0.10

                          pose
                                                   Current pose:            5
                  X:    1008.18 D: 0.5245          Number of poses:         5
                  Y:    361.68  A: 0.0687          Next pose time:         75
                  Z:    677.96  B:-0.8251          Relative time:      111.10
                                C: 0.1985
                                                   Number of solutions:     8
                                                   Optimal solution is:     2
                                                   Current cost:            4
```

Figure 9-20. The MEEDPD screen after completion of pose 5

Finally, the chapter defined a range of joint motion parameters which will ensure stable, minimal steady-state-error operation of the Pum 762 manipulator system. The last chapter summarizes all the information covered in this thesis and offers some ideas regarding future component development and alternative platform implementations.

CHAPTER 10
CONCLUSIONS AND FUTURE WORK

10.1 Conclusions

Our study focused on the design and development of 9 components used to command and control a manipulator arm. Considering that these components are developed for the Joint Architecture for Unmanned Systems (JAUS), this development had to meet three specific objectives. First, each of the components had to meet the criteria of a fully defined JAUS component. Second, a testing platform had to be chosen, and the functionality of the manipulator components had to be applied to the test platform. Finally, the performance of this platform had to be tested under the implementation of the new architecture.

A standard JAUS component is defined through a set of 4 criteria: the function of the component, the accepted input and output messages that the component may handle, and the full description of the component. In these terms, all 9 components were fully constrained to the JAUS architecture. The function was clearly defined for each of the components, varying from the simple open-loop control, to the discrete closed-loop end-effector position control. A full set of input and output messages were defined to meet the JAUS message standards. The component description defined the implementation of these components into the JAUS architecture, set the corresponding and unique identification numbers, and discuessed the specifics of their individual functionality.

The Puma 762 robot was chosen as the testing platform due to the availability and acceptable operating condition. The robot was reverse engineered prior to this implementation to accommodate a commercially available Galil DMC-2100 motion controller. Some of the safety functionality of the native Val II controller was kept to ensure that proper initialization routines are accomplished.

The implementation of the Primitive Manipulator component was used to interpret joint efforts as joint velocities with respect to the commands that the Galil controller used to perpetuate motion. This approach uses an independent jogging mode inherent to the Galil controller interface.

The scope of the software development during our study involved the programming of the JAUS functional elements on 4 levels. The design aspect took advantage of the multithreading tool and applied it to each of the components allowing a single program on a node level to spawn and run all of the manipulator components. On the lowest level a program had to be created to be able to interface the C based JAUS software to the Galil machine-based command language. On the highest level, the main program was used to handle the startup and shutdown routines of the Manipulator Control node. This program was further responsible for displaying all of the information pertaining to the particular component using the curses windows development tool standard to many Linux distributions.

Finally, the performance of each of the components was tested in terms of the guidelines set by the JAUS for component functionality. This was accomplished by passing the specific commanding messages to each of the components. The results were used to determine the performance specifications of the Puma robot under the new

modular architecture. More specifically, the accuracy of the resulting position and velocity is within 6 enc and 15 enc/s, respectively. Such errors, even though small, are still significant especially if high precision operation is required by the particular application. There are 3 reasons that could explain the inaccuracies. First, there is a small margin of error inherent to the independent jogging mode as it uses the trajectory generator for closed-loop position control. Second, messages sent through the node manager lose some precision during the process of packing and unpacking. Finally, a small frequency associated with the interface thread responsible for communications with the Galil controller provided delayed information to the commanding components running at much higher update rates.

Overall, the Manipulator Control node containing 9 JAUS manipulator components was tested successfully on the Puma 762 Robot. It can be assumed that these components could be now implemented on any serial manipulator system with slight modifications. Moreover, infrastructure in place allows for both teleop and autonomous control of an independent arm, as well as an arm that would be a part of the vehicle-manipulator system.

10.2 Future Work

This implementation has successfully addressed the aspects of low-level and mid-level controls of manipulators. More specific messages oriented towards advanced task completion could be added to the current node. The next major step would be to test this implementation on a vehicle-manipulator system requiring both teleop and autonomous control, as this implementation was only able to address one of the two aspects.

APPENDIX A
EQUATIONS FOR A SPHERICAL HEPTAGON

Table A-1. Fundamental formulas for a Spherical Heptagon

$X_{12345} = s_{67}s_6$	$Y_{12345} = s_{67}c_6$	$Z_{12345} = c_{67}$
$X_{23456} = s_{71}s_7$	$Y_{23456} = s_{71}c_7$	$Z_{23456} = c_{71}$
$X_{34567} = s_{12}s_1$	$Y_{34567} = s_{12}c_1$	$Z_{34567} = c_{12}$
$X_{45671} = s_{23}s_2$	$Y_{45671} = s_{23}c_2$	$Z_{45671} = c_{23}$
$X_{56712} = s_{34}s_3$	$Y_{56712} = s_{34}c_3$	$Z_{56712} = c_{34}$
$X_{67123} = s_{45}s_4$	$Y_{67123} = s_{45}c_4$	$Z_{67123} = c_{45}$
$X_{71234} = s_{56}s_5$	$Y_{71234} = s_{56}c_5$	$Z_{71234} = c_{56}$
$X_{54321} = s_{67}s_7$	$Y_{54321} = s_{67}c_7$	$Z_{54321} = c_{67}$
$X_{65432} = s_{71}s_1$	$Y_{65432} = s_{71}c_1$	$Z_{65432} = c_{71}$
$X_{76543} = s_{12}s_2$	$Y_{76543} = s_{12}c_2$	$Z_{76543} = c_{12}$
$X_{17654} = s_{23}s_3$	$Y_{17654} = s_{23}c_3$	$Z_{17654} = c_{23}$
$X_{21765} = s_{34}s_4$	$Y_{21765} = s_{34}c_4$	$Z_{21765} = c_{34}$
$X_{32176} = s_{45}s_5$	$Y_{32176} = s_{45}c_5$	$Z_{32176} = s_{45}$
$X_{43217} = s_{56}s_6$	$Y_{43217} = s_{56}c_6$	$X_{43217} = c_{56}$

Table A-2. Subsidiary formulas for a Spherical Heptagon Set 1

$X_{12345} = \overline{X}_6$	$-X^*_{12345} = \overline{Y}_6$	$Z_{12345} = \overline{Z}_6$
$X_{23456} = \overline{X}_7$	$-X^*_{23456} = \overline{Y}_7$	$Z_{23456} = \overline{Z}_1$
$X_{34567} = \overline{X}_1$	$-X^*_{34567} = \overline{Y}_1$	$Z_{34567} = \overline{Z}_2$
$X_{45671} = \overline{X}_2$	$-X^*_{45671} = \overline{Y}_2$	$Z_{45671} = \overline{Z}_3$
$X_{56712} = \overline{X}_3$	$-X^*_{56712} = \overline{Y}_3$	$Z_{56712} = \overline{Z}_4$
$X_{67123} = \overline{X}_4$	$-X^*_{67123} = \overline{Y}_4$	$Z_{67123} = \overline{Z}_5$
$X_{71234} = \overline{X}_5$	$-X^*_{71234} = \overline{Y}_5$	$Z_{71234} = \overline{Z}_6$
$X_{54321} = X_7$	$-X^*_{54321} = Y_7$	$Z_{54321} = Z_7$
$X_{65432} = X_1$	$-X^*_{65432} = Y_1$	$Z_{65432} = Z_1$
$X_{76543} = X_2$	$-X^*_{76543} = Y_2$	$Z_{76543} = Z_2$
$X_{17654} = X_3$	$-X^*_{17654} = Y_3$	$Z_{17654} = Z_3$
$X_{21765} = X_4$	$-X^*_{21765} = Y_4$	$Z_{21765} = Z_4$
$X_{32176} = X_5$	$-X^*_{32176} = Y_5$	$Z_{32176} = Z_5$
$X_{43217} = X_6$	$-X^*_{43217} = Y_6$	$X_{43217} = Z_6$

Other sets related to this configuration as well as the derivations of the direction cosines for the spatial heptagon are listed in the Appendix section of [9].

APPENDIX B
JAUS MANIPULATOR COMPONENTS: SOURCE CODE

B.1 The GalilInterface.c File and the Corresponding Header GalilInterface.h

```
////////////////////////////////////////////////////////////////////////////////
// File:               galilInterface.c
// Version:            0.1 Original Creation
// Written by:         Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:        Tom Galluzzo (galluzzt@ufl.edu)
// Date:               09/28/2004
//
// Description:        This file contains the C code used to interface JAUS to PUMA 762
//                     manipulator and the GALIL DMC2100 controller.
////////////////////////////////////////////////////////////////////////////////

#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include <sys/time.h>
#include <unistd.h>
#include <pthread.h>

#include "galilInterface.h"
#include "mcConstants.h"      // Constants used across Manipulator Control Node
#include "logLib.h"           // Debug functions
#include "timeLib.h"          // Timing functions
#include "dmclnx.h"           // Interface to the Galil Controller
#include "RoboTalk.h"         // Interface to the RoboWorks software
#include "mc.h"               // Defines getRoboWorksReady() function

// Controller Variables
long                  rc = 0;               // Return code
char                  buffer[32] = "";      // Response from the controller
char                  input_string[128] = "";// Command to the controller
HANDLEDMC             hdmc = -1;            // Handle to controller
int                   fInMotion[NUM_JOINTS]; // Motion complete flag
CONTROLLERINFO        controllerinfo;       // Controller information structure
char                  RV[] ={0x12,0x16,'\r',0x0};

// RoboTalk Variables
char                  filename[] = "Puma760Robot";  // Not case-sensitive
char                  ipAddress[] = "192.168.0.86"; // ipAddress of the computer on
                                                    //which RoboWorks is running
float   tagValues[NUM_JOINTS];
char*   TagNames[] = {"Puma_J1", "Puma_J2", "Puma_J3", "Puma_J4", "Puma_J5", "Puma_J6"};

// Ready Variables
int     interfaceReady = 0, initialReady = 0, homeReady = 0;
int     galilReady = 0, valReady = 0, armReady = 0;

// Interface Thread Variables
pthread_t             interfaceThreadId;
int                   interfaceRun;
int                   interfaceThreadRunning = FALSE;
double                interfaceThreadHz = 0;

// Other Variable Declarations
static double  jointCmdEffort[NUM_JOINTS], jointRetEffort[NUM_JOINTS];
```

```
static double  retPosition[NUM_JOINTS], retPositionEnc[NUM_JOINTS],
retPositionAbs[NUM_JOINTS];
static double   retVelocity[NUM_JOINTS], retForceTorque[NUM_JOINTS],
read_velocity[NUM_JOINTS];
static double  maxSpeed[NUM_JOINTS], maxAcceleration[NUM_JOINTS],
maxDeceleration[NUM_JOINTS];

int interfaceStartup(void)
{
	pthread_attr_t attr;
	int i;
	long index[NUM_JOINTS];

	// Initialize variables
	for (i=0; i<NUM_JOINTS; i++)
	{
		jointCmdEffort[i] = 0;
		jointRetEffort[i] = 0;
		if (i < 3) {
			maxSpeed[i] = 5000;    // Use this value as default unless other
values assigned
			maxAcceleration[i] = 150000;
			maxDeceleration[i] = 150000;
		}
		else {
			maxSpeed[i] = 1000;
			maxAcceleration[i] = 150000;
			maxDeceleration[i] = 150000;
		}
		fInMotion[i] = 0;
		index[i] = 0;
	}

	// Connect to RoboWorks 3D software
	if (getRoboWorksReady() == 1)
		Connect(filename, ipAddress);

	// Perform intialization procedures
	interfaceReady = 0;
	initManipulator(index);
	sleep(1);
//	homeManipulator();
//	sleep(3);

	//Start interface thread
	interfaceRun = TRUE;

	pthread_attr_init(&attr);
	pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
	pthread_create(&interfaceThreadId, &attr, interfaceThread, NULL);
	pthread_attr_destroy(&attr);

	// Check if initialization and homing procedures complete in order to proceed
	if (getInitialize() && getPumaHome() )
		interfaceReady = 1;
	return 0;
}

int interfaceShutdown(void)
{
	double timeOutSec;

	interfaceRun = FALSE;

	timeOutSec = getTimeSeconds() + INTERFACE_THREAD_TIMEOUT_SEC;
	while(interfaceThreadRunning)
	{
		usleep(10000);
		if(getTimeSeconds() >= timeOutSec)
		{
			pthread_cancel(interfaceThreadId);
```

```
                        interfaceThreadRunning = FALSE;
                        cError("interface: interfaceThread Shutdown Improperly\n");
                        break;
                }
        }

        // Shut down motors
    rc = DMCCommand(hdmc, "OP ?;", buffer, sizeof(buffer)); //Query actuator status
        if (rc)
        PrintError(rc);
        else
        {
        long temp;
        temp = atol(buffer);
        if (temp == 1)
        {
                rc = DMCCommand(hdmc, "OP0;", buffer, sizeof(buffer));
                if      (rc)
                        PrintError(rc);
        }
        }

        interfaceReady = 0;

        // Close connection to RoboWorks
//      if (getRoboWorksReady() == 1)
  //    Disconnect();

        // Reset Galil
        resetGalil();

    // Close connection to Galil
    rc = DMCClose(hdmc);
    if (rc)
    {
       PrintError(rc);
       return rc;
    }

        return 0;
}

void *interfaceThread(void *threadData)
{
        double time = 0, prevTime = 0;
        int i;

        interfaceThreadRunning = TRUE;
        time = getTimeSeconds();

        while(interfaceRun)
        {
                prevTime = time;
                time = getTimeSeconds();
                interfaceThreadHz = 1.0/(time-prevTime);
                motion(jointCmdEffort, jointRetEffort);
                position(retPosition, retPositionEnc, retPositionAbs);
                velocity(retVelocity);
//              forcetorque(retForceTorque);
                for (i=0; i<NUM_JOINTS; i++)
                {
                        tagValues[i] = (float)retPositionAbs[i];
                }
                usleep(100);

                // Sending position values to RoboWorks
                if (getRoboWorksReady() == 1)
                        SetTagValues(TagNames, tagValues, 6);
        }

    interfaceThreadRunning = FALSE;
```

```
	pthread_exit(NULL);
}

// Accessors
double getInterfaceUpdateRate(void){ return interfaceThreadHz; }
int getInterfaceThreadRunning(void){ return interfaceThreadRunning; }
double getJointCmdEffort(int i) { return jointCmdEffort[i]; }
double getJointRealEffort(int i) { return read_velocity[i]*100/maxSpeed[i]; }
double getJointPosition(int i) { return retPosition[i]; }
double getJointVelocity(int i) { return retVelocity[i]; }
double getJointForceTorque(int i) { return retForceTorque[i]; }
int getGalilReady(void) { return galilReady; }
int getValReady(void) { return valReady; }
int getArmReady(void) { return armReady; }
int getInitialize(void) { return initialReady; }
int getPumaHome (void) { return homeReady; }
int getInterfaceReady(void){ return interfaceReady; }

////////////////////////////////////////////////////////////////////////////////
// Private Functions
////////////////////////////////////////////////////////////////////////////////
// Function used to access values of SET_JOINT_EFFORT from the PRIMITIVE MANIPULATOR
////////////////////////////////////////////////////////////////////////////////
int setEffort(double *effort)
{
	int i;
	for (i=0; i<NUM_JOINTS; i++)
	{
		jointCmdEffort[i] = effort[i];
	}
	return 0;
}

////////////////////////////////////////////////////////////////////////////////
// This particular implementation interprets joint effort as percentage of maximum
// velocity. Thus in the interface startup routine, default maximum velocity for
// all six joints is set at 5000 enc/s. This default value is used with all but two
// JAUS components which dynamically set joint velocities.
//
// As a result, the function below is currently set up to overwrite the default
// velocity value if requested by any of the two components requiring this
// capability.
////////////////////////////////////////////////////////////////////////////////
void setMaxSpeed(double *max_speed)
{
	int i;

	for (i=0; i<NUM_JOINTS; i++)
	{
		if (i==0)
			maxSpeed[i] = max_speed[i] * J1_RAD_TO_ENC;
		else if (i==1)
			maxSpeed[i] = max_speed[i] * J2_RAD_TO_ENC;
		else if (i==2)
			maxSpeed[i] = max_speed[i] * J3_RAD_TO_ENC;
		else if (i==3)
			maxSpeed[i] = max_speed[i] * J4_RAD_TO_ENC;
		else if (i==4)
			maxSpeed[i] = max_speed[i] * J5_RAD_TO_ENC;
		else
			maxSpeed[i] = max_speed[i] * J6_RAD_TO_ENC;
	}
}

void setMaxAcceleration(double *max_acceleration)
{
	int i;

	for (i=0; i<NUM_JOINTS; i++)
	{
		if (i==0)
```

```
                        maxAcceleration[i] = max_acceleration[i] * J1_RAD_TO_ENC;
                else if (i==1)
                        maxAcceleration[i] = max_acceleration[i] * J2_RAD_TO_ENC;
                else if (i==2)
                        maxAcceleration[i] = max_acceleration[i] * J3_RAD_TO_ENC;
                else if (i==3)
                        maxAcceleration[i] = max_acceleration[i] * J4_RAD_TO_ENC;
                else if (i==4)
                        maxAcceleration[i] = max_acceleration[i] * J5_RAD_TO_ENC;
                else
                        maxAcceleration[i] = max_acceleration[i] * J6_RAD_TO_ENC;
        }
}

void setMaxDeceleration(double *max_deceleration)
{
        int i;

        for (i=0; i<NUM_JOINTS; i++)
        {
                if (i==0)
                        maxDeceleration[i] = max_deceleration[i] * J1_RAD_TO_ENC;
                else if (i==1)
                        maxDeceleration[i] = max_deceleration[i] * J2_RAD_TO_ENC;
                else if (i==2)
                        maxDeceleration[i] = max_deceleration[i] * J3_RAD_TO_ENC;
                else if (i==3)
                        maxDeceleration[i] = max_deceleration[i] * J4_RAD_TO_ENC;
                else if (i==4)
                        maxDeceleration[i] = max_deceleration[i] * J5_RAD_TO_ENC;
                else
                        maxDeceleration[i] = max_deceleration[i] * J6_RAD_TO_ENC;
        }
}

////////////////////////////////////////////////////////////////////////////////
// Function used to convert incoming joint efforts into velocity values and send
// them to the Galil controller. It returns actual effort values.
////////////////////////////////////////////////////////////////////////////////
void motion(double *jointCmdEffort, double *jointRetEffort)
{
        double cmd_velocity[NUM_JOINTS], curr_velocity[NUM_JOINTS],
read_acceleration[NUM_JOINTS];
        int     i, j=65;

        //Clip Command
        for (i=0; i<NUM_JOINTS; i++)
        {
                if (jointCmdEffort[i] > 100)
                        jointCmdEffort[i] = 100;
                if (jointCmdEffort[i] < -100)
                        jointCmdEffort[i] = -100;
        }

        //Convert commands into velocity values in units of enc/s and command motion
        for (i=0; i<NUM_JOINTS; i++)
        {
                cmd_velocity[i] = ((jointCmdEffort[i]/100) * maxSpeed[i]);
                if (i<3) {
                        if (cmd_velocity[i] > 10000)
                                cmd_velocity[i] = 10000;
                        else if (cmd_velocity[i] < -10000)
                                cmd_velocity[i] = -10000;
                }
                else {
                        if (cmd_velocity[i] > 2500)
                                cmd_velocity[i] = 2500;
                        else if (cmd_velocity[i] < -2500)
                                cmd_velocity[i] = -2500;
                }
                sprintf(input_string, "AC%c=?;",j);
```

```
                rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
                if (rc)
                        PrintError(rc);
            read_acceleration[i] = atof(buffer);
                j++;
        }
        sprintf(input_string, "JG %5.0lf,%5.0lf,%5.0lf,%5.0lf,%5.0lf,%5.0lf;",
cmd_velocity[0],cmd_velocity[1],cmd_velocity[2],cmd_velocity[3],cmd_velocity[4],cmd_veloc
ity[5]);
        rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);
        sprintf(input_string, "AC %7.0lf,%7.0lf,%7.0lf,%7.0lf,%7.0lf,%7.0lf",
maxAcceleration[0],maxAcceleration[1],maxAcceleration[2],maxAcceleration[3],maxAccelerati
on[4],maxAcceleration[5]);
        rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);
        for (i=0; i<NUM_JOINTS; i++)
        {
                cDebug(10,"Accelerations are %lf\n",read_acceleration[i]);
        }
        sprintf(input_string, "DC %7.0lf,%7.0lf,%7.0lf,%7.0lf,%7.0lf,%7.0lf;",
maxDeceleration[0],maxDeceleration[1],maxDeceleration[2],maxDeceleration[3],maxDecelerati
on[4],maxDeceleration[5]);
        rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);
        rc = DMCCommand(hdmc, "BG;", buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);

 }

////////////////////////////////////////////////////////////////////////////////
// Function used to look up position sensor data. retPositionAbs is the value used
// as RoboWorks input and has neccessary to replicate exact joint positions.
////////////////////////////////////////////////////////////////////////////////
void position(double *retPosition, double *retPositionEnc, double *retPositionAbs)
{
        int i, j=65;
        double read_position[NUM_JOINTS];

        //Read Position
        for (i=0; i<NUM_JOINTS; i++)
        {
                sprintf(input_string, "TP %c;",j);
                rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
                if (rc)
                        PrintError(rc);

            read_position[i] = atof(buffer);
                if (i == 0) {
                    retPosition[i] = read_position[i] / J1_RAD_TO_ENC; //rad
                    retPositionAbs[i] = (read_position[i])* JOINT1_ENC_TO_DEG - 90; //deg
                }
                else if (i == 1) {
                        retPosition[i] = read_position[i] / J2_RAD_TO_ENC; //rad
                        retPositionAbs[i] = (read_position[i])* JOINT2_ENC_TO_DEG - 90;
//deg
                }
                else if (i == 2) {
                        retPosition[i] = read_position[i] / J3_RAD_TO_ENC; //rad
                        retPositionAbs[i] = -(read_position[i])* JOINT3_ENC_TO_DEG - 90 ;
//deg
                }
                else if (i == 3) {
                        retPosition[i] = read_position[i] / J4_RAD_TO_ENC; //rad
                        retPositionAbs[i] = ((read_position[i])* JOINT4_ENC_TO_DEG); //deg
                }
                else if (i == 4) {
```

```
                        retPosition[i] = read_position[i] / J5_RAD_TO_ENC; //rad
                        retPositionAbs[i] = (read_position[i]) * JOINT5_ENC_TO_DEG ; //deg
                        retPosition[i] = retPosition[i] + 0.5*retPosition[i-1];
                }
                else {
                        retPosition[i] = read_position[i] / J6_RAD_TO_ENC; //rad
                        retPositionAbs[i] = (read_position[i]) * JOINT6_ENC_TO_DEG; //deg
                }
                retPositionEnc[i] = read_position[i]; //enc
                j++;
        }

}

////////////////////////////////////////////////////////////////////////////////
// Function used to look up velocity sensor data
////////////////////////////////////////////////////////////////////////////////
void velocity(double *retVelocity)
{
        int i, j=65;

        //Read Velocity
        for (i=0; i<NUM_JOINTS; i++)
        {
                sprintf(input_string, "TV %c;", j);
                rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
                if (rc)
                        PrintError(rc);
            read_velocity[i] = atof(buffer);
                if (i == 0)
                        retVelocity[i] = read_velocity[i] / J1_RAD_TO_ENC; //rad/s
                else if (i == 1)
                        retVelocity[i] = read_velocity[i] / J2_RAD_TO_ENC; //rad/s
                else if (i == 2)
                        retVelocity[i] = read_velocity[i] / J3_RAD_TO_ENC; //rad/s
                else if (i == 3)
                        retVelocity[i] = read_velocity[i] / J4_RAD_TO_ENC; //rad/s
                else if (i == 4)
                        retVelocity[i] = read_velocity[i] / J5_RAD_TO_ENC; //rad/s
                else
                        retVelocity[i] = read_velocity[i] / J6_RAD_TO_ENC; //rad/s
                j++;
        }
}
////////////////////////////////////////////////////////////////////////////////
// Function used to look up force/torque sensor data
//
// NOTE: PUMA 762 Manipulator does not have torque sensors built in, thus this
// function exists solely as an example and for potential future implementation
// on a system with torque or force sensors. Data collected below is just noise
// converted to imaginary torque values.
////////////////////////////////////////////////////////////////////////////////
void forcetorque(double *retForceTorque)
{
        int i, j=65;
        double curr_forcetorque[NUM_JOINTS], stall_forcetorque[NUM_JOINTS];
        double low_limit, high_limit, slope[NUM_JOINTS];

        //Function Parameters
        low_limit = -9.998; // Volts
        high_limit = 9.998; // Volts

        stall_forcetorque[0] = 260; //ft-lb
        stall_forcetorque[1] = 377; //ft-lb
        stall_forcetorque[2] = 294; //ft-lb
        stall_forcetorque[3] = 52; //ft-lb
        stall_forcetorque[4] = 52; //ft-lb
        stall_forcetorque[5] = 24; //ft-lb

        //Read Torque
        for (i=0; i<NUM_JOINTS; i++)
```

```
        {
                curr_forcetorque[i] = 0;
                sprintf(input_string, "TT %c;", j);
                rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
                if (rc)
                        PrintError(rc);
            curr_forcetorque[i] = atof(buffer);
            slope[i] = stall_forcetorque[i] / (high_limit - low_limit); //ft-lb/V
            retForceTorque[i] = slope[i] * curr_forcetorque[i]*FTLB_TO_NM;  //Nm
                j++;
        }
}

int resetGalil(void)
{
        rc = DMCReset(hdmc);
    if (rc)
    {
       PrintError(rc);
       return rc;
    }
    return 0;
}

////////////////////////////////////////////////////////////////////////
// Function opening the connection to the Galil controller.
////////////////////////////////////////////////////////////////////////
void startGALILController(void)
{
        memset(&controllerinfo, '\0', sizeof(controllerinfo));

        controllerinfo.cbSize = sizeof(controllerinfo);
        controllerinfo.usModelID = MODEL_2100;
        controllerinfo.fControllerType = ControllerTypeEthernet;
        controllerinfo.ulTimeout = 10000;
        controllerinfo.ulDelay = 5;
        // IP: 192.168.0.84 is the current IP address of the GALIL controller
        strcpy(controllerinfo.hardwareinfo.socketinfo.szIPAddress, "192.168.0.84");
        controllerinfo.hardwareinfo.socketinfo.fProtocol = EthernetProtocolTCP;

        DMCInitLibrary();

        // Open the connection
        rc = DMCOpen(&controllerinfo, &hdmc);
        if (rc)
        {
                PrintError(rc);
                galilReady = 0;
    }
    else
        galilReady = 1;
}

////////////////////////////////////////////////////////////////////////
// Check the status of the VAL II controller maintaing system checks
// provided by the manufacturer.
////////////////////////////////////////////////////////////////////////
void startVALController(void)
{
        int temp = 0;

        rc = DMCCommand(hdmc, "MO;", buffer, sizeof(buffer));
        if (rc)
        PrintError(rc);
    rc = DMCCommand(hdmc, "OP0;", buffer, sizeof(buffer));
        if (rc)
        PrintError(rc);
    rc = DMCCommand(hdmc, "AF 0,0,0,0,0,0;", buffer, sizeof(buffer));
        if (rc)
        PrintError(rc);
    rc = DMCCommand(hdmc, "IN_3=@IN[3];", buffer, sizeof(buffer));
```

```
        if (rc)
        PrintError(rc);
        rc = DMCCommand(hdmc, "IN_3=?;", buffer, sizeof(buffer));
        if (rc)
           PrintError(rc);
           else
        {
        temp = atol(buffer);
        }
        rc = DMCCommand(hdmc, "AT -1000;", buffer, sizeof(buffer));
        if (rc)
        PrintError(rc);
        rc = DMCCommand(hdmc, "SH;", buffer, sizeof(buffer));
        if (rc)
        PrintError(rc);
        rc = DMCCommand(hdmc, "OP1;", buffer, sizeof(buffer));
        if (rc)
        PrintError(rc);
        if (temp == 0)
                valReady = 0;
        else
                valReady = 1;
}

////////////////////////////////////////////////////////////////////////
// Check the status of the arm power (remote switch capable).
////////////////////////////////////////////////////////////////////////
void startARM(void)
{
        int temp = 0;

        rc = DMCCommand(hdmc, "IN_1=@IN[1];", buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);
        rc = DMCCommand(hdmc, "IN_1=?;", buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);
        else
        {
                temp = atol(buffer);
        }
        rc = DMCCommand(hdmc, "AT -1000;", buffer, sizeof(buffer));
        if (rc)
        PrintError(rc);
        if (temp == 0)
                armReady = 0;
        else
                armReady = 1;
}

////////////////////////////////////////////////////////////////////////
// Joint initialization is done by checking agains the stored voltage ratio
// values that correspond to a particular index. The function further finds
// the closest index thus allowing the user to know the exact position of
// the manipulator in any starting configuration.
////////////////////////////////////////////////////////////////////////
void initManipulator(long *index)
{
        int i, j;
        char k = 65;
        double min=0, diff[NUM_JOINTS][NUM_INDEX], pot[NUM_JOINTS],
stored_ratio[NUM_JOINTS][NUM_INDEX], current_ratio[NUM_JOINTS], power[NUM_JOINTS],
potpower;
        long pos[NUM_JOINTS], home_index[NUM_JOINTS], x=0, y=0, z=0, w=0;
        char index_string[32];

        for (i=0; i<NUM_JOINTS; i++)
        {
                current_ratio[i] = 0;
                fInMotion[i] = 1;
                pos[i] = 0;
```

```
                if (i<3)
                        home_index[i] = 35;
                else if (i==3)
                        home_index[i] = 35;
                else if (i==4)
                        home_index[4] = 16;
                else
                        home_index[5] = 15;
                for (j=0; j<70; j++)
                {
                        stored_ratio[i][j] = 0;
                        diff[i][j] = 0;
                }
        }

        averagePotPower(pot, &potpower);

    for (i=0; i<NUM_JOINTS; i++)
        {
            if(pot[i]>2.45)
            {
                        sprintf(index_string, "JG%c=-500;",k);
                        rc = DMCCommand(hdmc, index_string, buffer, sizeof(buffer));
                        if (rc)
                        PrintError(rc);
            }
            else
            {
                sprintf(index_string, "JG%c=500;",k);
                        rc = DMCCommand(hdmc, index_string, buffer, sizeof(buffer));
                        if (rc)
                                PrintError(rc);
            }
                k++;
        }
        sprintf(input_string, "FI ABCDEF");
        rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);
        sprintf(input_string, "BG ABCDEF;");
        rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);

        while (!rc && fInMotion[0] && fInMotion[1] && fInMotion[2] && fInMotion[3] &&
fInMotion[4] && fInMotion[5])
        {
                for (i=0; i<NUM_JOINTS; i++)
                {
                        j = 65;
                        sprintf(input_string, "MG_BG%c;", j);
                rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
                if (rc)
                        PrintError(rc);
                else
                        fInMotion[i] = atoi(buffer);
                j++;
                }
        }

        rc = DMCCommand(hdmc, "AM ABCDEF", buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);
        rc = DMCCommand(hdmc, "HX1;", buffer, sizeof(buffer));
        if (rc)
        PrintError(rc);
        rc = DMCCommand(hdmc, "SP 1000,1000,1000,1000,1000,1000;", buffer,
sizeof(buffer));
        if (rc)
        PrintError(rc);
```

```
        collectRatios(stored_ratio);

        k = 65;
        for (i=0; i<NUM_JOINTS; i++)
        {
                min = fabs(pot[i] - power[i]);
                for (j=0; j<NUM_INDEX; j++)
                {
                        power[i] = potpower * stored_ratio[i][j];
                        diff[i][j] = fabs(pot[i] - power[i]);
                        if(diff[i][j] < min)
                        {
                                index[i] = j;
                                min = diff[i][j];
                        }
                }
        if (i == 0)
                x = 3200, z = 3200, y = 1900, w = -1300;
        else if (i == 1)
                x = 3200, z = 3200, y = 0, w = -3200;
        else if (i == 2)
                x = 3200, z = 3200, y = 1200, w = -2000;
        else if (i == 3)
                x = 1000, z = 1000, y = 200, w = -800;
        else if (i == 4)
                x = 1000, z = 1000, y = 0, w = -1000;
        else
                x = 1000, z = 1000, y = 300, w = -700;

                if (index[i] <= home_index[i])
                pos[i] = (home_index[i] - index[i]) * x + y;
        else
                pos[i] = -(index[i] - home_index[i] - 1) * z + w;

                sprintf(input_string, "DP%c=%ld", k, -pos[i]);
        rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
                if (rc)
                PrintError(rc);
        k++;
        }
        rc = DMCCommand(hdmc, "SP 3000,3000,3000,1000,1000,1000;", buffer,
sizeof(buffer));
        if (rc)
        PrintError(rc);
        initialReady = 1;
}

////////////////////////////////////////////////////////////////////////
// Home the manipulator.
////////////////////////////////////////////////////////////////////////
void homeManipulator(void)
{
        int i, j=65;
        double  cmd_position[NUM_JOINTS], curr_position[NUM_JOINTS];

        for (i=0; i<NUM_JOINTS; i++)
        {
                fInMotion[i] = 1;
                sprintf(input_string, "TP %c;",j);
                rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
                if (rc)
                        PrintError(rc);
            curr_position[i] = atof(buffer);
                cmd_position[i] = 0;
                if (i < 3)
                {
                        if (fabs(curr_position[i]) > 112000) homeReady = 0;
                }
                else if (i == 3)
                {
                        if (fabs(curr_position[i]) > 35000)   homeReady = 0;
```

```
                }
                else if (i == 4)
                {
                        if (fabs(curr_position[i]) > 16000) homeReady = 0;
                }
                else
                {
                        if (fabs(curr_position[i]) > 15500)  homeReady = 0;
                }
                j++;
        }
        rc = DMCCommand(hdmc, "SP 3000,3000,3000,1000,1000,1000;", buffer,
sizeof(buffer));
        if (rc)
        PrintError(rc);
        sprintf(input_string, "PA 0,0,0,0,0,0;");
        rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);
        sprintf(input_string, "BG ABCDEF;");
        rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);

        while (!rc && fInMotion[0] && fInMotion[1] && fInMotion[2] && fInMotion[3] &&
fInMotion[4] && fInMotion[5])
        {
                for (i=0; i<NUM_JOINTS; i++)
                {
                        j = 65;
                        sprintf(input_string, "MG_BG%c;", j);
                rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
                if (rc)
                        PrintError(rc);
                else
                        fInMotion[i] = atoi(buffer);
                j++;
                }
        }
    rc = DMCCommand(hdmc, "AM ABCDEF", buffer, sizeof(buffer));
        if (rc)
                PrintError(rc);
        homeReady = 1;
}

////////////////////////////////////////////////////////////////////////
// Obtain voltage ratio values of the Galil EEPROM
////////////////////////////////////////////////////////////////////////
void collectRatios(double stored_ratio[NUM_JOINTS][NUM_INDEX])
{
    int i, j, k = 65;

    for (i=0; i<NUM_JOINTS; i++)
    {
        for (j=0; j<NUM_INDEX; j++)
        {
                        sprintf(input_string, "POT%c[%d]=?", k, j);
                rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
                        if (rc)
                        PrintError(rc);
                stored_ratio[i][j]=atof(buffer);
        }
        k++;
    }
}

////////////////////////////////////////////////////////////////////////
// Compute the average of one hundred queries to the current voltage. This
// is neccessary due to the voltage fluctuations caused by noise. Improper
// values could cause initilization to be slightly off.
////////////////////////////////////////////////////////////////////////
```

```
void averagePotPower(double *pot, double *potpower)
{
	double pott[NUM_JOINTS], temp[7], power;
	int i, j;

	power = 0;

	for(i=0; i<7; i++)
	{
		if (i<6)
			pott[i] = 0;
		if (i<7)
			temp[i] = 0;
	}

	for (i=0; i<100; i++)
	{
		for (j=1; j<8; j++)
	{
		sprintf(input_string, "AN_%d=@AN[%d]",j ,j);
		rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
			if (rc)
			PrintError(rc);
		sprintf(input_string, "AN_%d=?", j);
		rc = DMCCommand(hdmc, input_string, buffer, sizeof(buffer));
			if (rc)
			PrintError(rc);
		temp[j-1]=atof(buffer);
		if(j<7)
			pott[j-1]+=temp[j-1];
		else
			power += temp[j-1];
		}
	}
	for (j=0; j<NUM_JOINTS; j++)
	{
		pot[j]=pott[j]/100;
	}
		*potpower=power/100;

}

void PrintError(long rc)
{
   fflush(stdout);
}

////////////////////////////////////////////////////////////////////////////////
// File:               galilInterface.h
// Version:            0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Date:               09/28/2004
//
// Description:  This file contains function prototypes required for galilInterface.c
////////////////////////////////////////////////////////////////////////////////

#ifndef __GALILINTERFACE_H_
#define __GALILINTERFACE_H_

#define NUM_JOINTS     6
#define NUM_INDEX      70

#define INTERFACE_THREAD_TIMEOUT_SEC                        1.0

//public
int interfaceStartup(void);
int interfaceShutdown(void);
void *interfaceThread(void *threadData);
double getInterfaceUpdateRate(void);
int getInterfaceThreadRunning(void);
```

```
//private accessors
int setEffort(double *);
double getJointCmdEffort(int);
double getJointRealEffort(int);

double getJointPosition(int);
double getJointVelocity(int);
double getJointForceTorque(int);

int getGalilReady(void);
int getValReady(void);
int getArmReady(void);
int getInitialize(void);
int getPumaHome (void);
int getInterfaceReady(void);

double getTimeSeconds(void);

//private
int resetGalil(void);
void startGALILController(void);
void startVALController(void);
void startARM(void);
void initManipulator(long *);
void homeManipulator(void);
void setMaxSpeed(double *);
void setMaxAcceleration(double *);
void setMaxDeceleration(double *);
void motion(double *, double *);

void position(double *, double *,double *);
void velocity(double *);
void forcetorque(double *);

void averagePotPower(double *, double *);
void collectRatios(double stored_ratio[6][70]);
void PrintError(long rc);

#endif
```

B.2 The Pm.c File and the Corresponding Header Pm.h

```
////////////////////////////////////////////////////////////////////////////////
// File:              pm.c
// Version:           0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:       Tom Galluzzo (galluzzt@ufl.edu)
// Date:              09/28/2004
//
// Description:       This file contains the C code for implementation of a Primitive
//                    Manipulator JAUS component in a Linux environment. This code is
//                    designed to work with  the node manager and JAUS library software
//                    written by Jeff Witt.
////////////////////////////////////////////////////////////////////////////////

#include <jaus.h>                           // JAUS message set (USER: JAUS libraries
must be installed first)
#include <pthread.h>                // Multi-threading functions (standard to unix)
#include <timeLib.h>                // Precise timing routines (USER: must link
timeLib.c, written by Tom Galluzzo)
#include <unistd.h>                         // Unix standard functions
#include <stdio.h>                          // Unix standard input-output
#include <math.h>                           // Mathematical functions
#include <nodemgr/nodemgr.h>  // Node managment functions for sending and receiving JAUS
messages (USER: Node Manager must be installed)

#include "pm.h"                                     // File containing all function and
accessor defintions used in this component
```

```
#include "galilInterface.h"          // Access to setEffort() and getJointPosition()
functions
#include "logLib.h"                          // Debug definitions
#include "mcConstants.h"

#define DATASIZE JDATASIZE           // Default size for byte stream buffers in this file

// Private function prototypes
void *pmStateThread(void *);
void *pmNodemgrThread(void *);

// State messages set by this component
void pmStandbySsc(void);
void pmResumeSsc(void);

// Accessor to incoming information
jointEffort_t * pmSetJointEffort(void);                    // Accessor to the value of
joint efforts

// Accessors to outgoing information
jointEffort_t * pmReportJointEffort(void);
manipulatorSpecifications_t * pmReportManipulatorSpecifications(void);

unsigned char pmInstId, pmNodeId, pmSubsId;                // JAUS Instance, Node and
Subsystem IDs for this component
componentAuthority_t pmAuthority = 0;                      // JAUS Authority for this
component
int pmState = SHUTDOWN_STATE;                                      // JAUS State, running
State, and a thread running count
double pmThreadHz;                                                                 //
Stores the calculated update rate for main state thread
unsigned char pmCtrlCmptInstId, pmCtrlCmptCompId;    // Stores identity of the component
currently in control
unsigned char pmCtrlCmptNodeId, pmCtrlCmptSubsId;    // Stores identity of the component
currently in control
unsigned char pmCtrlCmptAuthority;                                 // Stores the
authority level of the component currently in control
int pmUnderControl = FALSE;                                                // FALSE - not
under control, TRUE is under control
int pmRun = FALSE;

int pmStateThreadRunning = FALSE;
pthread_t pmStateThreadId;
int pmNodemgrThreadRunning = FALSE;
pthread_t pmNodemgrThreadId;

// Structure storing incoming request under case SET_JOINT EFFORT
static jointEffort_t setJointEffort;         // Structure defining data received and sent
by this component
// Structure storing incoming request under case QUERY JOINT EFFORT
static jointEffort_t reportedJointEffort;
// Structure storing incoming request under case QUERY MANIPULATOR SPECIFICATIONS
static manipulatorSpecifications_t reportedManipulatorSpecifications;

jms_t pmJms;   // An accessor to the Node Manager JAUS Message Service (Jms) for this
component

// Function:   pmStartup
// Access:            Public
// Description: This function allows the abstracted component functionality contained in
this file to be started from an external source.
//                            It must be called first before the component state machine
and node manager interaction will begin
//                            Each call to "cmptStartup" should be followed by one call
to the "cmptShutdown" function

int pmStartup(void)
{
       pthread_attr_t attr;   // Thread attributed for the component threads spawned in
this function
```

```
	if(pmState == SHUTDOWN_STATE) // Execute the startup routines only if the
component is not running
	{
		// Check in to the Node Manager and obtain Instance, Node and Subsystem
IDs
		if(jcmCheckIn(PRIMITIVE_MANIPULATOR, &pmInstId, &pmNodeId, &pmSubsId) < 0)
// USER: Insert your component ID define on this line
		{
			cError("pm: Could not check in to nodemgr\n");
			return PM_NODE_MANAGER_CHECKIN_ERROR;
		}

		// Open a conection to the Node Manager and obtain a Jms accessor
		if((pmJms = jmsOpen(PRIMITIVE_MANIPULATOR, pmInstId, 0)) == NULL) // USER:
Insert your component ID define on this line
		{
			jcmCheckOut(PRIMITIVE_MANIPULATOR, pmInstId); // USER: Insert your
component ID define on this line
			return PM_JMS_OPEN_ERROR;
		}

		pmJms->timeout.tv_usec = 100000; // Timeout for non-blocking jmsRecv,
specified in micro seconds

		pthread_attr_init(&attr);
		pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

		// Set the state of the JAUS state machine to INITIALIZE
		pmState = INITIALIZE_STATE;
		pmRun = TRUE;

		if(pthread_create(&pmStateThreadId, &attr, pmStateThread, NULL) != 0)
		{
			cError("pm: Could not create cmptStateThread\n");
			pmShutdown();
			return PM_STATE_THREAD_CREATE_ERROR;
		}

		if(pthread_create(&pmNodemgrThreadId, &attr, pmNodemgrThread, NULL) != 0)
		{
			cError("pm: Could not create cmptNodemgrThread\n");
			pmShutdown();
			return PM_NODEMGR_THREAD_CREATE_ERROR;
		}

		pthread_attr_destroy(&attr);
	}
	else
	{
		cError("pm: Attempted startup while not shutdown\n");
		return PM_STARTUP_BEFORE_SHUTDOWN_ERROR;
	}

	return 0;
}

// Function: pmShutdown
// Access:		Public
// Description:	This function allows the abstracted component functionality
contained in this file to be stoped from an external source.
//				If the component is in the running state, this function
will terminate all threads running in this file
//				This function will also close the Jms connection to the
Node Manager and check out the component from the Node Manager

int pmShutdown(void)
{
	double timeOutSec;
	double effort[NUM_JOINTS];
	int i;
```

```
	if (pmState != SHUTDOWN_STATE) // Execute the startup routines only if the
component is not running
	{
		pmRun = FALSE;

		timeOutSec = getTimeSeconds() + PM_THREAD_TIMEOUT_SEC;
		while(pmStateThreadRunning)
		{
			usleep(100000);
			if(getTimeSeconds() >= timeOutSec)
			{
				pthread_cancel(pmStateThreadId);
				pmStateThreadRunning = FALSE;
				cError("pm: cmptStateThread Shutdown Improperly\n");
				break;
			}
		}
		timeOutSec = getTimeSeconds() + PM_THREAD_TIMEOUT_SEC;
		while(pmNodemgrThreadRunning)
		{
			usleep(100000);
			if(getTimeSeconds() >= timeOutSec)
			{
				pthread_cancel(pmNodemgrThreadId);
				pmNodemgrThreadRunning = FALSE;
				cError("pm: cmptNodemgrThread Shutdown Improperly\n");
				break;
			}
		}

		// Set jog values to zero eliminating possibility of jerky motion
		for (i=0; i<NUM_JOINTS; i++)
		{
			effort[i] = 0;
		}
		setEffort(effort);

		// Set the state of the subsystem commander to standby
		pmStandbySsc();

		// Close Node Manager Connection and check out
		jmsClose(pmJms);
		jcmCheckOut(PRIMITIVE_MANIPULATOR, pmInstId); // USER: Insert your
component ID define on this line

		pmState = SHUTDOWN_STATE;
	}

	return 0;
}

// The series of functions below allow public access to essential component information
// Access:		Public (All)
int pmGetState(void){ return pmState; }
unsigned char pmGetInstanceId(void){ return pmInstId; }
unsigned char pmGetCompId(void){ return (unsigned char)PRIMITIVE_MANIPULATOR; }
unsigned char pmGetNodeId(void){ return pmNodeId; }
unsigned char pmGetSubsystemId(void){ return pmSubsId; }
double pmGetUpdateRate(void){ return pmThreadHz; }

jointEffort_t * pmSetJointEffort(void) { return &setJointEffort; }
jointEffort_t * pmReportJointEffort(void) { return &reportedJointEffort; }
manipulatorSpecifications_t * pmReportManipulatorSpecifications(void) { return
&reportedManipulatorSpecifications; }

// Function: pmThread
// Access:		Private
// Description:	All core component functionality is contained in this thread.
//			All of the JAUS component state machine code can be found
here.
void *pmStateThread(void *threadData)
```

```
{
	jmh_t txMsgHeader;
	unsigned char data[DATASIZE];
	double time, prevTime, effort[NUM_JOINTS], curr_position[NUM_JOINTS];
	int i;

	pmStateThreadRunning = TRUE; //

	// Setup transmit header
	txMsgHeader.isExpMsg = 0;		// not experimental msg - USER: change to 1
if experimental
	txMsgHeader.isSvcMsg = 0;		// not a service connection
	txMsgHeader.acknCtrl = 0;		// no ack/nak
	txMsgHeader.priority = 6;		// default priority
	txMsgHeader.reserved = 0;
	txMsgHeader.version = 1;		// JAUS 3.0
	txMsgHeader.srcInstId = pmInstId;
	txMsgHeader.srcCompId = PRIMITIVE_MANIPULATOR;
	txMsgHeader.srcNodeId = pmNodeId;
	txMsgHeader.srcSubsId = pmSubsId;
	txMsgHeader.dataCtrl = 0;
	txMsgHeader.seqNumber = 0;

	time = getTimeSeconds();
	pmState = INITIALIZE_STATE; // Set JAUS state to INITIALIZE

	while(pmRun) // Execute state machine code while not in the SHUTDOWN state
	{
		prevTime = time;
		time = getTimeSeconds();
		pmThreadHz = 1.0/(time-prevTime); // Compute the update rate of this
thread

		switch(pmState) // Switch component behavior based on which state the
machine is in
		{
			case INITIALIZE_STATE:
				// Check for system initialization
				if(getInterfaceReady()) {
					pmState = READY_STATE;
				}
				else
					pmState = STANDBY_STATE;
				break;

			case STANDBY_STATE:
				break;

			case READY_STATE:
				// Resume system commander if in standby mode
				pmResumeSsc();
				// Command effort to the manipulator interface
				reportedJointEffort.numJoints = NUM_JOINTS;

				for(i=0; i<NUM_JOINTS; i++)
				{
					reportedJointEffort.jointEffort[i] =
getJointRealEffort(i);
					effort[i] = setJointEffort.jointEffort[i];
				}
				setEffort(effort);

				// Set up fail-safe with software limits a little lower
than mechanical limits
				for (i=0; i<NUM_JOINTS; i++)
				{
					curr_position[i] = getJointPosition(i);
					if (i < 5) {
						if (fabs(curr_position[i]) >
reportedManipulatorSpecifications.jointSpecifications[i].jointMaximumValue / 1000)
							pmState = EMERGENCY_STATE;
```

```
                                        }
                                        else
                                                if (fabs(curr_position[i]) >
reportedManipulatorSpecifications.jointNmaximumValue / 1000)
                                                        pmState = EMERGENCY_STATE;

                                }
                                break;

                        case EMERGENCY_STATE:
                                // If emergency occurs, command zero effort
                                for (i=0; i<NUM_JOINTS; i++)
                                {
                                        effort[i] = 0;
                                }
                                setEffort(effort);
                                break;

                        case FAILURE_STATE:
                                // If failure occurs, command zero effort
                                for (i=0; i<NUM_JOINTS; i++)
                                {
                                        effort[i] = 0;
                                }
                                setEffort(effort);
                                break;

                        case SHUTDOWN_STATE:
                                break;

                        default:
                                pmState = FAILURE_STATE; // The default case is undefined,
therefore go into Failure State
                                break;
                }
                usleep(1000); // Sleep for one millisecond
        }

        if(pmUnderControl)
        {
                // Terminate control of current component
                txMsgHeader.cmdCode    = REJECT_COMPONENT_CONTROL;
                txMsgHeader.dstSubsId = pmCtrlCmptSubsId;
                txMsgHeader.dstNodeId = pmCtrlCmptNodeId;
                txMsgHeader.dstCompId = pmCtrlCmptCompId;
                txMsgHeader.dstInstId = pmCtrlCmptInstId;
                txMsgHeader.dataBytes = 0;
                jmsSend(pmJms, &txMsgHeader, data, txMsgHeader.dataBytes);
        }

        // Sleep for 50 milliseconds and then exit
        usleep(50000);

        pmStateThreadRunning = FALSE;

        pthread_exit(NULL);
}

// Function: cmptNodemgrThread
// Access:             Private
// Description:        This thread is responsible for handling incoming JAUS messages from
the Node Manager JAUS message service (Jms)
//                          incoming messages are processed according to message type.
void *pmNodemgrThread(void *threadData)
{
        jmh_t rxMsgHeader, txMsgHeader; // Declare JAUS header data structures, where the
jmh_t data structure is defined in jaus.h
        unsigned char data[DATASIZE];
        createServiceConnection_t createServiceConnection;
        confirmServiceConnection_t confirmServiceConnection;
        activateServiceConnection_t activateServiceConnection;
```

```
        suspendServiceConnection_t suspendServiceConnection;
        terminateServiceConnection_t terminateServiceConnection;
        requestComponentControl_t requestComponentControl;
        confirmComponentControl_t confirmComponentControl;
        componentAuthority_t componentAuthority;
        componentStatus_t componentStatus;
        int recvCount;

        pmNodemgrThreadRunning = TRUE; // Increment the number of threads running in this
file

        // Setup Manipulator Specifications

        reportedManipulatorSpecifications.numJoints = NUM_JOINTS;
        reportedManipulatorSpecifications.jointNtype = 1;
        reportedManipulatorSpecifications.jointNoffsetOrAngle = 129; //mm
        reportedManipulatorSpecifications.jointNminimumValue = 0 / J6_RAD_TO_ENC * 1000;
//10^-3 rad
        reportedManipulatorSpecifications.jointNmaximumValue = 31000 / J6_RAD_TO_ENC *
1000; //10^-3 rad
        reportedManipulatorSpecifications.jointNmaximumVelocity = 1000 / J6_RAD_TO_ENC *
1000;//10^-3 rad/sec
        reportedManipulatorSpecifications.manipulatorCoordinateSystemX = 0;
        reportedManipulatorSpecifications.manipulatorCoordinateSystemY = 0;
        reportedManipulatorSpecifications.manipulatorCoordinateSystemZ = 0;
        reportedManipulatorSpecifications.quaternionQcomponentD = 1;
        reportedManipulatorSpecifications.quaternionQcomponentA = 0;
        reportedManipulatorSpecifications.quaternionQcomponentB = 0;
        reportedManipulatorSpecifications.quaternionQcomponentC = 0;

        reportedManipulatorSpecifications.jointSpecifications[0].jointType = 1;
        reportedManipulatorSpecifications.jointSpecifications[0].linkLength = 0.0; //mm
        reportedManipulatorSpecifications.jointSpecifications[0].twistAngle = 90 *
DEG_TO_RAD * 1000; //10^-3 rad
        reportedManipulatorSpecifications.jointSpecifications[0].jointOffsetOrAngle = 0;
//mm
        reportedManipulatorSpecifications.jointSpecifications[0].jointMinimumValue = 0 /
J1_RAD_TO_ENC * 1000; //10^-3 rad
        reportedManipulatorSpecifications.jointSpecifications[0].jointMaximumValue =
224000 / J1_RAD_TO_ENC * 1000;       //10^-3 rad
        reportedManipulatorSpecifications.jointSpecifications[0].jointMaximumVelocity =
5000 / J1_RAD_TO_ENC * 1000; //10^-3 rad/sec

        reportedManipulatorSpecifications.jointSpecifications[1].jointType = 1;
        reportedManipulatorSpecifications.jointSpecifications[1].linkLength = 650; //mm
        reportedManipulatorSpecifications.jointSpecifications[1].twistAngle = 0; //10^-3
rad
        reportedManipulatorSpecifications.jointSpecifications[1].jointOffsetOrAngle =
190;//mm
        reportedManipulatorSpecifications.jointSpecifications[1].jointMinimumValue = 0 /
J2_RAD_TO_ENC * 1000; //10^-3 rad
        reportedManipulatorSpecifications.jointSpecifications[1].jointMaximumValue =
224000 / J2_RAD_TO_ENC * 1000; //10^-3 rad
        reportedManipulatorSpecifications.jointSpecifications[1].jointMaximumVelocity =
5000 / J2_RAD_TO_ENC * 1000; //10^-3 rad/sec

        reportedManipulatorSpecifications.jointSpecifications[2].jointType = 1;
        reportedManipulatorSpecifications.jointSpecifications[2].linkLength = 0.0;//mm
        reportedManipulatorSpecifications.jointSpecifications[2].twistAngle = 270 *
DEG_TO_RAD * 1000; //10^-3 rad
        reportedManipulatorSpecifications.jointSpecifications[2].jointOffsetOrAngle = 0;
//mm
        reportedManipulatorSpecifications.jointSpecifications[2].jointMinimumValue = 0 /
J3_RAD_TO_ENC * 1000; //10^-3 rad
        reportedManipulatorSpecifications.jointSpecifications[2].jointMaximumValue =
224000 / J3_RAD_TO_ENC * 1000;//10^-3 rad
        reportedManipulatorSpecifications.jointSpecifications[2].jointMaximumVelocity =
5000 / J3_RAD_TO_ENC * 1000;  //10^-3 rad/sec

        reportedManipulatorSpecifications.jointSpecifications[3].jointType = 1;
        reportedManipulatorSpecifications.jointSpecifications[3].linkLength = 0.0; //mm
```

```
	reportedManipulatorSpecifications.jointSpecifications[3].twistAngle = 90 *
DEG_TO_RAD * 1000; //10^-3 rad
	reportedManipulatorSpecifications.jointSpecifications[3].jointOffsetOrAngle = 600;
	//mm
	reportedManipulatorSpecifications.jointSpecifications[3].jointMinimumValue = 0 /
J4_RAD_TO_ENC * 1000; //10^-3 rad
	reportedManipulatorSpecifications.jointSpecifications[3].jointMaximumValue = 70000
/ J4_RAD_TO_ENC * 1000;	//10^-3 rad
	reportedManipulatorSpecifications.jointSpecifications[3].jointMaximumVelocity =
1000 / J4_RAD_TO_ENC * 1000; //10^-3 rad/sec

	reportedManipulatorSpecifications.jointSpecifications[4].jointType = 1;
	reportedManipulatorSpecifications.jointSpecifications[4].linkLength = 0.0; //mm
	reportedManipulatorSpecifications.jointSpecifications[4].twistAngle = 90 *
DEG_TO_RAD * 1000; //10^-3 rad
	reportedManipulatorSpecifications.jointSpecifications[4].jointOffsetOrAngle = 0;
//mm
	reportedManipulatorSpecifications.jointSpecifications[4].jointMinimumValue = 0 /
J5_RAD_TO_ENC * 1000; //10^-3 rad
	reportedManipulatorSpecifications.jointSpecifications[4].jointMaximumValue = 32000
/ J5_RAD_TO_ENC * 1000; //10^-3 rad
	reportedManipulatorSpecifications.jointSpecifications[4].jointMaximumVelocity =
1000 / J5_RAD_TO_ENC * 1000; //10^-3 rad/sec

	// Setup transmit header
	txMsgHeader.isExpMsg = 0;		// not experimental msg - USER: change to 1
if experimental
	txMsgHeader.isSvcMsg = 0;		// not a service connection
	txMsgHeader.acknCtrl = 0;		// no ack/nak
	txMsgHeader.priority = 6;		// default priority
	txMsgHeader.reserved = 0;
	txMsgHeader.version = 1;		// JAUS 3.0
	txMsgHeader.srcInstId = pmInstId;
	txMsgHeader.srcCompId = PRIMITIVE_MANIPULATOR;
	txMsgHeader.srcNodeId = pmNodeId;
	txMsgHeader.srcSubsId = pmSubsId;
	txMsgHeader.dataCtrl = 0;
	txMsgHeader.seqNumber = 0;

	while(pmRun)  // Execute incoming JAUS message processing while not in the
SHUTDOWN state
	{
		// Check for a new message in the Node Manager JMS queue
		// If a message is present then store the header information in
rxMsgHeader, and store the message content information in the data buffer
		recvCount = jmsRecv(pmJms, &rxMsgHeader, data, DATASIZE);
		if (recvCount < 0)
		{
			cError("pm: Node manager jmsRecv returned error: %d\n", recvCount);
			break;
		}
		if (recvCount == 0) continue;

		// This block of code is intended to reject commands from non-controlling
components
		// However, the one exception allowed is a REQUEST_COMPONENT_CONTROL
		if(pmUnderControl && rxMsgHeader.cmdCode < 0x2000) // If not currently
under control or it's not a command, then go to the switch statement
		{
			// If the source component isn't the one in control, and it's not a
request to gain control, then exit this iteration of the while loop
			if(!(pmCtrlCmptInstId==rxMsgHeader.srcInstId &&
				pmCtrlCmptCompId==rxMsgHeader.srcCompId &&
				pmCtrlCmptNodeId==rxMsgHeader.srcNodeId &&
				pmCtrlCmptSubsId==rxMsgHeader.srcSubsId)&&
			   !(rxMsgHeader.cmdCode==REQUEST_COMPONENT_CONTROL)
			)continue; // to next iteration
		}

		switch(rxMsgHeader.cmdCode) // Switch the processing algorithm according
to the JAUS message type
```

```
		{
			// Set the component authority according to the incoming authority
code
			case SET_COMPONENT_AUTHORITY:
				convertComponentAuthority(data, &pmAuthority, DATASIZE,
UNPACK);	// Unpack and store the incoming authority code
				break;

			case SHUTDOWN:
				pmState = SHUTDOWN_STATE;
				break;

			case STANDBY:
				if(pmState == READY_STATE)
					pmState = STANDBY_STATE;
				break;

			case RESUME:
				if(pmState == STANDBY_STATE)
					pmState = READY_STATE;
				break;

			case RESET:
				pmState = INITIALIZE_STATE;
				break;

			case SET_EMERGENCY:
				pmState = EMERGENCY_STATE;
				break;

			case CLEAR_EMERGENCY:
				pmState = STANDBY_STATE;
				break;

			case CREATE_SERVICE_CONNECTION:
				convertCreateServiceConnection(data,
&createServiceConnection, DATASIZE, UNPACK);
				// USER: Insert code here to handle the create service
connection message
				break;

			case CONFIRM_SERVICE_CONNECTION:
				convertConfirmServiceConnection(data,
&confirmServiceConnection, DATASIZE, UNPACK);
				// USER: Insert code here to handle the confirm service
connection message
				break;

			case ACTIVATE_SERVICE_CONNECTION:
				convertActivateServiceConnection(data,
&activateServiceConnection, DATASIZE, UNPACK);
				// USER: Insert code here to handle the activate service
connection message
				break;

			case SUSPEND_SERVICE_CONNECTION:
				convertSuspendServiceConnection(data,
&suspendServiceConnection, DATASIZE, UNPACK);
				// USER: Insert code here to handle the suspend service
connection message
				break;

			case TERMINATE_SERVICE_CONNECTION:
				convertTerminateServiceConnection(data,
&terminateServiceConnection, DATASIZE, UNPACK);
				// USER: Insert code here to handle the terminate service
connection message
				break;

			case REQUEST_COMPONENT_CONTROL:
```

```
                        convertRequestComponentControl(data,
&requestComponentControl, DATASIZE, UNPACK);

                        if(pmUnderControl)
                        {
                                if(requestComponentControl > pmCtrlCmptAuthority) //
Test for higher authority
                                {
                                        // Terminate control of current component
                                        txMsgHeader.cmdCode    =
REJECT_COMPONENT_CONTROL;
                                        txMsgHeader.dstSubsId = pmCtrlCmptSubsId;
                                        txMsgHeader.dstNodeId = pmCtrlCmptNodeId;
                                        txMsgHeader.dstCompId = pmCtrlCmptCompId;
                                        txMsgHeader.dstInstId = pmCtrlCmptInstId;
                                        txMsgHeader.dataBytes = 0;
                                        jmsSend(pmJms, &txMsgHeader, data,
txMsgHeader.dataBytes);

                                        // Accept control of new component
                                        txMsgHeader.cmdCode    =
CONFIRM_COMPONENT_CONTROL;
                                        txMsgHeader.dstSubsId =
rxMsgHeader.srcSubsId;
                                        txMsgHeader.dstNodeId =
rxMsgHeader.srcNodeId;
                                        txMsgHeader.dstCompId =
rxMsgHeader.srcCompId;
                                        txMsgHeader.dstInstId =
rxMsgHeader.srcInstId;
                                        confirmComponentControl.asByte = 0;
                                        txMsgHeader.dataBytes =
convertConfirmComponentControl(data, &confirmComponentControl, DATASIZE, PACK);
                                        jmsSend(pmJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                        pmCtrlCmptInstId = rxMsgHeader.srcInstId;
                                        pmCtrlCmptCompId = rxMsgHeader.srcCompId;
                                        pmCtrlCmptNodeId = rxMsgHeader.srcNodeId;
                                        pmCtrlCmptSubsId = rxMsgHeader.srcSubsId;
                                        pmCtrlCmptAuthority =
requestComponentControl;
                                }
                                else
                                {
                                        if(     rxMsgHeader.srcSubsId !=
pmCtrlCmptSubsId || rxMsgHeader.srcNodeId != pmCtrlCmptNodeId ||
                                                rxMsgHeader.srcCompId !=
pmCtrlCmptCompId || rxMsgHeader.srcInstId != pmCtrlCmptInstId )
                                        {
                                                txMsgHeader.cmdCode    =
REJECT_COMPONENT_CONTROL;
                                                txMsgHeader.dstSubsId =
rxMsgHeader.srcSubsId;
                                                txMsgHeader.dstNodeId =
rxMsgHeader.srcNodeId;
                                                txMsgHeader.dstCompId =
rxMsgHeader.srcCompId;
                                                txMsgHeader.dstInstId =
rxMsgHeader.srcInstId;
                                                txMsgHeader.dataBytes = 0;
                                                jmsSend(pmJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                        }
                                }
                        }
                        else // Not currently under component control, so give
control
                        {
                                txMsgHeader.cmdCode    = CONFIRM_COMPONENT_CONTROL;
                                txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
```

```
                                    txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                    txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                    confirmComponentControl.asByte = 0;
                                    txMsgHeader.dataBytes =
convertConfirmComponentControl(data, &confirmComponentControl, DATASIZE, PACK);
                                    jmsSend(pmJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                    pmCtrlCmptInstId = rxMsgHeader.srcInstId;
                                    pmCtrlCmptCompId = rxMsgHeader.srcCompId;
                                    pmCtrlCmptNodeId = rxMsgHeader.srcNodeId;
                                    pmCtrlCmptSubsId = rxMsgHeader.srcSubsId;
                                    pmCtrlCmptAuthority = requestComponentControl;
                                    pmUnderControl = TRUE;
                             }
                             break;

                      case RELEASE_COMPONENT_CONTROL:
                             pmUnderControl = FALSE;
                             break;

                      case CONFIRM_COMPONENT_CONTROL:
                             convertConfirmComponentControl(data,
&confirmComponentControl, DATASIZE, UNPACK);
                             // USER: Insert code here to handle the confirm component
control message if needed
                             break;

                      case REJECT_COMPONENT_CONTROL:
                             // USER: Insert code here to handle the reject component
control message if needed
                             break;

                      case QUERY_COMPONENT_AUTHORITY:
                             txMsgHeader.cmdCode    = REPORT_COMPONENT_AUTHORITY;
                             txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                             txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                             txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                             txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                             componentAuthority = pmAuthority;
                             txMsgHeader.dataBytes = convertComponentAuthority(data,
&componentAuthority, DATASIZE, PACK);
                             jmsSend(pmJms, &txMsgHeader, data, txMsgHeader.dataBytes);
                             break;

                      case QUERY_COMPONENT_STATUS:
                             txMsgHeader.cmdCode    = REPORT_COMPONENT_STATUS;
                             txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                             txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                             txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                             txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                             componentStatus.primaryStatus.asByte = pmState;
                             txMsgHeader.dataBytes = convertComponentStatus(data,
&componentStatus, DATASIZE, PACK);
                             jmsSend(pmJms, &txMsgHeader, data, txMsgHeader.dataBytes);
                             break;

                      case REPORT_COMPONENT_AUTHORITY:
                             convertComponentAuthority(data, &componentAuthority,
DATASIZE, UNPACK);
                             // USER: Insert code here to handle the report component
authority message if needed
                             break;

                      case REPORT_COMPONENT_STATUS:
                             convertComponentStatus(data, &componentStatus, DATASIZE,
UNPACK);
                             break;

                      case SET_JOINT_EFFORT:
                             convertJointEffort(data, &setJointEffort, DATASIZE,
UNPACK);
```

```
                                break;

                        case QUERY_JOINT_EFFORT:
                            txMsgHeader.cmdCode = REPORT_JOINT_EFFORT;
                            txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                            txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                            txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                            txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                            txMsgHeader.dataBytes = convertJointEffort(data,
&reportedJointEffort, DATASIZE, PACK);
                                jmsSend(pmJms, &txMsgHeader, data, txMsgHeader.dataBytes);
                                break;

                        case QUERY_MANIPULATOR_SPECIFICATIONS:
                                // unpack presence vector
                                txMsgHeader.cmdCode = REPORT_MANIPULATOR_SPECIFICATIONS;
                                txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                txMsgHeader.dataBytes =
convertManipulatorSpecifications(data, &reportedManipulatorSpecifications, DATASIZE,
PACK);
                                jmsSend(pmJms, &txMsgHeader, data, txMsgHeader.dataBytes);
                                break;

                        default:
                                break;
                }
        }

        pmNodemgrThreadRunning = FALSE;
        pthread_exit(NULL);
}

void pmStandbySsc(void)
{
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;

        // Setup Transmit msg
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcInstId = pmInstId;
        txMsg.srcCompId = PRIMITIVE_MANIPULATOR;
        txMsg.srcNodeId = pmNodeId;
        txMsg.srcSubsId = pmSubsId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = STANDBY;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = SUBSYSTEM_COMMANDER;
        txMsg.dstNodeId = pmNodeId;
        txMsg.dstSubsId = pmSubsId ;
        txMsg.dataBytes = 0;

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(pmJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}

void pmResumeSsc(void)
{
```

```
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;

        // Setup Transmit msg
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcInstId = pmInstId;
        txMsg.srcCompId = PRIMITIVE_MANIPULATOR;
        txMsg.srcNodeId = pmNodeId;
        txMsg.srcSubsId = pmSubsId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = RESUME;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = SUBSYSTEM_COMMANDER;
        txMsg.dstNodeId = pmNodeId;
        txMsg.dstSubsId = pmSubsId ;
        txMsg.dataBytes = 0;

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(pmJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}

////////////////////////////////////////////////////////////////////////////////
// File:              pm.h
// Version:           0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:       Tom Galluzzo (galluzzt@ufl.edu)
// Date:              09/28/2004
//
// Description:       This file contains the skeleton C header file code for implementing
//                          the pm.c file
////////////////////////////////////////////////////////////////////////////////

#ifndef PM_H
#define PM_H

#include "jaus.h"

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE 1
#endif

#define PM_NODE_MANAGER_CHECKIN_ERROR        -1
#define PM_JMS_OPEN_ERROR                                -2
#define PM_STARTUP_BEFORE_SHUTDOWN_ERROR     -3
#define PM_STATE_THREAD_CREATE_ERROR         -4
#define PM_NODEMGR_THREAD_CREATE_ERROR               -5

#define PM_THREAD_TIMEOUT_SEC                        1.0

// Public
int pmStartup(void);
int pmShutdown(void);
int pmGetState(void);
unsigned char pmGetInstanceId(void);
unsigned char pmGetCompId(void);
unsigned char pmGetNodeId(void);
unsigned char pmGetSubsystemId(void);
```

```
double pmGetUpdateRate(void);

// State messages set by this component
void pmResetSsc(void);
void pmResumeSsc(void);

// Private accessors
jointEffort_t * pmSetJointEffort(void);
jointEffort_t * pmReportJointEffort(void);
manipulatorSpecifications_t * pmReportManipulatorSpecifications(void);

#endif // PM_H
```

B.3 The Mjps.c File and the Corresponding Header Mjps.h

```
////////////////////////////////////////////////////////////////////////////////
// File:              mjps.c
// Version:           0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:       Tom Galluzzo (galluzzt@ufl.edu)
// Date:              09/28/2004
//
// Description:       This file contains the C code for implementation of a Manipulator
//                    Joint Position Sensor JAUS component in a Linux environment. This
//                    code is designed to work with the node manager and JAUS library
//                    software written by Jeff Witt.
////////////////////////////////////////////////////////////////////////////////

#include <jaus.h>                           // JAUS message set (USER: JAUS libraries
must be installed first)
#include <pthread.h>                // Multi-threading functions (standard to unix)
#include <timeLib.h>                // Precise timing routines (USER: must link
timeLib.c, written by Tom Galluzzo)
#include <unistd.h>                         // Unix standard functions
#include <stdio.h>                          // Unix standard input-output
#include <math.h>                           // Mathematical functions
#include <nodemgr/nodemgr.h>  // Node managment functions for sending and receiving JAUS
messages (USER: Node Manager must be installed)

#include "mjps.h"     // USER: Implement and rename this header file. Include prototypes
for all public functions contained in this file.
#include "mcConstants.h"
#include "galilInterface.h" // Access to the getJointPosition() function
#include "logLib.h"

#define DATASIZE JDATASIZE    // Default size for byte stream buffers in this file

// Private function prototypes
void *mjpsStateThread(void *);
void *mjpsNodemgrThread(void *);

// Query messages requested by this component
void mjpsQueryPmComponentStatus(void);

// Accessor to outgoing information
jointPosition_t * mjpsReportJointPosition(void);

unsigned char mjpsInstId, mjpsNodeId, mjpsSubsId;          // JAUS Instance, Node and
Subsystem IDs for this component
componentAuthority_t mjpsAuthority = 0;                                  // JAUS
Authority for this component
int mjpsState = SHUTDOWN_STATE;                                                 // JAUS
State
double mjpsThreadHz;                                                           //
Stores the calculated update rate for main state thread
unsigned char mjpsCtrlCmptInstId, mjpsCtrlCmptCompId;      // Stores identity of the
component currently in control
unsigned char mjpsCtrlCmptNodeId, mjpsCtrlCmptSubsId;      // Stores identity of the
component currently in control
```

```
unsigned char mjpsCtrlCmptAuthority;                                    // Stores the
authority level of the component currently in control
int mjpsUnderControl = FALSE;                                                   //
FALSE - not under control, TRUE is under control
int mjpsRun = FALSE;

int mjpsStateThreadRunning = FALSE;
pthread_t mjpsStateThreadId;                                            // pthread
node manager thread identifier
int mjpsNodemgrThreadRunning = FALSE;
pthread_t mjpsNodemgrThreadId;                                                  //
pthread node manager thread identifier

// Structure storing incoming request under case QUERY_JOINT_POSITION
static jointPosition_t reportedJointPosition;

jmh_t mjpsScHeader;
int mjpsScActive = FALSE;

jms_t mjpsJms; // An accessor to the Node Manager JAUS Message Service (Jms) for this
component

int mjpsPmState = -1;
unsigned char mjpsPmNodeId = 0;
int mjpsPmControl = FALSE;

static double curr_theta[NUM_JOINTS];

// Function:   mjpsStartup
// Access:            Public
// Description: This function allows the abstracted component functionality contained in
this file to be started from an external source.
//                            It must be called first before the component state machine
and node manager interaction will begin
//                            Each call to "mjpsStartup" should be followed by one call
to the "cmptShutdown" function
int mjpsStartup(void)
{
       pthread_attr_t attr;  // Thread attributed for the component threads spawned in
this function

       if(mjpsState == SHUTDOWN_STATE)      // Execute the startup routines only if the
component is not running
       {
              // Check in to the Node Manager and obtain Instance, Node and Subsystem
IDs
              if(jcmCheckIn(MANIPULATOR_JOINT_POSITION_SENSOR, &mjpsInstId, &mjpsNodeId,
&mjpsSubsId) < 0) // USER: Insert your component ID define on this line
              {
                            cError("mjps: Could not check in to nodemgr\n");
                            return MJPS_NODE_MANAGER_CHECKIN_ERROR;
              }

              // Open a conection to the Node Manager and obtain a Jms accessor
              if((mjpsJms = jmsOpen(MANIPULATOR_JOINT_POSITION_SENSOR, mjpsInstId, 0))
== NULL) // USER: Insert your component ID define on this line
              {
                     cError("mjps: Could not open connection to nodemgr\n");
                     jcmCheckOut(MANIPULATOR_JOINT_POSITION_SENSOR, mjpsInstId); //
USER: Insert your component ID define on this line
                     return MJPS_JMS_OPEN_ERROR;
              }

              mjpsJms->timeout.tv_usec = 100000; // Timeout for non-blocking jmsRecv,
specified in micro seconds

              pthread_attr_init(&attr);
              pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

              mjpsState = INITIALIZE_STATE; // Set the state of the JAUS state machine
to INITIALIZE
```

```
		mjpsRun = TRUE;

		if(pthread_create(&mjpsStateThreadId, &attr, mjpsStateThread, NULL) != 0)
		{
			cError("mjps: Could not create mjpsStateThread\n");
			mjpsShutdown();
			return MJPS_STATE_THREAD_CREATE_ERROR;
		}

		if(pthread_create(&mjpsNodemgrThreadId, &attr, mjpsNodemgrThread, NULL) !=
0)
		{
			cError("mjps: Could not create mjpsNodemgrThread\n");
			mjpsShutdown();
			return MJPS_NODEMGR_THREAD_CREATE_ERROR;
		}

		pthread_attr_destroy(&attr);
	}
	else
	{
		cError("mjps: Attempted startup while not shutdown\n");
		return MJPS_STARTUP_BEFORE_SHUTDOWN_ERROR;
	}

	return 0;
}

// Function: mjpsShutdown
// Access:		Public
// Description:	This function allows the abstracted component functionality
contained in this file to be stoped from an external source.
//				If the component is in the running state, this function
will terminate all threads running in this file
//				This function will also close the Jms connection to the
Node Manager and check out the component from the Node Manager
int mjpsShutdown(void)
{
	double timeOutSec;

	if(mjpsState != SHUTDOWN_STATE)	// Execute the startup routines only if the
component is not running
	{
		mjpsRun = FALSE;

		timeOutSec = getTimeSeconds() + MJPS_THREAD_TIMEOUT_SEC;
		while(mjpsStateThreadRunning)
		{
			usleep(100000);
			if(getTimeSeconds() >= timeOutSec)
			{
				pthread_cancel(mjpsStateThreadId);
				mjpsStateThreadRunning = FALSE;
				cError("mjps: mjpsStateThread Shutdown Improperly\n");
				break;
			}
		}

		timeOutSec = getTimeSeconds() + MJPS_THREAD_TIMEOUT_SEC;
		while(mjpsNodemgrThreadRunning)
		{
			usleep(100000);
			if(getTimeSeconds() >= timeOutSec)
			{
				pthread_cancel(mjpsNodemgrThreadId);
				mjpsNodemgrThreadRunning = FALSE;
				cError("mjps: mjpsNodemgrThread Shutdown Improperly\n");
				break;
			}
		}
```

```
		// Close Node Manager Connection and check out
		jmsClose(mjpsJms);
		jcmCheckOut(MANIPULATOR_JOINT_POSITION_SENSOR, mjpsInstId); // USER:
Insert your component ID define on this line

		mjpsState = SHUTDOWN_STATE;
	}

	return 0;
}

// The series of functions below allow public access to essential component information
// Access:		Public (All)
int mjpsGetState(void){ return mjpsState; }
unsigned char mjpsGetInstanceId(void){ return mjpsInstId; }
unsigned char mjpsGetCompId(void){ return (unsigned
char)MANIPULATOR_JOINT_POSITION_SENSOR; }
unsigned char mjpsGetNodeId(void){ return mjpsNodeId; }
unsigned char mjpsGetSubsystemId(void){ return mjpsSubsId; }
double mjpsGetUpdateRate(void){ return mjpsThreadHz; }
int mjpsGetPmState(void){ return mjpsPmState; }

// USER: Insert any additional public variable accessors here
jointPosition_t * mjpsReportJointPosition(void) { return &reportedJointPosition; }

// Function: mjpsThread
// Access:		Private
// Description:	All core component functionality is contained in this thread.
//			All of the JAUS component state machine code can be found
here.
void *mjpsStateThread(void *threadData)
{
	jmh_t txMsgHeader;
	double time, prevTime;
	unsigned char data[DATASIZE];
	int i;

	mjpsStateThreadRunning = TRUE; //

	// Setup transmit header
	txMsgHeader.isExpMsg = 0;		// not experimental msg - USER: change to 1
if experimental
	txMsgHeader.isSvcMsg = 0;		// not a service connection
	txMsgHeader.acknCtrl = 0;		// no ack/nak
	txMsgHeader.priority = 6;		// default priority
	txMsgHeader.reserved = 0;
	txMsgHeader.version = 1;		// JAUS 3.0
	txMsgHeader.srcInstId = mjpsInstId;
	txMsgHeader.srcCompId = MANIPULATOR_JOINT_POSITION_SENSOR;
	txMsgHeader.srcNodeId = mjpsNodeId;
	txMsgHeader.srcSubsId = mjpsSubsId;
	txMsgHeader.dataCtrl = 0;
	txMsgHeader.seqNumber = 0;

	time = getTimeSeconds();
	mjpsState = INITIALIZE_STATE; // Set JAUS state to INITIALIZE

	while(mjpsRun) // Execute state machine code while not in the SHUTDOWN state
	{
		prevTime = time;
		time = getTimeSeconds();
		mjpsThreadHz = 1.0/(time-prevTime); // Compute the update rate of this
thread

		switch(mjpsState) // Switch component behavior based on which state the
machine is in
		{
			case INITIALIZE_STATE:
				//Check the status of PRIMITIVE MANIPULATOR
				mjpsQueryPmComponentStatus();
```

```
                                if(mjpsPmState == EMERGENCY_STATE)
                                {
                                        cError("mjps: PM in emergency state, switching to
emergency state");
                                        mjpsState = EMERGENCY_STATE;
                                }
                                if(mjpsPmState == STANDBY_STATE)
                                {
                                        txMsgHeader.cmdCode = RESUME;
                                        txMsgHeader.dstInstId = 1;
                                        txMsgHeader.dstCompId = PRIMITIVE_MANIPULATOR;
                                        txMsgHeader.dstNodeId = mjpsNodeId;
                                        txMsgHeader.dstSubsId = mjpsSubsId ;
                                        txMsgHeader.dataBytes = 0;
                                        jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                }
                                if(mjpsPmState == READY_STATE)
                                        mjpsState = READY_STATE;
                                break;

                        case STANDBY_STATE:
                                break;

                        case READY_STATE:
                                //Check the status of PRIMITIVE MANIPULATOR
                                mjpsQueryPmComponentStatus();

                                if(mjpsPmState == EMERGENCY_STATE)
                                {
                                        cError("mjps: PM in emergency state, switching to
emergency state");
                                        mjpsState = EMERGENCY_STATE;
                                }
                                // Set up the structure returning current sensor data
                                reportedJointPosition.numJoints = NUM_JOINTS;
                                for (i=0; i<NUM_JOINTS; i++)
                                {
                                        reportedJointPosition.jointPosition[i] =
getJointPosition(i);
                                }
                                break;

                        case EMERGENCY_STATE:
                                break;

                        case FAILURE_STATE:
                                break;

                        case SHUTDOWN_STATE:
                                break;

                        default:
                                mjpsState = FAILURE_STATE; // The default case is
undefined, therefore go into Failure State
                                break;
                }
                usleep(1000); // Sleep for one millisecond
        }

        // Sleep for 50 milliseconds and then exit
        usleep(50000);

        mjpsStateThreadRunning = FALSE;

        pthread_exit(NULL);
}

// Function: mjpsNodemgrThread
// Access:            Private
```

```
// Description:       This thread is responsible for handling incoming JAUS messages from
the Node Manager JAUS message service (Jms)
//                           incoming messages are processed according to message type.
void *mjpsNodemgrThread(void *threadData)
{
       jmh_t rxMsgHeader, txMsgHeader; // Declare JAUS header data structures, where the
jmh_t data structure is defined in jaus.h
       unsigned char data[DATASIZE];
       createServiceConnection_t createServiceConnection;
       confirmServiceConnection_t confirmServiceConnection;
       activateServiceConnection_t activateServiceConnection;
       suspendServiceConnection_t suspendServiceConnection;
       terminateServiceConnection_t terminateServiceConnection;
       requestComponentControl_t requestComponentControl;
       confirmComponentControl_t confirmComponentControl;
       componentAuthority_t componentAuthority;
       componentStatus_t componentStatus;
       int recvCount;

       mjpsNodemgrThreadRunning = TRUE; //

       // Setup transmit header
       txMsgHeader.isExpMsg = 0;            // not experimental msg - USER: change to 1
if experimental
       txMsgHeader.isSvcMsg = 0;            // not a service connection
       txMsgHeader.acknCtrl = 0;            // no ack/nak
       txMsgHeader.priority = 6;            // default priority
       txMsgHeader.reserved = 0;
       txMsgHeader.version = 1;             // JAUS 3.0
       txMsgHeader.srcInstId = mjpsInstId;
       txMsgHeader.srcCompId = MANIPULATOR_JOINT_POSITION_SENSOR;
       txMsgHeader.srcNodeId = mjpsNodeId;
       txMsgHeader.srcSubsId = mjpsSubsId;
       txMsgHeader.dataCtrl = 0;
       txMsgHeader.seqNumber = 0;

       while(mjpsRun)  // Execute incoming JAUS message processing while not in the
SHUTDOWN state
       {
              // Check for a new message in the Node Manager JMS queue
              // If a message is present then store the header information in
rxMsgHeader, and store the message content information in the data buffer
              recvCount = jmsRecv(mjpsJms, &rxMsgHeader, data, DATASIZE);
              if(recvCount < 0)
              {
                     cError("mjps: Node manager jmsRecv returned error: %d\n",
recvCount);
                     break;
              }
              if(recvCount == 0) continue;

              // This block of code is intended to reject commands from non-controlling
components
              // However, the one exception allowed is a REQUEST_COMPONENT_CONTROL
              if(mjpsUnderControl && rxMsgHeader.cmdCode < 0x2000) // If not currently
under control or it's not a command, then go to the switch statement
              {
                     // If the source component isn't the one in control, and it's not a
request to gain control, then exit this iteration of the while loop
                     if(    !(     mjpsCtrlCmptInstId==rxMsgHeader.srcInstId &&
                                   mjpsCtrlCmptCompId==rxMsgHeader.srcCompId &&
                                   mjpsCtrlCmptNodeId==rxMsgHeader.srcNodeId &&
                                   mjpsCtrlCmptSubsId==rxMsgHeader.srcSubsId)&&
                            !(     rxMsgHeader.cmdCode==REQUEST_COMPONENT_CONTROL)
                     )continue; // to next iteration
              }

              switch(rxMsgHeader.cmdCode) // Switch the processing algorithm according
to the JAUS message type
              {
```

```
                        // Set the component authority according to the incoming authority
code
                        case SET_COMPONENT_AUTHORITY:
                                convertComponentAuthority(data, &mjpsAuthority, DATASIZE,
UNPACK);       // Unpack and store the incoming authority code
                                break;

                        case SHUTDOWN:
                                mjpsState = SHUTDOWN_STATE;
                                break;

                        case STANDBY:
                                if(mjpsState == READY_STATE)
                                        mjpsState = STANDBY_STATE;
                                break;

                        case RESUME:
                                if(mjpsState == STANDBY_STATE)
                                        mjpsState = READY_STATE;
                                break;

                        case RESET:
                                mjpsState = INITIALIZE_STATE;
                                break;

                        case SET_EMERGENCY:
                                mjpsState = EMERGENCY_STATE;
                                break;

                        case CLEAR_EMERGENCY:
                                mjpsState = STANDBY_STATE;
                                break;

                        case CREATE_SERVICE_CONNECTION:
                                convertCreateServiceConnection(data,
&createServiceConnection, DATASIZE, UNPACK);
                                if(createServiceConnection.cmdCode ==
REPORT_JOINT_POSITION)
                                {
                                        mjpsScHeader = txMsgHeader;
                                        mjpsScHeader.isSvcMsg = 1;
                                        mjpsScHeader.dstInstId = rxMsgHeader.srcInstId;
                                        mjpsScHeader.dstCompId = rxMsgHeader.srcCompId;
                                        mjpsScHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                        mjpsScHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                        mjpsScHeader.cmdCode = REPORT_JOINT_POSITION;
                                        mjpsScHeader.seqNumber = 1;

                                        confirmServiceConnection.cmdCode =
REPORT_JOINT_POSITION;
                                        confirmServiceConnection.instanceId = 1;
                                        confirmServiceConnection.confirmedRate =
createServiceConnection.requestedRate; // BUG: The actual rate may not be the same as the
request
                                        confirmServiceConnection.responseCode =
CONNECTION_SUCCESSFUL;
                                        txMsgHeader.cmdCode = CONFIRM_SERVICE_CONNECTION;
                                        txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                        txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                        txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                        txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                        txMsgHeader.dataBytes =
convertConfirmServiceConnection(data, &confirmServiceConnection, DATASIZE, PACK);
                                        jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);

                                        mjpsScActive = TRUE;
                                }
                                else
                                {
```

```
                                    confirmServiceConnection.cmdCode =
createServiceConnection.cmdCode;
                                    confirmServiceConnection.instanceId = 1;
                                    confirmServiceConnection.confirmedRate =
createServiceConnection.requestedRate;
                                    confirmServiceConnection.responseCode =
CONNECTION_REFUSED;
                                    txMsgHeader.cmdCode = CONFIRM_SERVICE_CONNECTION;
                                    txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                    txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                    txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                    txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                    txMsgHeader.dataBytes =
convertConfirmServiceConnection(data, &confirmServiceConnection, DATASIZE, PACK);
                                    jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                            }
                            break;

                     case CONFIRM_SERVICE_CONNECTION:
                            convertConfirmServiceConnection(data,
&confirmServiceConnection, DATASIZE, UNPACK);
                            // USER: Insert code here to handle the confirm service
connection message
                            break;

                     case ACTIVATE_SERVICE_CONNECTION:
                            convertActivateServiceConnection(data,
&activateServiceConnection, DATASIZE, UNPACK);
                            // USER: Insert code here to handle the activate service
connection message
                            break;

                     case SUSPEND_SERVICE_CONNECTION:
                            convertSuspendServiceConnection(data,
&suspendServiceConnection, DATASIZE, UNPACK);
                            // USER: Insert code here to handle the suspend service
connection message
                            break;

                     case TERMINATE_SERVICE_CONNECTION:
                            convertTerminateServiceConnection(data,
&terminateServiceConnection, DATASIZE, UNPACK);
                            if (terminateServiceConnection.cmdCode ==
REPORT_JOINT_POSITION)
                                   mjpsScActive = FALSE;
                            break;

                     case REQUEST_COMPONENT_CONTROL:
                            convertRequestComponentControl(data,
&requestComponentControl, DATASIZE, UNPACK);

                            if(mjpsUnderControl)
                            {
                                   if(requestComponentControl > mjpsCtrlCmptAuthority)
// Test for higher authority
                                   {
                                          // Terminate control of current component
                                          txMsgHeader.cmdCode    =
REJECT_COMPONENT_CONTROL;
                                          txMsgHeader.dstSubsId = mjpsCtrlCmptSubsId;
                                          txMsgHeader.dstNodeId = mjpsCtrlCmptNodeId;
                                          txMsgHeader.dstCompId = mjpsCtrlCmptCompId;
                                          txMsgHeader.dstInstId = mjpsCtrlCmptInstId;
                                          txMsgHeader.dataBytes = 0;
                                          jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);

                                          // Accept control of new component
                                          txMsgHeader.cmdCode    =
CONFIRM_COMPONENT_CONTROL;
```

```
                                        txMsgHeader.dstSubsId =
rxMsgHeader.srcSubsId;
                                        txMsgHeader.dstNodeId =
rxMsgHeader.srcNodeId;
                                        txMsgHeader.dstCompId =
rxMsgHeader.srcCompId;
                                        txMsgHeader.dstInstId =
rxMsgHeader.srcInstId;
                                        confirmComponentControl.asByte = 0;
                                        txMsgHeader.dataBytes =
convertConfirmComponentControl(data, &confirmComponentControl, DATASIZE, PACK);
                                        jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                        mjpsCtrlCmptInstId = rxMsgHeader.srcInstId;
                                        mjpsCtrlCmptCompId = rxMsgHeader.srcCompId;
                                        mjpsCtrlCmptNodeId = rxMsgHeader.srcNodeId;
                                        mjpsCtrlCmptSubsId = rxMsgHeader.srcSubsId;
                                        mjpsCtrlCmptAuthority =
requestComponentControl;
                                }
                                else
                                {
                                        if(     rxMsgHeader.srcSubsId !=
mjpsCtrlCmptSubsId || rxMsgHeader.srcNodeId != mjpsCtrlCmptNodeId ||
                                                rxMsgHeader.srcCompId !=
mjpsCtrlCmptCompId || rxMsgHeader.srcInstId != mjpsCtrlCmptInstId )
                                        {
                                                txMsgHeader.cmdCode    =
REJECT_COMPONENT_CONTROL;
                                                txMsgHeader.dstSubsId =
rxMsgHeader.srcSubsId;
                                                txMsgHeader.dstNodeId =
rxMsgHeader.srcNodeId;
                                                txMsgHeader.dstCompId =
rxMsgHeader.srcCompId;
                                                txMsgHeader.dstInstId =
rxMsgHeader.srcInstId;
                                                txMsgHeader.dataBytes = 0;
                                                jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                        }
                                }
                        }
                        else // Not currently under component control, so give
control
                        {
                                txMsgHeader.cmdCode    = CONFIRM_COMPONENT_CONTROL;
                                txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                confirmComponentControl.asByte = 0;
                                txMsgHeader.dataBytes =
convertConfirmComponentControl(data, &confirmComponentControl, DATASIZE, PACK);
                                jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                mjpsCtrlCmptInstId = rxMsgHeader.srcInstId;
                                mjpsCtrlCmptCompId = rxMsgHeader.srcCompId;
                                mjpsCtrlCmptNodeId = rxMsgHeader.srcNodeId;
                                mjpsCtrlCmptSubsId = rxMsgHeader.srcSubsId;
                                mjpsCtrlCmptAuthority = requestComponentControl;
                                mjpsUnderControl = TRUE;
                        }
                        break;

                case RELEASE_COMPONENT_CONTROL:
                        mjpsUnderControl = FALSE;
                        break;

                case CONFIRM_COMPONENT_CONTROL:
```

```
                            convertConfirmComponentControl(data,
&confirmComponentControl, DATASIZE, UNPACK);
                            // USER: Insert code here to handle the confirm component
control message if needed
                            break;

                    case REJECT_COMPONENT_CONTROL:
                            // USER: Insert code here to handle the reject component
control message if needed
                            break;

                    case QUERY_COMPONENT_AUTHORITY:
                            txMsgHeader.cmdCode    = REPORT_COMPONENT_AUTHORITY;
                            txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                            txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                            txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                            txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                            componentAuthority = mjpsAuthority;
                            txMsgHeader.dataBytes = convertComponentAuthority(data,
&componentAuthority, DATASIZE, PACK);
                            jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                            break;

                    case QUERY_COMPONENT_STATUS:
                            txMsgHeader.cmdCode    = REPORT_COMPONENT_STATUS;
                            txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                            txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                            txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                            txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                            componentStatus.primaryStatus.asByte = mjpsState;
                            txMsgHeader.dataBytes = convertComponentStatus(data,
&componentStatus, DATASIZE, PACK);
                            jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                            break;

                    case REPORT_COMPONENT_AUTHORITY:
                            convertComponentAuthority(data, &componentAuthority,
DATASIZE, UNPACK);
                            // USER: Insert code here to handle the report component
authority message if needed
                            break;

                    case REPORT_COMPONENT_STATUS:
                            convertComponentStatus(data, &componentStatus, DATASIZE,
UNPACK);
                            if(rxMsgHeader.srcCompId == PRIMITIVE_MANIPULATOR)
                                    mjpsPmState =
componentStatus.primaryStatus.asField.primaryStatusCode;
                            break;

                    case QUERY_JOINT_POSITION:
                        txMsgHeader.cmdCode = REPORT_JOINT_POSITION;
                        txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                        txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                        txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                        txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                        txMsgHeader.dataBytes = convertJointPosition(data,
&reportedJointPosition, DATASIZE, PACK);
                            jmsSend(mjpsJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                            break;

                    default:
                            break;
            }
    }

    mjpsNodemgrThreadRunning = FALSE; //
    pthread_exit(NULL);
```

```
}

////////////////////////////////////////////////////////////////////////////////
// Function used to send a message to the primitive manipulator and query for
// component status. Only the primitive manipulator component performs the initial
// system checks. Thus it is essential that primitive manipulator is running and its
// status is ready before any other components are started up.
////////////////////////////////////////////////////////////////////////////////
void mjpsQueryPmComponentStatus(void)
{
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;

        // Setup Transmit msg
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcInstId = mjpsInstId;
        txMsg.srcCompId = MANIPULATOR_JOINT_POSITION_SENSOR;
        txMsg.srcNodeId = mjpsNodeId;
        txMsg.srcSubsId = mjpsSubsId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = QUERY_COMPONENT_STATUS;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = PRIMITIVE_MANIPULATOR;
        txMsg.dstNodeId = mjpsNodeId;
        txMsg.dstSubsId = mjpsSubsId ;
        txMsg.dataBytes = 0;

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(mjpsJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}


////////////////////////////////////////////////////////////////////////////////
// File:               mjps.h
// Version:            0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:        Tom Galluzzo (galluzzt@ufl.edu)
// Date:               09/28/2004
//
// Description:        This file contains the skeleton C header file code for implementing
//                             the mjps.c file
////////////////////////////////////////////////////////////////////////////////

#ifndef MJPS_H
#define MJPS_H

#include "jaus.h"

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE 1
#endif

#define MJPS_NODE_MANAGER_CHECKIN_ERROR             -1
#define MJPS_JMS_OPEN_ERROR                                 -2
#define MJPS_STARTUP_BEFORE_SHUTDOWN_ERROR   -3
#define MJPS_STATE_THREAD_CREATE_ERROR              -4
#define MJPS_NODEMGR_THREAD_CREATE_ERROR     -5
```

```
#define MJPS_THREAD_TIMEOUT_SEC                          1.0

// Public
int mjpsStartup(void);
int mjpsShutdown(void);
int mjpsGetState(void);
unsigned char mjpsGetInstanceId(void);
unsigned char mjpsGetCompId(void);
unsigned char mjpsGetNodeId(void);
unsigned char mjpsGetSubsystemId(void);
double mjpsGetUpdateRate(void);
int mjpsGetPmState(void);

// Query messages
void mjpsQueryPmComponentStatus(void);

// Private accessors
jointPosition_t * mjpsReportJointPosition(void);

#endif // MJPS_H
```

B.4 The Meepd.c File and the Corresponding Header Meepd.h

```
////////////////////////////////////////////////////////////////////////////////
// File:              meepd.c
// Version:           0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:       Tom Galluzzo (galluzzt@ufl.edu)
// Date:              09/28/2004
//
// Description:       This file contains the C code for implementation of a Manipulator
//                    Joint Move Driver JAUS component in a Linux environment. This code
//                    is designed to work with the node manager and JAUS library software
//                    written by Jeff Witt
////////////////////////////////////////////////////////////////////////////////

#include <jaus.h>                           // JAUS message set (USER: JAUS libraries
must be installed first)
#include <pthread.h>                 // Multi-threading functions (standard to unix)
#include <timeLib.h>                 // Precise timing routines (USER: must link
timeLib.c, written by Tom Galluzzo)
#include <unistd.h>                         // Unix standard functions
#include <stdio.h>                          // Unix standard input-output
#include <math.h>                           // Mathematical functions
#include <nodemgr/nodemgr.h>  // Node managment functions for sending and receiving JAUS
messages (USER: Node Manager must be installed)

#include "meepd.h"    // USER: Implement and rename this header file. Include prototypes
for all public functions contained in this file.
#include "mcConstants.h"
#include "galilInterface.h" // Access to setMaxSpeed() function
#include "cppInterface.h"
#include "logLib.h"

#define DATASIZE JDATASIZE    // Default size for byte stream buffers in this file

// Private function prototypes
void *meepdStateThread(void *);
void *meepdNodemgrThread(void *);

// Query messages requested by this component
void meepdQueryPmComponentStatus(void);
void meepdQueryPmJointEffort(void);
void meepdQueryMjpsJointPosition(void);
void meepdQueryPmManipulatorSpecifications(void);

// Set messages that this component commands
void meepdSetPmJointEffort(void);
```

```
// Accessors to incoming information
toolPoint_t * meepdSetToolPoint(void);
endEffectorPose_t * meepdSetEndEffectorPose(void);
jointEffort_t * meepdReportJointEffort(void);
manipulatorSpecifications_t * meepdReportManipulatorSpecifications(void);
jointPosition_t * meepdReportJointPosition(void);

// Accessors to outgoing information
toolPoint_t * meepdReportToolPoint(void);
jointEffort_t * meepdSetJointEffort(void);

// Accessots to other information
double * meepdGetCurrentOrientation(void);

unsigned char meepdInstId, meepdNodeId, meepdSubsId;        // JAUS Instance, Node and
Subsystem IDs for this component
componentAuthority_t meepdAuthority = 0;                                    // JAUS
Authority for this component
int meepdState = SHUTDOWN_STATE;                                                    // JAUS
State
double meepdThreadHz;                                                               //
Stores the calculated update rate for main state thread
unsigned char meepdCtrlCmptInstId, meepdCtrlCmptCompId;     // Stores identity of the
component currently in control
unsigned char meepdCtrlCmptNodeId, meepdCtrlCmptSubsId;     // Stores identity of the
component currently in control
unsigned char meepdCtrlCmptAuthority;                                       // Stores the
authority level of the component currently in control
int meepdUnderControl = FALSE;                                                      //
FALSE - not under control, TRUE is under control
int meepdRun = FALSE;

int meepdStateThreadRunning = FALSE;
pthread_t meepdStateThreadId;                                               // pthread
node manager thread identifier
int meepdNodemgrThreadRunning = FALSE;
pthread_t meepdNodemgrThreadId;                                                     //
pthread node manager thread identifier

// Structure storing incoming request under case SET TOOL POINT
static toolPoint_t setToolPoint_6;

// Structure storing incoming request under case SET END EFFECTOR POSE
static endEffectorPose_t setEndEffectorPose;

// Structure storing incoming request under case QUERY_TOOL_POINT
static toolPoint_t reportedToolPoint;

// Structures storing queried information
static jointEffort_t reportedJointEffort;
static manipulatorSpecifications_t reportedManipulatorSpecifications;
static jointPosition_t reportedJointPosition;

// Structure storing outgoing request to the PRIMITIVE MANIPULATOR
static jointEffort_t setJointEffort;

static double max_speed[NUM_JOINTS], max_acceleration[NUM_JOINTS],
max_deceleration[NUM_JOINTS];
static double cmd_position[NUM_JOINTS];
static int num_soln;
static double orientS[3], orientA[3];
static double des_position[3], des_orientS[3], des_orientA[3], curr_orientation[4];

jmh_t meepdScHeader;
jms_t meepdJms;       // An accessor to the Node Manager JAUS Message Service (Jms) for
this component
int meepdScActive = FALSE;

int meepdPmState = -1;
unsigned char meepdPmNodeId = 0;
int meepdPmControl = FALSE;
```

```
// Function:   meepdStartup
// Access:            Public
// Description: This function allows the abstracted component functionality contained in
this file to be started from an external source.
//                          It must be called first before the component state machine
and node manager interaction will begin
//                          Each call to "meepdStartup" should be followed by one call
to the "cmptShutdown" function
int meepdStartup(void)
{
       pthread_attr_t attr;  // Thread attributed for the component threads spawned in
this function

       if(meepdState == SHUTDOWN_STATE)     // Execute the startup routines only if the
component is not running
       {
              // Check in to the Node Manager and obtain Instance, Node and Subsystem
IDs
              if(jcmCheckIn(MANIPULATOR_END_EFFECTOR_POSE_DRIVER, &meepdInstId,
&meepdNodeId, &meepdSubsId) < 0) // USER: Insert your component ID define on this line
              {
                           cError("meepd: Could not check in to nodemgr\n");
                           return MEEPD_NODE_MANAGER_CHECKIN_ERROR;
              }

              // Open a conection to the Node Manager and obtain a Jms accessor
              if((meepdJms = jmsOpen(MANIPULATOR_END_EFFECTOR_POSE_DRIVER, meepdInstId,
0)) == NULL) // USER: Insert your component ID define on this line
              {
                     cError("meepd: Could not open connection to nodemgr\n");
                     jcmCheckOut(MANIPULATOR_END_EFFECTOR_POSE_DRIVER, meepdInstId); //
USER: Insert your component ID define on this line
                     return MEEPD_JMS_OPEN_ERROR;
              }

              meepdJms->timeout.tv_usec = 100000; // Timeout for non-blocking jmsRecv,
specified in micro seconds

              pthread_attr_init(&attr);
              pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

              meepdState = INITIALIZE_STATE; // Set the state of the JAUS state machine
to INITIALIZE
              meepdRun = TRUE;

              if(pthread_create(&meepdStateThreadId, &attr, meepdStateThread, NULL) !=
0)
              {
                     cError("meepd: Could not create meepdStateThread\n");
                     meepdShutdown();
                     return MEEPD_STATE_THREAD_CREATE_ERROR;
              }

              if(pthread_create(&meepdNodemgrThreadId, &attr, meepdNodemgrThread, NULL)
!= 0)
              {
                     cError("meepd: Could not create meepdNodemgrThread\n");
                     meepdShutdown();
                     return MEEPD_NODEMGR_THREAD_CREATE_ERROR;
              }

              pthread_attr_destroy(&attr);
       }
       else
       {
              cError("meepd: Attempted startup while not shutdown\n");
              return MEEPD_STARTUP_BEFORE_SHUTDOWN_ERROR;
       }

       return 0;
```

```
}

// Function: meepdShutdown
// Access:            Public
// Description:       This function allows the abstracted component functionality
contained in this file to be stoped from an external source.
//                           If the component is in the running state, this function
will terminate all threads running in this file
//                           This function will also close the Jms connection to the
Node Manager and check out the component from the Node Manager
int meepdShutdown(void)
{
       double timeOutSec;

       if(meepdState != SHUTDOWN_STATE)     // Execute the startup routines only if the
component is not running
       {
              meepdRun = FALSE;

              timeOutSec = getTimeSeconds() + MEEPD_THREAD_TIMEOUT_SEC;
              while(meepdStateThreadRunning)
              {
                     usleep(100000);
                     if(getTimeSeconds() >= timeOutSec)
                     {
                            pthread_cancel(meepdStateThreadId);
                            meepdStateThreadRunning = FALSE;
                            cError("meepd: meepdStateThread Shutdown Improperly\n");
                            break;
                     }
              }

              timeOutSec = getTimeSeconds() + MEEPD_THREAD_TIMEOUT_SEC;
              while(meepdNodemgrThreadRunning)
              {
                     usleep(100000);
                     if(getTimeSeconds() >= timeOutSec)
                     {
                            pthread_cancel(meepdNodemgrThreadId);
                            meepdNodemgrThreadRunning = FALSE;
                            cError("meepd: meepdNodemgrThread Shutdown Improperly\n");
                            break;
                     }
              }

              // Close Node Manager Connection and check out
              jmsClose(meepdJms);
              jcmCheckOut(MANIPULATOR_END_EFFECTOR_POSE_DRIVER, meepdInstId); // USER:
Insert your component ID define on this line

              meepdState = SHUTDOWN_STATE;
       }

       return 0;
}

// The series of functions below allow public access to essential component information
// Access:            Public (All)
int meepdGetState(void){ return meepdState; }
unsigned char meepdGetInstanceId(void){ return meepdInstId; }
unsigned char meepdGetCompId(void){ return (unsigned
char)MANIPULATOR_END_EFFECTOR_POSE_DRIVER; }
unsigned char meepdGetNodeId(void){ return meepdNodeId; }
unsigned char meepdGetSubsystemId(void){ return meepdSubsId; }
double meepdGetUpdateRate(void){ return meepdThreadHz; }
int meepdGetPmState(void){ return meepdPmState; }

// USER: Insert any additional public variable accessors here
toolPoint_t * meepdSetToolPoint(void) {return &setToolPoint_6; }
endEffectorPose_t * meepdSetEndEffectorPose(void) {return &setEndEffectorPose; }
jointEffort_t * meepdReportJointEffort(void) { return &reportedJointEffort; }
```

```
manipulatorSpecifications_t * meepdReportPmManipulatorSpecifications(void) { return
&reportedManipulatorSpecifications; }
jointPosition_t * meepdReportJointPosition(void) { return &reportedJointPosition; }
toolPoint_t * meepdReportToolPoint(void) { return &reportedToolPoint; }
jointEffort_t * meepdSetJointEffort(void) { return &setJointEffort; }

double * meepdGetCurrentOrientation(void) { return curr_orientation; }

// Function: meepdThread
// Access:            Private
// Description:       All core component functionality is contained in this thread.
//                          All of the JAUS component state machine code can be found
here.
void *meepdStateThread(void *threadData)
{
       jmh_t txMsgHeader;
       double time, prevTime;
       unsigned char data[DATASIZE];
       int i;
       double curr_position[NUM_JOINTS], curr_effort[NUM_JOINTS], cmd_effort[NUM_JOINTS],
curr_tool_point[3], tool_point_6[3], cmd_position[3], cmd_orientation[4];
       double spec_a[NUM_JOINTS-1], spec_alpha[NUM_JOINTS-1], spec_S[NUM_JOINTS];
       meepdStateThreadRunning = TRUE; //

       // Setup transmit header
       txMsgHeader.isExpMsg = 0;            // not experimental msg - USER: change to 1
if experimental
       txMsgHeader.isSvcMsg = 0;            // not a service connection
       txMsgHeader.acknCtrl = 0;            // no ack/nak
       txMsgHeader.priority = 6;            // default priority
       txMsgHeader.reserved = 0;
       txMsgHeader.version = 1;             // JAUS 3.0u
       txMsgHeader.srcInstId = meepdInstId;
       txMsgHeader.srcCompId = MANIPULATOR_END_EFFECTOR_POSE_DRIVER;
       txMsgHeader.srcNodeId = meepdNodeId;
       txMsgHeader.srcSubsId = meepdSubsId;
       txMsgHeader.dataCtrl = 0;
       txMsgHeader.seqNumber = 0;

       time = getTimeSeconds();
       meepdState = INITIALIZE_STATE; // Set JAUS state to INITIALIZE

       while(meepdRun) // Execute state machine code while not in the SHUTDOWN state
       {
              prevTime = time;
              time = getTimeSeconds();
              meepdThreadHz = 1.0/(time-prevTime); // Compute the update rate of this
thread

              switch(meepdState) // Switch component behavior based on which state the
machine is in
              {
                     case INITIALIZE_STATE:
                            //Check the status of PRIMITIVE MANIPULATOR
                            meepdQueryPmComponentStatus();
                            if(meepdPmState == EMERGENCY_STATE)
                            {
                                   cError("meepd: PM in emergency state, switching to
emergency state");
                                   meepdState = EMERGENCY_STATE;
                            }
                            if(meepdPmState == STANDBY_STATE)
                            {
                                   txMsgHeader.cmdCode = RESUME;
                                   txMsgHeader.dstInstId = 1;
                                   txMsgHeader.dstCompId = PRIMITIVE_MANIPULATOR;
                                   txMsgHeader.dstNodeId = meepdNodeId;
                                   txMsgHeader.dstSubsId = meepdSubsId ;
                                   txMsgHeader.dataBytes = 0;
                                   jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
```

```
                        }
                        if(meepdPmState == READY_STATE)
                                meepdState = STANDBY_STATE;
                        break;

                case STANDBY_STATE:
                        break;

                case READY_STATE:
                        // Set up default velocity, acceleration, and deceleration
values
                        for (i=0; i<NUM_JOINTS; i++)
                        {
                                if (i == 0){
                                        max_speed[i] = 5000 /
J1_RAD_TO_ENC;//reportedManipulatorSpecifications.jointSpecifications[i].jointMaximumVelo
city / 1000;
                                        max_acceleration[i] = 150000 /
J1_RAD_TO_ENC;
                                }
                                else if (i == 1) {
                                        max_speed[i] = 5000 / J2_RAD_TO_ENC;
                                        max_acceleration[i] = 150000 /
J2_RAD_TO_ENC;
                                }
                                else if (i == 2) {
                                        max_speed[i] = 5000 / J3_RAD_TO_ENC;
                                        max_acceleration[i] = 150000 /
J3_RAD_TO_ENC;
                                }
                                else if (i == 3) {
                                        max_speed[i] = 1000 / J4_RAD_TO_ENC;
                                        max_acceleration[i] = 150000 /
J4_RAD_TO_ENC;
                                }
                                else if (i == 4) {
                                        max_speed[i] = 1000 / J5_RAD_TO_ENC;
                                        max_acceleration[i] = 150000 /
J5_RAD_TO_ENC;
                                }
                                else {
                                        max_speed[i] = 1000 /
J6_RAD_TO_ENC;//reportedManipulatorSpecifications.jointNmaximumVelocity / 1000;
                                        max_acceleration[i] = 150000 /
J6_RAD_TO_ENC;
                                }
                                max_deceleration[i] = max_acceleration[i];
                        }
                        setMaxSpeed(max_speed);
                        setMaxAcceleration(max_acceleration);
                        setMaxDeceleration(max_deceleration);

                        //Check the status of PRIMITIVE MANIPULATOR
                        meepdQueryPmComponentStatus();
                        if(meepdPmState == EMERGENCY_STATE)
                        {
                                cError("meepd: PM in emergency state, switching to
emergency state");
                                meepdState = EMERGENCY_STATE;
                        }

                        // Query for required information
                        meepdQueryMjpsJointPosition();
                        meepdQueryPmManipulatorSpecifications();
                        meepdQueryPmJointEffort();

                        for (i=0; i<NUM_JOINTS; i++)
                        {
                                curr_position[i] =
reportedJointPosition.jointPosition[i];
                                curr_effort[i] = reportedJointEffort.jointEffort[i];
```

```
                                }

                                for(i=0; i<NUM_JOINTS-1; i++)
                                {
                                        spec_a[i] =
reportedManipulatorSpecifications.jointSpecifications[i].linkLength;
                                        spec_alpha[i] =
reportedManipulatorSpecifications.jointSpecifications[i].twistAngle/1000;
                                        if (i<3 || i==4)
                                                spec_S[i] =
reportedManipulatorSpecifications.jointSpecifications[i].jointOffsetOrAngle;
                                        else if (i==3)
                                                spec_S[i] = -
(reportedManipulatorSpecifications.jointSpecifications[i].jointOffsetOrAngle);
                                }
                                spec_S[5] = -
(reportedManipulatorSpecifications.jointNoffsetOrAngle);

                                // Extract incoming tool point in 6th coordinate
                                tool_point_6[0] = setToolPoint_6.X * 1000; // mm
                                tool_point_6[1] = setToolPoint_6.Y * 1000;
                                tool_point_6[2] = setToolPoint_6.Z * 1000;

                                cDebug(9, "Set Tool Point %lf\n", tool_point_6[0] );
                                cDebug(9, "Set Tool Point %lf\n", tool_point_6[1] );
                                cDebug(9, "Set Tool Point %lf\n", tool_point_6[2] );
                                cDebug(9, "\n");

                                // This function returns the current position and
orientation of the end-effector
                                getCurrentEndEffectorPose(spec_a, spec_alpha, spec_S,
tool_point_6, curr_position,
                                                                        curr_tool_point,
orientS, orientA);

                                matrixToQuaternion(orientA, orientS, curr_orientation);

                                // Report Tool Point
                                reportedToolPoint.X = curr_tool_point[0];
                                reportedToolPoint.Y = curr_tool_point[1];
                                reportedToolPoint.Z = curr_tool_point[2];

/*
                                cDebug(9, "Reported Tool Poisiton %lf\n",
reportedToolPoint.X );
                                cDebug(9, "Reported Tool Poisiton %lf\n",
reportedToolPoint.Y );
                                cDebug(9, "Reported Tool Poisiton %lf\n",
reportedToolPoint.Z );
                                cDebug(9, "\n");

                                cDebug(9, "Reported Orientation S %lf\n", orientS[0]);
                                cDebug(9, "Reported Orientation S %lf\n", orientS[1]);
                                cDebug(9, "Reported Orientation S %lf\n", orientS[2]);
                                cDebug(9, "\n");

                                cDebug(9, "Reported Orientation A %lf\n", orientA[0]);
                                cDebug(9, "Reported Orientation A %lf\n", orientA[1]);
                                cDebug(9, "Reported Orientation A %lf\n", orientA[2]);
                                cDebug(9, "\n");
*/
/*                              des_position[0] = 849;
                                des_position[1] = 473;
                                des_position[2] = 879;

                                des_orientS[0] = -0.674398;
                                des_orientS[1] = -0.729044;
                                des_orientS[2] = -0.116967;

                                des_orientA[0] = 0.732137;
                                des_orientA[1] = -0.680800;
```

```
                              des_orientA[2] = 0.022070;
*/
/*                            des_position[0] = 908;
                              des_position[1] = -602;
                              des_position[2] = 512;

                              des_orientS[0] = -0.6229841;
                              des_orientS[1] = 0.68939257;
                              des_orientS[2] = -0.3696331;

                              des_orientA[0] = -0.4888207;
                              des_orientA[1] = -0.02581027;
                              des_orientA[2] = 0.87200234;

                              matrixToQuaternion(des_orientA, des_orientS,
cmd_orientation);
*/                            // Extract information about desired position and
orientation
                              des_position[0] = setEndEffectorPose.X * 1000; // mm
                              des_position[1] = setEndEffectorPose.Y * 1000;
                              des_position[2] = setEndEffectorPose.Z * 1000;
                              cmd_orientation[0] =
setEndEffectorPose.quaternionQcomponentD;
                              cmd_orientation[1] =
setEndEffectorPose.quaternionQcomponentA;
                              cmd_orientation[2] =
setEndEffectorPose.quaternionQcomponentB;
                              cmd_orientation[3] =
setEndEffectorPose.quaternionQcomponentC;

                              cDebug(9, "Set Tool Point %lf\n", des_position[0] );
                              cDebug(9, "Set Tool Point %lf\n", des_position[1] );
                              cDebug(9, "Set Tool Point %lf\n", des_position[2] );
                              cDebug(9, "\n");

                              quaternionToMatrix(cmd_orientation, des_orientS,
des_orientA);

                              // This function returns the joint angles corresponding to
the desired pose
                              getDesiredJointAngles(des_position, des_orientS,
des_orientA, cmd_position);

                              cDebug(9, "Cmd Positions are: %lf\n",
cmd_position[0]*J1_RAD_TO_ENC);
                              cDebug(9, "Cmd Positions are: %lf\n",
cmd_position[1]*J2_RAD_TO_ENC);
                              cDebug(9, "Cmd Positions are: %lf\n",
cmd_position[2]*J3_RAD_TO_ENC);
                              cDebug(9, "Cmd Positions are: %lf\n",
cmd_position[3]*J4_RAD_TO_ENC);
                              cDebug(9, "Cmd Positions are: %lf\n",
cmd_position[4]*J5_RAD_TO_ENC);
                              cDebug(9, "Cmd Positions are: %lf\n",
cmd_position[5]*J6_RAD_TO_ENC);

                              // This function performs closed loop end effector pose
control
                              endEffectorPoseDriver(cmd_position, curr_position,
cmd_effort);

                              // Set Joint Efforts
                              setJointEffort.numJoints = NUM_JOINTS;
                              for (i=0; i<NUM_JOINTS; i++)
                              {
                                     setJointEffort.jointEffort[i] = cmd_effort[i];
                              }
                              meepdSetPmJointEffort();

                              break;
```

```
                        case EMERGENCY_STATE:
                                break;

                        case FAILURE_STATE:
                                break;

                        case SHUTDOWN_STATE:
                                break;

                        default:
                                meepdState = FAILURE_STATE; // The default case is
undefined, therefore go into Failure State
                                break;
                }
                usleep(1000); // Sleep for one millisecond
        }

        // Sleep for 50 milliseconds and then exit
        usleep(50000);

        meepdStateThreadRunning = FALSE;

        pthread_exit(NULL);
}

// Function: meepdNodemgrThread
// Access:             Private
// Description:        This thread is responsible for handling incoming JAUS messages from
the Node Manager JAUS message service (Jms)
//                             incoming messages are processed according to message type.
void *meepdNodemgrThread(void *threadData)
{
        jmh_t rxMsgHeader, txMsgHeader; // Declare JAUS header data structures, where the
jmh_t data structure is defined in jaus.h
        unsigned char data[DATASIZE];
        createServiceConnection_t createServiceConnection;
        confirmServiceConnection_t confirmServiceConnection;
        activateServiceConnection_t activateServiceConnection;
        suspendServiceConnection_t suspendServiceConnection;
        terminateServiceConnection_t terminateServiceConnection;
        requestComponentControl_t requestComponentControl;
        confirmComponentControl_t confirmComponentControl;
        componentAuthority_t componentAuthority;
        componentStatus_t componentStatus;
        int recvCount;

        meepdNodemgrThreadRunning = TRUE; //

        // Setup transmit header
        txMsgHeader.isExpMsg = 0;            // not experimental msg - USER: change to 1
if experimental
        txMsgHeader.isSvcMsg = 0;            // not a service connection
        txMsgHeader.acknCtrl = 0;            // no ack/nak
        txMsgHeader.priority = 6;            // default priority
        txMsgHeader.reserved = 0;
        txMsgHeader.version = 1;             // JAUS 3.0
        txMsgHeader.srcInstId = meepdInstId;
        txMsgHeader.srcCompId = MANIPULATOR_END_EFFECTOR_POSE_DRIVER;
        txMsgHeader.srcNodeId = meepdNodeId;
        txMsgHeader.srcSubsId = meepdSubsId;
        txMsgHeader.dataCtrl = 0;
        txMsgHeader.seqNumber = 0;

        while(meepdRun)  // Execute incoming JAUS message processing while not in the
SHUTDOWN state
        {
                // Check for a new message in the Node Manager JMS queue
                // If a message is present then store the header information in
rxMsgHeader, and store the message content information in the data buffer
                recvCount = jmsRecv(meepdJms, &rxMsgHeader, data, DATASIZE);
                if(recvCount < 0)
```

```
		{
			cError("meepd: Node manager jmsRecv returned error: %d\n",
recvCount);
			break;
		}
		if(recvCount == 0) continue;

		// This block of code is intended to reject commands from non-controlling
components
		// However, the one exception allowed is a REQUEST_COMPONENT_CONTROL
		if(meepdUnderControl && rxMsgHeader.cmdCode < 0x2000) // If not currently
under control or it's not a command, then go to the switch statement
		{
			// If the source component isn't the one in control, and it's not a
request to gain control, then exit this iteration of the while loop
			if(	!(	meepdCtrlCmptInstId==rxMsgHeader.srcInstId &&
					meepdCtrlCmptCompId==rxMsgHeader.srcCompId &&
					meepdCtrlCmptNodeId==rxMsgHeader.srcNodeId &&
					meepdCtrlCmptSubsId==rxMsgHeader.srcSubsId)&&
				!(	rxMsgHeader.cmdCode==REQUEST_COMPONENT_CONTROL)
			)continue; // to next iteration
		}

		switch(rxMsgHeader.cmdCode) // Switch the processing algorithm according
to the JAUS message type
		{
			// Set the component authority according to the incoming authority
code
			case SET_COMPONENT_AUTHORITY:
				convertComponentAuthority(data, &meepdAuthority, DATASIZE,
UNPACK);	// Unpack and store the incoming authority code
				break;

			case SHUTDOWN:
				meepdState = SHUTDOWN_STATE;
				break;

			case STANDBY:
				if(meepdState == READY_STATE)
					meepdState = STANDBY_STATE;
				break;

			case RESUME:
				if(meepdState == STANDBY_STATE)
					meepdState = READY_STATE;
				break;

			case RESET:
				meepdState = INITIALIZE_STATE;
				break;

			case SET_EMERGENCY:
				meepdState = EMERGENCY_STATE;
				break;

			case CLEAR_EMERGENCY:
				meepdState = STANDBY_STATE;
				break;

			case CREATE_SERVICE_CONNECTION:
				convertCreateServiceConnection(data,
&createServiceConnection, DATASIZE, UNPACK);
				if(createServiceConnection.cmdCode ==
REPORT_JOINT_VELOCITY)
				{
					meepdScHeader = txMsgHeader;
					meepdScHeader.isSvcMsg = 1;
					meepdScHeader.dstInstId = rxMsgHeader.srcInstId;
					meepdScHeader.dstCompId = rxMsgHeader.srcCompId;
					meepdScHeader.dstNodeId = rxMsgHeader.srcNodeId;
					meepdScHeader.dstSubsId = rxMsgHeader.srcSubsId;
```

```
                                    meepdScHeader.cmdCode = REPORT_JOINT_VELOCITY;
                                    meepdScHeader.seqNumber = 1;

                                    confirmServiceConnection.cmdCode =
REPORT_JOINT_VELOCITY;
                                    confirmServiceConnection.instanceId = 1;
                                    confirmServiceConnection.confirmedRate =
createServiceConnection.requestedRate; // BUG: The actual rate may not be the same as the
request
                                    confirmServiceConnection.responseCode =
CONNECTION_SUCCESSFUL;
                                    txMsgHeader.cmdCode = CONFIRM_SERVICE_CONNECTION;
                                    txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                    txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                    txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                    txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                    txMsgHeader.dataBytes =
convertConfirmServiceConnection(data, &confirmServiceConnection, DATASIZE, PACK);
                                    jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);

                                    meepdScActive = TRUE;
                            }
                            else
                            {
                                    confirmServiceConnection.cmdCode =
createServiceConnection.cmdCode;
                                    confirmServiceConnection.instanceId = 1;
                                    confirmServiceConnection.confirmedRate =
createServiceConnection.requestedRate;
                                    confirmServiceConnection.responseCode =
CONNECTION_REFUSED;
                                    txMsgHeader.cmdCode = CONFIRM_SERVICE_CONNECTION;
                                    txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                    txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                    txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                    txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                    txMsgHeader.dataBytes =
convertConfirmServiceConnection(data, &confirmServiceConnection, DATASIZE, PACK);
                                    jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                            }
                            break;

                    case CONFIRM_SERVICE_CONNECTION:
                            convertConfirmServiceConnection(data,
&confirmServiceConnection, DATASIZE, UNPACK);
                            // USER: Insert code here to handle the confirm service
connection message
                            break;

                    case ACTIVATE_SERVICE_CONNECTION:
                            convertActivateServiceConnection(data,
&activateServiceConnection, DATASIZE, UNPACK);
                            // USER: Insert code here to handle the activate service
connection message
                            break;

                    case SUSPEND_SERVICE_CONNECTION:
                            convertSuspendServiceConnection(data,
&suspendServiceConnection, DATASIZE, UNPACK);
                            // USER: Insert code here to handle the suspend service
connection message
                            break;

                    case TERMINATE_SERVICE_CONNECTION:
                            convertTerminateServiceConnection(data,
&terminateServiceConnection, DATASIZE, UNPACK);
                            if (terminateServiceConnection.cmdCode ==
REPORT_JOINT_VELOCITY)
                                    meepdScActive = FALSE;
```

```
                        break;

                    case REQUEST_COMPONENT_CONTROL:
                        convertRequestComponentControl(data,
&requestComponentControl, DATASIZE, UNPACK);

                        if(meepdUnderControl)
                        {
                            if(requestComponentControl > meepdCtrlCmptAuthority)
// Test for higher authority
                            {
                                // Terminate control of current component
                                txMsgHeader.cmdCode    =
REJECT_COMPONENT_CONTROL;
                                txMsgHeader.dstSubsId = meepdCtrlCmptSubsId;
                                txMsgHeader.dstNodeId = meepdCtrlCmptNodeId;
                                txMsgHeader.dstCompId = meepdCtrlCmptCompId;
                                txMsgHeader.dstInstId = meepdCtrlCmptInstId;
                                txMsgHeader.dataBytes = 0;
                                jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);

                                // Accept control of new component
                                txMsgHeader.cmdCode    =
CONFIRM_COMPONENT_CONTROL;
                                txMsgHeader.dstSubsId =
rxMsgHeader.srcSubsId;
                                txMsgHeader.dstNodeId =
rxMsgHeader.srcNodeId;
                                txMsgHeader.dstCompId =
rxMsgHeader.srcCompId;
                                txMsgHeader.dstInstId =
rxMsgHeader.srcInstId;
                                confirmComponentControl.asByte = 0;
                                txMsgHeader.dataBytes =
convertConfirmComponentControl(data, &confirmComponentControl, DATASIZE, PACK);
                                jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                meepdCtrlCmptInstId = rxMsgHeader.srcInstId;
                                meepdCtrlCmptCompId = rxMsgHeader.srcCompId;
                                meepdCtrlCmptNodeId = rxMsgHeader.srcNodeId;
                                meepdCtrlCmptSubsId = rxMsgHeader.srcSubsId;
                                meepdCtrlCmptAuthority =
requestComponentControl;
                            }
                            else
                            {
                                if(     rxMsgHeader.srcSubsId !=
meepdCtrlCmptSubsId || rxMsgHeader.srcNodeId != meepdCtrlCmptNodeId ||
                                        rxMsgHeader.srcCompId !=
meepdCtrlCmptCompId || rxMsgHeader.srcInstId != meepdCtrlCmptInstId )
                                {
                                    txMsgHeader.cmdCode    =
REJECT_COMPONENT_CONTROL;
                                    txMsgHeader.dstSubsId =
rxMsgHeader.srcSubsId;
                                    txMsgHeader.dstNodeId =
rxMsgHeader.srcNodeId;
                                    txMsgHeader.dstCompId =
rxMsgHeader.srcCompId;
                                    txMsgHeader.dstInstId =
rxMsgHeader.srcInstId;
                                    txMsgHeader.dataBytes = 0;
                                    jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                }
                            }
                        }
                        else // Not currently under component control, so give
control
                        {
```

```
                                    txMsgHeader.cmdCode    = CONFIRM_COMPONENT_CONTROL;
                                    txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                    txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                    txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                    txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                    confirmComponentControl.asByte = 0;
                                    txMsgHeader.dataBytes =
convertConfirmComponentControl(data, &confirmComponentControl, DATASIZE, PACK);
                                    jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                    meepdCtrlCmptInstId = rxMsgHeader.srcInstId;
                                    meepdCtrlCmptCompId = rxMsgHeader.srcCompId;
                                    meepdCtrlCmptNodeId = rxMsgHeader.srcNodeId;
                                    meepdCtrlCmptSubsId = rxMsgHeader.srcSubsId;
                                    meepdCtrlCmptAuthority = requestComponentControl;
                                    meepdUnderControl = TRUE;
                            }
                            break;

                    case RELEASE_COMPONENT_CONTROL:
                            meepdUnderControl = FALSE;
                            break;

                    case CONFIRM_COMPONENT_CONTROL:
                            convertConfirmComponentControl(data,
&confirmComponentControl, DATASIZE, UNPACK);
                            // USER: Insert code here to handle the confirm component
control message if needed
                            break;

                    case REJECT_COMPONENT_CONTROL:
                            // USER: Insert code here to handle the reject component
control message if needed
                            break;

                    case QUERY_COMPONENT_AUTHORITY:
                            txMsgHeader.cmdCode    = REPORT_COMPONENT_AUTHORITY;
                            txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                            txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                            txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                            txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                            componentAuthority = meepdAuthority;
                            txMsgHeader.dataBytes = convertComponentAuthority(data,
&componentAuthority, DATASIZE, PACK);
                            jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                            break;

                    case QUERY_COMPONENT_STATUS:
                            txMsgHeader.cmdCode    = REPORT_COMPONENT_STATUS;
                            txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                            txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                            txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                            txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                            componentStatus.primaryStatus.asByte = meepdState;
                            txMsgHeader.dataBytes = convertComponentStatus(data,
&componentStatus, DATASIZE, PACK);
                            jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                            break;

                    case REPORT_COMPONENT_AUTHORITY:
                            convertComponentAuthority(data, &componentAuthority,
DATASIZE, UNPACK);
                            // USER: Insert code here to handle the report component
authority message if needed
                            break;

                    case REPORT_COMPONENT_STATUS:
                            convertComponentStatus(data, &componentStatus, DATASIZE,
UNPACK);
```

```
                                if(rxMsgHeader.srcCompId == PRIMITIVE_MANIPULATOR)
                                        meepdPmState =
componentStatus.primaryStatus.asField.primaryStatusCode;
                                break;

                        case SET_TOOL_POINT:
                                convertToolPoint(data, &setToolPoint_6, DATASIZE, UNPACK);
                                break;

                        case SET_END_EFFECTOR_POSE:
                                convertEndEffectorPose(data, &setEndEffectorPose, DATASIZE,
UNPACK);
                                break;

                        case REPORT_JOINT_EFFORT:
                                convertJointEffort(data, &reportedJointEffort, DATASIZE,
UNPACK);
                                break;

                        case REPORT_MANIPULATOR_SPECIFICATIONS:
                                convertManipulatorSpecifications(data,
&reportedManipulatorSpecifications, DATASIZE, UNPACK);
                                break;

                        case REPORT_JOINT_POSITION:
                                convertJointPosition(data, &reportedJointPosition,
DATASIZE, UNPACK);
                                break;

                        case QUERY_TOOL_POINT:
                            txMsgHeader.cmdCode = REPORT_TOOL_POINT;
                            txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                            txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                            txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                            txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                            txMsgHeader.dataBytes = convertToolPoint(data,
&reportedToolPoint, DATASIZE, PACK);
                                jmsSend(meepdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                break;

                        default:
                                break;
                }
        }

        meepdNodemgrThreadRunning = FALSE;
        pthread_exit(NULL);
}

////////////////////////////////////////////////////////////////////////////
// Function used to send a message to the primitive manipulator and query for
// component status. Only the primitive manipulator component performs the initial
// system checks. Thus it is essential that primitive manipulator is running and its
// status is ready before any other components are started up.
////////////////////////////////////////////////////////////////////////////
void meepdQueryPmComponentStatus(void)
{
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;

        // Setup Transmit msg
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcInstId = meepdInstId;
```

```
        txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_POSE_DRIVER;
        txMsg.srcNodeId = meepdNodeId;
        txMsg.srcSubsId = meepdSubsId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = QUERY_COMPONENT_STATUS;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = PRIMITIVE_MANIPULATOR;
        txMsg.dstNodeId = meepdNodeId;
        txMsg.dstSubsId = meepdSubsId ;
        txMsg.dataBytes = 0;

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(meepdJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}

////////////////////////////////////////////////////////////////////////////////
// Function used to set up a message sent to manipulator joint position sensor
// to obtain current joint position data.
////////////////////////////////////////////////////////////////////////////////
void meepdQueryMjpsJointPosition(void)
{
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;

        // Setup Transmit msg
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcSubsId = meepdSubsId;
        txMsg.srcNodeId = meepdNodeId;
        txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_POSE_DRIVER;
        txMsg.srcInstId = meepdInstId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = QUERY_JOINT_POSITION;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = MANIPULATOR_JOINT_POSITION_SENSOR;
        txMsg.dstNodeId = meepdNodeId;
        txMsg.dstSubsId = meepdSubsId ;
        txMsg.dataBytes = 0;

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(meepdJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}

////////////////////////////////////////////////////////////////////////////////
// Function used to set up a message sent to the primitive manipulator to obtain
// current effort values.
////////////////////////////////////////////////////////////////////////////////
void meepdQueryPmJointEffort(void)
{
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;

        // Setup Transmit msg
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
```

```
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcSubsId = meepdSubsId;
        txMsg.srcNodeId = meepdNodeId;
        txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_POSE_DRIVER;
        txMsg.srcInstId = meepdInstId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = QUERY_JOINT_EFFORT;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = PRIMITIVE_MANIPULATOR;
        txMsg.dstNodeId = meepdNodeId;
        txMsg.dstSubsId = meepdSubsId ;
        txMsg.dataBytes = 0;

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(meepdJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}

////////////////////////////////////////////////////////////////////////////////
// Function used to set up a message sent to the primitive manipulator to obtain
// manipulator specifications.
////////////////////////////////////////////////////////////////////////////////
void meepdQueryPmManipulatorSpecifications(void)
{
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;

        // Setup Transmit msg
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcSubsId = meepdSubsId;
        txMsg.srcNodeId = meepdNodeId;
        txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_POSE_DRIVER;
        txMsg.srcInstId = meepdInstId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = QUERY_MANIPULATOR_SPECIFICATIONS;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = PRIMITIVE_MANIPULATOR;
        txMsg.dstNodeId = meepdNodeId;
        txMsg.dstSubsId = meepdSubsId ;
        txMsg.dataBytes = 0;

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(meepdJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}

////////////////////////////////////////////////////////////////////////////////
// Function used to set up a message sent to the primitive manipulator with new
// joint effort values.
////////////////////////////////////////////////////////////////////////////////
void meepdSetPmJointEffort(void)
{
        jmh_t txMsg;
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;
```

```
	unsigned char data[DATASIZE];

	//Setup Transmit Message
	txMsg.isExpMsg = 0;
	txMsg.priority = 6;
	txMsg.acknCtrl = 0;
	txMsg.isSvcMsg = 0;
	txMsg.version = 1;
	txMsg.reserved = 0;
	txMsg.srcInstId = meepdInstId;
	txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_POSE_DRIVER;
	txMsg.srcNodeId = meepdNodeId;
	txMsg.srcSubsId = meepdSubsId;
	txMsg.dataCtrl = 0;
	txMsg.seqNumber = 0;
	txMsg.cmdCode = SET_JOINT_EFFORT;
	txMsg.dstInstId = 1;
	txMsg.dstCompId = PRIMITIVE_MANIPULATOR;
	txMsg.dstNodeId = meepdNodeId;
	txMsg.dstSubsId = meepdSubsId;
	txMsg.dataBytes = convertJointEffort(data, &setJointEffort, DATASIZE, PACK);

	if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
	{
		jmsSend(meepdJms, &txMsg, data, txMsg.dataBytes);
		lastRequestTimeSec = getTimeSeconds();
	}
}

////////////////////////////////////////////////////////////////////////////
// Function used to perform closed loop end-effector position control
////////////////////////////////////////////////////////////////////////////
void endEffectorPoseDriver(double *cmd_position, double *curr_position, double
*cmd_effort)
{
	int i;
	double k[NUM_JOINTS];

	for (i=0; i<NUM_JOINTS; i++)
	{
		if (i < 3) k[i] = 34;
		else if (i == 3) k[i] = 34;
		else if (i == 4) k[i] = 15;
		else k[i] = 14;

		if (i == 0)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J1_RAD_TO_ENC * 100 / 112000;
		else if (i == 1)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J2_RAD_TO_ENC * 100 / 112000;
		else if (i == 2)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J3_RAD_TO_ENC * 100 / 112000;
		else if (i == 3)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J4_RAD_TO_ENC * 100 / 35000;
		else if (i == 4)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J5_RAD_TO_ENC * 100 / 16000;
		else
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J6_RAD_TO_ENC * 100 / 15500;
	}
}

void matrixToQuaternion(double *orientA, double *orientS, double *curr_orientation)
{
	double T, S;
	double W, X, Y, Z;
	double M0, M1, M2;
```

```
        double M4, M5, M6;
        double M8, M9, M10;

        M0 = orientA[0];
        M4 = orientA[1];
        M8 = orientA[2];

        M1 = orientS[1]*orientA[2] - orientS[2]*orientA[1];
        M5 = orientS[2]*orientA[0] - orientS[0]*orientA[2];
        M9 = orientS[0]*orientA[1] - orientS[1]*orientA[0];

        M2 = orientS[0];
        M6 = orientS[1];
        M10 = orientS[2];

        T = M0 + M5 + M10 + 1;
        S = 0.5/sqrt(T);

        W = 0.25/S;
        X = (M9 - M6)*S;
        Y = (M2 - M8)*S;
        Z = (M4 - M1)*S;

        curr_orientation[0] = W;
        curr_orientation[1] = X;
        curr_orientation[2] = Y;
        curr_orientation[3] = Z;
}

void quaternionToMatrix(double *cmd_orientation,double *des_orientS, double *des_orientA)
{
        double W, X, Y, Z;
        double M0, M1, M2;
        double M4, M5, M6;
        double M8, M9, M10;

        W = cmd_orientation[0];
        X = cmd_orientation[1];
        Y = cmd_orientation[2];
        Z = cmd_orientation[3];

        M0 = 1 - 2*Y*Y - 2*Z*Z;
        M4 = 2*X*Y + 2*Z*W;
        M8 = 2*X*Z - 2*Y*W;

        M1 = 2*X*Y - 2*Z*W;
        M5 = 1 - 2*X*X - 2*Z*Z;
        M9 = 2*Y*Z + 2*X*W;

        M2 = 2*X*Z + 2*Y*W;
        M6 = 2*Y*Z - 2*X*W;
        M10 = 1 - 2*X*X - 2*Y*Y;

        des_orientA[0] = M0;
        des_orientA[1] = M4;
        des_orientA[2] = M8;

        des_orientS[0] = M2;
        des_orientS[1] = M6;
        des_orientS[2] = M10;

}



////////////////////////////////////////////////////////////////////////////////
// File:              meepd.h
// Version:           0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:       Tom Galluzzo (galluzzt@ufl.edu)
```

```
// Date:              09/28/2004
//
// Description:       This file contains the skeleton C header file code for implementing
//                           the meepd.c file
////////////////////////////////////////////////////////////////////////////////////////

#ifndef MEEPD_H
#define MEEPD_H

#include "jaus.h"

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE 1
#endif

#define MEEPD_NODE_MANAGER_CHECKIN_ERROR            -1
#define MEEPD_JMS_OPEN_ERROR                                -2
#define MEEPD_STARTUP_BEFORE_SHUTDOWN_ERROR  -3
#define MEEPD_STATE_THREAD_CREATE_ERROR             -4
#define MEEPD_NODEMGR_THREAD_CREATE_ERROR    -5

#define MEEPD_THREAD_TIMEOUT_SEC                          1.0

// Public
int meepdStartup(void);
int meepdShutdown(void);
int meepdGetState(void);
unsigned char meepdGetInstanceId(void);
unsigned char meepdGetCompId(void);
unsigned char meepdGetNodeId(void);
unsigned char meepdGetSubsystemId(void);
double meepdGetUpdateRate(void);
int meepdGetPmState(void);

// Query messages requested by this component
void meepdQueryPmComponentStatus(void);
void meepdQueryPmJointEffort(void);
void meepdQueryMjpsJointPosition(void);
void meepdQueryPmManipulatorSpecifications(void);

// Set messages that this component commands
void meepdSetPmJointEffort(void);

// Accessors to incoming information
toolPoint_t * meepdSetToolPoint(void);
endEffectorPose_t * meepdSetEndEffectorPose(void);
jointEffort_t * meepdReportJointEffort(void);
manipulatorSpecifications_t * meepdReportManipulatorSpecifications(void);
jointPosition_t * meepdReportJointPosition(void);

// Accessors to outgoing information
toolPoint_t * meepdReportToolPoint(void);
jointEffort_t * meepdSetJointEffort(void);

double * meepdGetCurrentOrientation(void);

void endEffectorPoseDriver(double *, double *, double *);
void matrixToQuaternion(double *, double *, double *);
void quaternionToMatrix(double *, double *, double *);

#endif // MEEPD_H
```

B.5 The Meedpd.c File and the Corresponding Header Meedpd.h

```
////////////////////////////////////////////////////////////////////////////////////////
// File:              meedpd.c
```

```
// Version:           0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:       Tom Galluzzo (galluzzt@ufl.edu)
// Date:              09/28/2004
//
// Description:       This file contains the C code for implementation of a Manipulator
//                    End-Effector Discrete Pose Driver JAUS component in a Linux
//                    environment. This code is designed to work with the node manager
//                    and JAUS library software written by Jeff Witt
////////////////////////////////////////////////////////////////////////////////

#include <jaus.h>                           // JAUS message set (USER: JAUS libraries
must be installed first)
#include <pthread.h>                 // Multi-threading functions (standard to unix)
#include <timeLib.h>                 // Precise timing routines (USER: must link
timeLib.c, written by Tom Galluzzo)
#include <unistd.h>                         // Unix standard functions
#include <stdio.h>                          // Unix standard input-output
#include <math.h>                           // Mathematical functions
#include <nodemgr/nodemgr.h>  // Node managment functions for sending and receiving JAUS
messages (USER: Node Manager must be installed)

#include "meedpd.h"    // USER: Implement and rename this header file. Include prototypes
for all public functions contained in this file.
#include "mcConstants.h"
#include "galilInterface.h" // Access to setMaxSpeed() function
#include "cppInterface.h"
#include "logLib.h"

#define DATASIZE JDATASIZE    // Default size for byte stream buffers in this file

// Private function prototypes
void *meedpdStateThread(void *);
void *meedpdNodemgrThread(void *);

// Query messages requested by this component
void meedpdQueryPmComponentStatus(void);
void meedpdQueryPmJointEffort(void);
void meedpdQueryMjpsJointPosition(void);
void meedpdQueryPmManipulatorSpecifications(void);

// Set messages that this component commands
void meedpdSetPmJointEffort(void);

// Accessors to incoming information
toolPoint_t * meedpdSetToolPoint(void);
endEffectorPathMotion_t * meedpdSetEndEffectorPathMotion(void);
jointEffort_t * meedpdReportJointEffort(void);
manipulatorSpecifications_t * meedpdReportManipulatorSpecifications(void);
jointPosition_t * meedpdReportJointPosition(void);

// Accessors to outgoing information
toolPoint_t * meedpdReportToolPoint(void);
jointEffort_t * meedpdSetJointEffort(void);

// Accessots to other information
double * meedpdGetCurrentOrientation(void);
int meedpdGetPose(void);
double meedpdGetPoseTime(void);
double meedpdGetCurrentTime(void);
double * meedpdGetCurrentEndEffectorPosition(void);
double * meedpdGetCurrentEndEffectorOrientation(void);

unsigned char meedpdInstId, meedpdNodeId, meedpdSubsId;            // JAUS Instance, Node
and Subsystem IDs for this component
componentAuthority_t meedpdAuthority = 0;                                  // JAUS
Authority for this component
int meedpdState = SHUTDOWN_STATE;                                                 // JAUS
State
double meedpdThreadHz;                                                            //
Stores the calculated update rate for main state thread
```

```
unsigned char meedpdCtrlCmptInstId, meedpdCtrlCmptCompId;   // Stores identity of the
component currently in control
unsigned char meedpdCtrlCmptNodeId, meedpdCtrlCmptSubsId;   // Stores identity of the
component currently in control
unsigned char meedpdCtrlCmptAuthority;                                    // Stores the
authority level of the component currently in control
int meedpdUnderControl = FALSE;                                                  //
FALSE - not under control, TRUE is under control
int meedpdRun = FALSE;

int meedpdStateThreadRunning = FALSE;
pthread_t meedpdStateThreadId;                                                   //
pthread node manager thread identifier
int meedpdNodemgrThreadRunning = FALSE;
pthread_t meedpdNodemgrThreadId;                                                 //
pthread node manager thread identifier

// Structure storing incoming request under case SET TOOL POINT
static toolPoint_t setToolPoint_6;

// Structure storing incoming request under case SET END EFFECTOR DISCRETE_POSE
static endEffectorPathMotion_t setEndEffectorPathMotion;

// Structure storing incoming request under case QUERY_TOOL_POINT
static toolPoint_t reportedToolPoint;

// Structures storing queried information
static jointEffort_t reportedJointEffort;
static manipulatorSpecifications_t reportedManipulatorSpecifications;
static jointPosition_t reportedJointPosition;

// Structure storing outgoing request to the PRIMITIVE MANIPULATOR
static jointEffort_t setJointEffort;

static double max_speed[NUM_JOINTS], max_acceleration[NUM_JOINTS],
max_deceleration[NUM_JOINTS];
static double cmd_position[NUM_JOINTS];
static int num_soln;
static double orientS[3], orientA[3];
static double des_position[3], des_orientS[3], des_orientA[3], curr_orientation[4],
cmd_orientation[4];
static int pose, numPoses;
static double ref_time = 0, rel_time = 0, init_pose_time = 0, pose_time = 0;

jmh_t meedpdScHeader;
jms_t meedpdJms;      // An accessor to the Node Manager JAUS Message Service (Jms) for
this component
int meedpdScActive = FALSE;

int meedpdPmState = -1;
unsigned char meedpdPmNodeId = 0;
int meedpdPmControl = FALSE;

// Function:   meedpdStartup
// Access:            Public
// Description: This function allows the abstracted component functionality contained in
this file to be started from an external source.
//                            It must be called first before the component state machine
and node manager interaction will begin
//                            Each call to "meedpdStartup" should be followed by one call
to the "cmptShutdown" function
int meedpdStartup(void)
{
       pthread_attr_t attr;  // Thread attributed for the component threads spawned in
this function

       ref_time = getTimeSeconds();

       if(meedpdState == SHUTDOWN_STATE)     // Execute the startup routines only if the
component is not running
       {
```

```
		// Check in to the Node Manager and obtain Instance, Node and Subsystem
IDs
		if(jcmCheckIn(MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER,
&meedpdInstId, &meedpdNodeId, &meedpdSubsId) < 0) // USER: Insert your component ID
define on this line
		{
				cError("meedpd: Could not check in to nodemgr\n");
				return MEEDPD_NODE_MANAGER_CHECKIN_ERROR;
		}

		// Open a conection to the Node Manager and obtain a Jms accessor
		if((meedpdJms = jmsOpen(MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER,
meedpdInstId, 0)) == NULL) // USER: Insert your component ID define on this line
		{
			cError("meedpd: Could not open connection to nodemgr\n");
			jcmCheckOut(MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER,
meedpdInstId); // USER: Insert your component ID define on this line
			return MEEDPD_JMS_OPEN_ERROR;
		}

		meedpdJms->timeout.tv_usec = 100000; // Timeout for non-blocking jmsRecv,
specified in micro seconds

		pthread_attr_init(&attr);
		pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

		meedpdState = INITIALIZE_STATE; // Set the state of the JAUS state machine
to INITIALIZE
		meedpdRun = TRUE;

		if(pthread_create(&meedpdStateThreadId, &attr, meedpdStateThread, NULL) !=
0)
		{
			cError("meedpd: Could not create meedpdStateThread\n");
			meedpdShutdown();
			return MEEDPD_STATE_THREAD_CREATE_ERROR;
		}

		if(pthread_create(&meedpdNodemgrThreadId, &attr, meedpdNodemgrThread,
NULL) != 0)
		{
			cError("meedpd: Could not create meedpdNodemgrThread\n");
			meedpdShutdown();
			return MEEDPD_NODEMGR_THREAD_CREATE_ERROR;
		}

		pthread_attr_destroy(&attr);
	}
	else
	{
		cError("meedpd: Attempted startup while not shutdown\n");
		return MEEDPD_STARTUP_BEFORE_SHUTDOWN_ERROR;
	}

	return 0;
}

// Function: meedpdShutdown
// Access:		Public
// Description:	This function allows the abstracted component functionality
contained in this file to be stoped from an external source.
//				If the component is in the running state, this function
will terminate all threads running in this file
//				This function will also close the Jms connection to the
Node Manager and check out the component from the Node Manager
int meedpdShutdown(void)
{
	double timeOutSec;

	if(meedpdState != SHUTDOWN_STATE)    // Execute the startup routines only if the
component is not running
```

```
	{
		meedpdRun = FALSE;

		timeOutSec = getTimeSeconds() + MEEDPD_THREAD_TIMEOUT_SEC;
		while(meedpdStateThreadRunning)
		{
			usleep(100000);
			if(getTimeSeconds() >= timeOutSec)
			{
				pthread_cancel(meedpdStateThreadId);
				meedpdStateThreadRunning = FALSE;
				cError("meedpd: meedpdStateThread Shutdown Improperly\n");
				break;
			}
		}

		timeOutSec = getTimeSeconds() + MEEDPD_THREAD_TIMEOUT_SEC;
		while(meedpdNodemgrThreadRunning)
		{
			usleep(100000);
			if(getTimeSeconds() >= timeOutSec)
			{
				pthread_cancel(meedpdNodemgrThreadId);
				meedpdNodemgrThreadRunning = FALSE;
				cError("meedpd: meedpdNodemgrThread Shutdown
Improperly\n");
				break;
			}
		}

		// Close Node Manager Connection and check out
		jmsClose(meedpdJms);
		jcmCheckOut(MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER, meedpdInstId);
// USER: Insert your component ID define on this line

		meedpdState = SHUTDOWN_STATE;
	}

	return 0;
}

// The series of functions below allow public access to essential component information
// Access:		Public (All)
int meedpdGetState(void){ return meedpdState; }
unsigned char meedpdGetInstanceId(void){ return meedpdInstId; }
unsigned char meedpdGetCompId(void){ return (unsigned
char)MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER; }
unsigned char meedpdGetNodeId(void){ return meedpdNodeId; }
unsigned char meedpdGetSubsystemId(void){ return meedpdSubsId; }
double meedpdGetUpdateRate(void){ return meedpdThreadHz; }
int meedpdGetPmState(void){ return meedpdPmState; }

// USER: Insert any additional public variable accessors here
toolPoint_t * meedpdSetToolPoint(void) {return &setToolPoint_6; }
endEffectorPathMotion_t * meedpdSetEndEffectorPathMotion(void) {return
&setEndEffectorPathMotion; }
jointEffort_t * meedpdReportJointEffort(void) { return &reportedJointEffort; }
manipulatorSpecifications_t * meedpdReportPmManipulatorSpecifications(void) { return
&reportedManipulatorSpecifications; }
jointPosition_t * meedpdReportJointPosition(void) { return &reportedJointPosition; }
toolPoint_t * meedpdReportToolPoint(void) { return &reportedToolPoint; }
jointEffort_t * meedpdSetJointEffort(void) { return &setJointEffort; }

double * meedpdGetCurrentOrientation(void) { return curr_orientation; }
int meedpdGetPose(void) { return pose; }
double meedpdGetPoseTime(void) {return pose_time; }
double meedpdGetCurrentTime(void) {return rel_time; }
double * meedpdGetCurrentEndEffectorPosition(void) {return des_position; }
double * meedpdGetCurrentEndEffectorOrientation(void) {return cmd_orientation; }

// Function: meedpdThread
```

```
// Access:            Private
// Description:       All core component functionality is contained in this thread.
//                           All of the JAUS component state machine code can be found
here.
void *meedpdStateThread(void *threadData)
{
       jmh_t txMsgHeader;
       double time, prevTime;
       unsigned char data[DATASIZE];
       int i, j;
       double curr_position[NUM_JOINTS], curr_effort[NUM_JOINTS], cmd_effort[NUM_JOINTS],
curr_tool_point[3], tool_point_6[3], cmd_position[3];
       double spec_a[NUM_JOINTS-1], spec_alpha[NUM_JOINTS-1], spec_S[NUM_JOINTS];
       meedpdStateThreadRunning = TRUE; //

       // Setup transmit header
       txMsgHeader.isExpMsg = 0;            // not experimental msg - USER: change to 1
if experimental
       txMsgHeader.isSvcMsg = 0;            // not a service connection
       txMsgHeader.acknCtrl = 0;            // no ack/nak
       txMsgHeader.priority = 6;            // default priority
       txMsgHeader.reserved = 0;
       txMsgHeader.version = 1;             // JAUS 3.0u
       txMsgHeader.srcInstId = meedpdInstId;
       txMsgHeader.srcCompId = MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER;
       txMsgHeader.srcNodeId = meedpdNodeId;
       txMsgHeader.srcSubsId = meedpdSubsId;
       txMsgHeader.dataCtrl = 0;
       txMsgHeader.seqNumber = 0;

       time = getTimeSeconds();
       meedpdState = INITIALIZE_STATE; // Set JAUS state to INITIALIZE

       while(meedpdRun) // Execute state machine code while not in the SHUTDOWN state
       {
              prevTime = time;
              time = getTimeSeconds();
              meedpdThreadHz = 1.0/(time-prevTime); // Compute the update rate of this
thread

              switch(meedpdState) // Switch component behavior based on which state the
machine is in
              {
                     case INITIALIZE_STATE:
                           //Check the status of PRIMITIVE MANIPULATOR
                           meedpdQueryPmComponentStatus();
                           if(meedpdPmState == EMERGENCY_STATE)
                           {
                                  cError("meedpd: PM in emergency state, switching to
emergency state");
                                  meedpdState = EMERGENCY_STATE;
                           }
                           if(meedpdPmState == STANDBY_STATE)
                           {
                                  txMsgHeader.cmdCode = RESUME;
                                  txMsgHeader.dstInstId = 1;
                                  txMsgHeader.dstCompId = PRIMITIVE_MANIPULATOR;
                                  txMsgHeader.dstNodeId = meedpdNodeId;
                                  txMsgHeader.dstSubsId = meedpdSubsId ;
                                  txMsgHeader.dataBytes = 0;
                                  jmsSend(meedpdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                           }
                           if(meedpdPmState == READY_STATE)
                                  meedpdState = STANDBY_STATE;
                           break;

                     case STANDBY_STATE:
                           ref_time = getTimeSeconds();
                           rel_time = getTimeSeconds() - ref_time;
                           pose = 0, j = 0;
```

```
                                break;

                        case READY_STATE:
                                // Set up default velocity, acceleration, and deceleration
values
                                for (i=0; i<NUM_JOINTS; i++)
                                {
                                        if (i == 0){
                                                max_speed[i] = 5000 /
J1_RAD_TO_ENC;//reportedManipulatorSpecifications.jointSpecifications[i].jointMaximumVelo
city / 1000;
                                                max_acceleration[i] = 128000 /
J1_RAD_TO_ENC;
                                        }
                                        else if (i == 1) {
                                                max_speed[i] = 5000 / J2_RAD_TO_ENC;
                                                max_acceleration[i] = 128000 /
J2_RAD_TO_ENC;
                                        }
                                        else if (i == 2) {
                                                max_speed[i] = 5000 / J3_RAD_TO_ENC;
                                                max_acceleration[i] = 128000 /
J3_RAD_TO_ENC;
                                        }
                                        else if (i == 3) {
                                                max_speed[i] = 1000 / J4_RAD_TO_ENC;
                                                max_acceleration[i] = 128000 /
J4_RAD_TO_ENC;
                                        }
                                        else if (i == 4) {
                                                max_speed[i] = 1000 / J5_RAD_TO_ENC;
                                                max_acceleration[i] = 128000 /
J5_RAD_TO_ENC;
                                        }
                                        else {
                                                max_speed[i] = 1000 /
J6_RAD_TO_ENC;//reportedManipulatorSpecifications.jointNmaximumVelocity / 1000;
                                                max_acceleration[i] = 128000 /
J6_RAD_TO_ENC;
                                        }
                                        max_deceleration[i] = max_acceleration[i];
                                }
                                setMaxSpeed(max_speed);
                                setMaxAcceleration(max_acceleration);
                                setMaxDeceleration(max_deceleration);

                                //Check the status of PRIMITIVE MANIPULATOR
                                meedpdQueryPmComponentStatus();
                                if(meedpdPmState == EMERGENCY_STATE)
                                {
                                        cError("meedpd: PM in emergency state, switching to
emergency state");
                                        meedpdState = EMERGENCY_STATE;
                                }

                                numPoses = setEndEffectorPathMotion.numPoses;

                                // Start timer
                                rel_time = getTimeSeconds() - ref_time;

                                init_pose_time =
setEndEffectorPathMotion.endEffectorPoses[0].time;
                                pose_time =
setEndEffectorPathMotion.endEffectorPoses[j].time;

                                // Set zero efforts until first motion profle comes in
                                if (rel_time <= init_pose_time)
                                {
                                        pose = 0;
                                        // Set Joint Efforts to zero
                                        setJointEffort.numJoints = NUM_JOINTS;
```

```
                                        for (i=0; i<NUM_JOINTS; i++)
                                        {
                                                setJointEffort.jointEffort[i] = 0;
                                        }
                                        meedpdSetPmJointEffort();
                                }

                                // Query for required information
                                meedpdQueryMjpsJointPosition();
                                meedpdQueryPmManipulatorSpecifications();
                                meedpdQueryPmJointEffort();

                                for (i=0; i<NUM_JOINTS; i++)
                                {
                                        curr_position[i] =
reportedJointPosition.jointPosition[i];
                                        curr_effort[i] = reportedJointEffort.jointEffort[i];
                                }

                                for(i=0; i<NUM_JOINTS-1; i++)
                                {
                                        spec_a[i] =
reportedManipulatorSpecifications.jointSpecifications[i].linkLength;
                                        spec_alpha[i] =
reportedManipulatorSpecifications.jointSpecifications[i].twistAngle/1000;
                                        if (i<3 || i==4)
                                                spec_S[i] =
reportedManipulatorSpecifications.jointSpecifications[i].jointOffsetOrAngle;
                                        else if (i==3)
                                                spec_S[i] = -
(reportedManipulatorSpecifications.jointSpecifications[i].jointOffsetOrAngle);
                                }
                                spec_S[5] = -
(reportedManipulatorSpecifications.jointNoffsetOrAngle);

                                // Extract incoming tool point in 6th coordinate
                                tool_point_6[0] = 0; // mm
                                tool_point_6[1] = 10;
                                tool_point_6[2] = 0;

                                // This function returns the current position and
orientation of the end-effector
                                getCurrentEndEffectorPose(spec_a, spec_alpha, spec_S,
tool_point_6, curr_position,
                                                                        curr_tool_point,
orientS, orientA);

                                PmatrixToQuaternion(orientA, orientS, curr_orientation);

                                // Report Tool Point
                                reportedToolPoint.X = curr_tool_point[0];
                                reportedToolPoint.Y = curr_tool_point[1];
                                reportedToolPoint.Z = curr_tool_point[2];

                                if (rel_time >= pose_time)
                                {
                                        // Extract information about desired position and
orientation
                                        des_position[0] =
setEndEffectorPathMotion.endEffectorPoses[j].X * 1000; // mm
                                        des_position[1] =
setEndEffectorPathMotion.endEffectorPoses[j].Y * 1000;
                                        des_position[2] =
setEndEffectorPathMotion.endEffectorPoses[j].Z * 1000;
                                        cmd_orientation[0] =
setEndEffectorPathMotion.endEffectorPoses[j].quaternionQcomponentD;
                                        cmd_orientation[1] =
setEndEffectorPathMotion.endEffectorPoses[j].quaternionQcomponentA;
                                        cmd_orientation[2] =
setEndEffectorPathMotion.endEffectorPoses[j].quaternionQcomponentB;
```

```
                                        cmd_orientation[3] =
setEndEffectorPathMotion.endEffectorPoses[j].quaternionQcomponentC;

                                        PquaternionToMatrix(cmd_orientation, des_orientS,
des_orientA);

                                        // This function returns the joint angles
corresponding to the desired pose
                                        getDesiredJointAngles(des_position, des_orientS,
des_orientA, cmd_position);
                                        if (j < numPoses-1){
                                                pose++;
                                                j++;
                                        }
                                        else {
                                                j = numPoses-1;
                                                pose = numPoses;
                                        }
                                }

                                // This function performs closed loop end effector pose
control
                                endEffectorDiscretePoseDriver(cmd_position, curr_position,
cmd_effort);

                                // Set Joint Efforts
                                setJointEffort.numJoints = NUM_JOINTS;
                                for (i=0; i<NUM_JOINTS; i++)
                                {
                                        setJointEffort.jointEffort[i] = cmd_effort[i];
                                }
                                meedpdSetPmJointEffort();

                                break;

                        case EMERGENCY_STATE:
                                break;

                        case FAILURE_STATE:
                                break;

                        case SHUTDOWN_STATE:
                                break;

                        default:
                                meedpdState = FAILURE_STATE; // The default case is
undefined, therefore go into Failure State
                                break;
                }
                usleep(1000); // Sleep for one millisecond
        }

        // Sleep for 50 milliseconds and then exit
        usleep(50000);

        meedpdStateThreadRunning = FALSE;

        pthread_exit(NULL);
}

// Function: meedpdNodemgrThread
// Access:            Private
// Description:       This thread is responsible for handling incoming JAUS messages from
the Node Manager JAUS message service (Jms)
//                              incoming messages are processed according to message type.
void *meedpdNodemgrThread(void *threadData)
{
        jmh_t rxMsgHeader, txMsgHeader; // Declare JAUS header data structures, where the
jmh_t data structure is defined in jaus.h
        unsigned char data[DATASIZE];
        createServiceConnection_t createServiceConnection;
```

```
        confirmServiceConnection_t confirmServiceConnection;
        activateServiceConnection_t activateServiceConnection;
        suspendServiceConnection_t suspendServiceConnection;
        terminateServiceConnection_t terminateServiceConnection;
        requestComponentControl_t requestComponentControl;
        confirmComponentControl_t confirmComponentControl;
        componentAuthority_t componentAuthority;
        componentStatus_t componentStatus;
        int recvCount;

        meedpdNodemgrThreadRunning = TRUE; //

        // Setup transmit header
        txMsgHeader.isExpMsg = 0;            // not experimental msg - USER: change to 1
if experimental
        txMsgHeader.isSvcMsg = 0;            // not a service connection
        txMsgHeader.acknCtrl = 0;            // no ack/nak
        txMsgHeader.priority = 6;            // default priority
        txMsgHeader.reserved = 0;
        txMsgHeader.version = 1;             // JAUS 3.0
        txMsgHeader.srcInstId = meedpdInstId;
        txMsgHeader.srcCompId = MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER;
        txMsgHeader.srcNodeId = meedpdNodeId;
        txMsgHeader.srcSubsId = meedpdSubsId;
        txMsgHeader.dataCtrl = 0;
        txMsgHeader.seqNumber = 0;

        while(meedpdRun)  // Execute incoming JAUS message processing while not in the
SHUTDOWN state
        {
                // Check for a new message in the Node Manager JMS queue
                // If a message is present then store the header information in
rxMsgHeader, and store the message content information in the data buffer
                recvCount = jmsRecv(meedpdJms, &rxMsgHeader, data, DATASIZE);
                if(recvCount < 0)
                {
                        cError("meedpd: Node manager jmsRecv returned error: %d\n",
recvCount);
                        break;
                }
                if(recvCount == 0) continue;

                // This block of code is intended to reject commands from non-controlling
components
                // However, the one exception allowed is a REQUEST_COMPONENT_CONTROL
                if(meedpdUnderControl && rxMsgHeader.cmdCode < 0x2000) // If not currently
under control or it's not a command, then go to the switch statement
                {
                        // If the source component isn't the one in control, and it's not a
request to gain control, then exit this iteration of the while loop
                        if(     !(      meedpdCtrlCmptInstId==rxMsgHeader.srcInstId &&
                                        meedpdCtrlCmptCompId==rxMsgHeader.srcCompId &&
                                        meedpdCtrlCmptNodeId==rxMsgHeader.srcNodeId &&
                                        meedpdCtrlCmptSubsId==rxMsgHeader.srcSubsId)&&
                                !(      rxMsgHeader.cmdCode==REQUEST_COMPONENT_CONTROL)
                        )continue; // to next iteration
                }

                switch(rxMsgHeader.cmdCode) // Switch the processing algorithm according
to the JAUS message type
                {
                        // Set the component authority according to the incoming authority
code
                        case SET_COMPONENT_AUTHORITY:
                                convertComponentAuthority(data, &meedpdAuthority, DATASIZE,
UNPACK);       // Unpack and store the incoming authority code
                                break;

                        case SHUTDOWN:
                                meedpdState = SHUTDOWN_STATE;
                                break;
```

```
                        case STANDBY:
                                if(meedpdState == READY_STATE)
                                        meedpdState = STANDBY_STATE;
                                break;

                        case RESUME:
                                if(meedpdState == STANDBY_STATE)
                                        meedpdState = READY_STATE;
                                break;

                        case RESET:
                                meedpdState = INITIALIZE_STATE;
                                break;

                        case SET_EMERGENCY:
                                meedpdState = EMERGENCY_STATE;
                                break;

                        case CLEAR_EMERGENCY:
                                meedpdState = STANDBY_STATE;
                                break;

                        case CREATE_SERVICE_CONNECTION:
                                convertCreateServiceConnection(data,
&createServiceConnection, DATASIZE, UNPACK);
                                if(createServiceConnection.cmdCode ==
REPORT_JOINT_VELOCITY)
                                {
                                        meedpdScHeader = txMsgHeader;
                                        meedpdScHeader.isSvcMsg = 1;
                                        meedpdScHeader.dstInstId = rxMsgHeader.srcInstId;
                                        meedpdScHeader.dstCompId = rxMsgHeader.srcCompId;
                                        meedpdScHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                        meedpdScHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                        meedpdScHeader.cmdCode = REPORT_JOINT_VELOCITY;
                                        meedpdScHeader.seqNumber = 1;

                                        confirmServiceConnection.cmdCode =
REPORT_JOINT_VELOCITY;
                                        confirmServiceConnection.instanceId = 1;
                                        confirmServiceConnection.confirmedRate =
createServiceConnection.requestedRate; // BUG: The actual rate may not be the same as the
request
                                        confirmServiceConnection.responseCode =
CONNECTION_SUCCESSFUL;
                                        txMsgHeader.cmdCode = CONFIRM_SERVICE_CONNECTION;
                                        txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                        txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                        txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                        txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                        txMsgHeader.dataBytes =
convertConfirmServiceConnection(data, &confirmServiceConnection, DATASIZE, PACK);
                                        jmsSend(meedpdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);

                                        meedpdScActive = TRUE;
                                }
                                else
                                {
                                        confirmServiceConnection.cmdCode =
createServiceConnection.cmdCode;
                                        confirmServiceConnection.instanceId = 1;
                                        confirmServiceConnection.confirmedRate =
createServiceConnection.requestedRate;
                                        confirmServiceConnection.responseCode =
CONNECTION_REFUSED;
                                        txMsgHeader.cmdCode = CONFIRM_SERVICE_CONNECTION;
                                        txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                        txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                        txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
```

```
                                    txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                    txMsgHeader.dataBytes =
convertConfirmServiceConnection(data, &confirmServiceConnection, DATASIZE, PACK);
                                    jmsSend(meedpdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                             }
                             break;

                      case CONFIRM_SERVICE_CONNECTION:
                             convertConfirmServiceConnection(data,
&confirmServiceConnection, DATASIZE, UNPACK);
                             // USER: Insert code here to handle the confirm service
connection message
                             break;

                      case ACTIVATE_SERVICE_CONNECTION:
                             convertActivateServiceConnection(data,
&activateServiceConnection, DATASIZE, UNPACK);
                             // USER: Insert code here to handle the activate service
connection message
                             break;

                      case SUSPEND_SERVICE_CONNECTION:
                             convertSuspendServiceConnection(data,
&suspendServiceConnection, DATASIZE, UNPACK);
                             // USER: Insert code here to handle the suspend service
connection message
                             break;

                      case TERMINATE_SERVICE_CONNECTION:
                             convertTerminateServiceConnection(data,
&terminateServiceConnection, DATASIZE, UNPACK);
                             if (terminateServiceConnection.cmdCode ==
REPORT_JOINT_VELOCITY)
                                    meedpdScActive = FALSE;
                             break;

                      case REQUEST_COMPONENT_CONTROL:
                             convertRequestComponentControl(data,
&requestComponentControl, DATASIZE, UNPACK);

                             if(meedpdUnderControl)
                             {
                                    if(requestComponentControl >
meedpdCtrlCmptAuthority) // Test for higher authority
                                    {
                                           // Terminate control of current component
                                           txMsgHeader.cmdCode    =
REJECT_COMPONENT_CONTROL;
                                           txMsgHeader.dstSubsId =
meedpdCtrlCmptSubsId;
                                           txMsgHeader.dstNodeId =
meedpdCtrlCmptNodeId;
                                           txMsgHeader.dstCompId =
meedpdCtrlCmptCompId;
                                           txMsgHeader.dstInstId =
meedpdCtrlCmptInstId;
                                           txMsgHeader.dataBytes = 0;
                                           jmsSend(meedpdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);

                                           // Accept control of new component
                                           txMsgHeader.cmdCode    =
CONFIRM_COMPONENT_CONTROL;
                                           txMsgHeader.dstSubsId =
rxMsgHeader.srcSubsId;
                                           txMsgHeader.dstNodeId =
rxMsgHeader.srcNodeId;
                                           txMsgHeader.dstCompId =
rxMsgHeader.srcCompId;
```

```
                                        txMsgHeader.dstInstId =
rxMsgHeader.srcInstId;
                                        confirmComponentControl.asByte = 0;
                                        txMsgHeader.dataBytes =
convertConfirmComponentControl(data, &confirmComponentControl, DATASIZE, PACK);
                                        jmsSend(meedpdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                        meedpdCtrlCmptInstId =
rxMsgHeader.srcInstId;
                                        meedpdCtrlCmptCompId =
rxMsgHeader.srcCompId;
                                        meedpdCtrlCmptNodeId =
rxMsgHeader.srcNodeId;
                                        meedpdCtrlCmptSubsId =
rxMsgHeader.srcSubsId;
                                        meedpdCtrlCmptAuthority =
requestComponentControl;
                                }
                                else
                                {
                                        if(     rxMsgHeader.srcSubsId !=
meedpdCtrlCmptSubsId || rxMsgHeader.srcNodeId != meedpdCtrlCmptNodeId ||
                                                rxMsgHeader.srcCompId !=
meedpdCtrlCmptCompId || rxMsgHeader.srcInstId != meedpdCtrlCmptInstId )
                                        {
                                                txMsgHeader.cmdCode    =
REJECT_COMPONENT_CONTROL;
                                                txMsgHeader.dstSubsId =
rxMsgHeader.srcSubsId;
                                                txMsgHeader.dstNodeId =
rxMsgHeader.srcNodeId;
                                                txMsgHeader.dstCompId =
rxMsgHeader.srcCompId;
                                                txMsgHeader.dstInstId =
rxMsgHeader.srcInstId;
                                                txMsgHeader.dataBytes = 0;
                                                jmsSend(meedpdJms, &txMsgHeader,
data, txMsgHeader.dataBytes);
                                        }
                                }
                        }
                        else // Not currently under component control, so give
control
                        {
                                txMsgHeader.cmdCode    = CONFIRM_COMPONENT_CONTROL;
                                txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                                txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                                txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                                txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                                confirmComponentControl.asByte = 0;
                                txMsgHeader.dataBytes =
convertConfirmComponentControl(data, &confirmComponentControl, DATASIZE, PACK);
                                jmsSend(meedpdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                                meedpdCtrlCmptInstId = rxMsgHeader.srcInstId;
                                meedpdCtrlCmptCompId = rxMsgHeader.srcCompId;
                                meedpdCtrlCmptNodeId = rxMsgHeader.srcNodeId;
                                meedpdCtrlCmptSubsId = rxMsgHeader.srcSubsId;
                                meedpdCtrlCmptAuthority = requestComponentControl;
                                meedpdUnderControl = TRUE;
                        }
                        break;

                case RELEASE_COMPONENT_CONTROL:
                        meedpdUnderControl = FALSE;
                        break;

                case CONFIRM_COMPONENT_CONTROL:
                        convertConfirmComponentControl(data,
&confirmComponentControl, DATASIZE, UNPACK);
```

```
                              // USER: Insert code here to handle the confirm component
control message if needed
                              break;

                       case REJECT_COMPONENT_CONTROL:
                              // USER: Insert code here to handle the reject component
control message if needed
                              break;

                       case QUERY_COMPONENT_AUTHORITY:
                              txMsgHeader.cmdCode    = REPORT_COMPONENT_AUTHORITY;
                              txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                              txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                              txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                              txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                              componentAuthority = meedpdAuthority;
                              txMsgHeader.dataBytes = convertComponentAuthority(data,
&componentAuthority, DATASIZE, PACK);
                              jmsSend(meedpdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                              break;

                       case QUERY_COMPONENT_STATUS:
                              txMsgHeader.cmdCode    = REPORT_COMPONENT_STATUS;
                              txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                              txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                              txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                              txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                              componentStatus.primaryStatus.asByte = meedpdState;
                              txMsgHeader.dataBytes = convertComponentStatus(data,
&componentStatus, DATASIZE, PACK);
                              jmsSend(meedpdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                              break;

                       case REPORT_COMPONENT_AUTHORITY:
                              convertComponentAuthority(data, &componentAuthority,
DATASIZE, UNPACK);
                              // USER: Insert code here to handle the report component
authority message if needed
                              break;

                       case REPORT_COMPONENT_STATUS:
                              convertComponentStatus(data, &componentStatus, DATASIZE,
UNPACK);
                              if(rxMsgHeader.srcCompId == PRIMITIVE_MANIPULATOR)
                                     meedpdPmState =
componentStatus.primaryStatus.asField.primaryStatusCode;
                              break;

                       case SET_END_EFFECTOR_PATH_MOTION:
                              convertEndEffectorPathMotion(data,
&setEndEffectorPathMotion, DATASIZE, UNPACK);
                              break;

                       case REPORT_JOINT_EFFORT:
                              convertJointEffort(data, &reportedJointEffort, DATASIZE,
UNPACK);
                              break;

                       case REPORT_MANIPULATOR_SPECIFICATIONS:
                              convertManipulatorSpecifications(data,
&reportedManipulatorSpecifications, DATASIZE, UNPACK);
                              break;

                       case REPORT_JOINT_POSITION:
                              convertJointPosition(data, &reportedJointPosition,
DATASIZE, UNPACK);
                              break;

                       case QUERY_TOOL_POINT:
```

```
                        txMsgHeader.cmdCode = REPORT_TOOL_POINT;
                        txMsgHeader.dstInstId = rxMsgHeader.srcInstId;
                        txMsgHeader.dstCompId = rxMsgHeader.srcCompId;
                        txMsgHeader.dstNodeId = rxMsgHeader.srcNodeId;
                        txMsgHeader.dstSubsId = rxMsgHeader.srcSubsId;
                        txMsgHeader.dataBytes = convertToolPoint(data,
&reportedToolPoint, DATASIZE, PACK);
                            jmsSend(meedpdJms, &txMsgHeader, data,
txMsgHeader.dataBytes);
                            break;

                    default:
                            break;
                }
        }

        meedpdNodemgrThreadRunning = FALSE;
        pthread_exit(NULL);
}

////////////////////////////////////////////////////////////////////////////////
// Function used to send a message to the primitive manipulator and query for
// component status. Only the primitive manipulator component performs the initial
// system checks. Thus it is essential that primitive manipulator is running and its
// status is ready before any other components are started up.
////////////////////////////////////////////////////////////////////////////////
void meedpdQueryPmComponentStatus(void)
{
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;

        // Setup Transmit msg
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcInstId = meedpdInstId;
        txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER;
        txMsg.srcNodeId = meedpdNodeId;
        txMsg.srcSubsId = meedpdSubsId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = QUERY_COMPONENT_STATUS;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = PRIMITIVE_MANIPULATOR;
        txMsg.dstNodeId = meedpdNodeId;
        txMsg.dstSubsId = meedpdSubsId ;
        txMsg.dataBytes = 0;

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(meedpdJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}

////////////////////////////////////////////////////////////////////////////////
// Function used to set up a message sent to manipulator joint position sensor
// to obtain current joint position data.
////////////////////////////////////////////////////////////////////////////////
void meedpdQueryMjpsJointPosition(void)
{
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;
```

```
	// Setup Transmit msg
	txMsg.isExpMsg = 0;
	txMsg.priority = 6;
	txMsg.acknCtrl = 0;
	txMsg.isSvcMsg = 0;
	txMsg.version = 1;
	txMsg.reserved = 0;
	txMsg.srcSubsId = meedpdSubsId;
	txMsg.srcNodeId = meedpdNodeId;
	txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER;
	txMsg.srcInstId = meedpdInstId;
	txMsg.dataCtrl = 0;
	txMsg.seqNumber = 0;
	txMsg.cmdCode = QUERY_JOINT_POSITION;
	txMsg.dstInstId = 1;
	txMsg.dstCompId = MANIPULATOR_JOINT_POSITION_SENSOR;
	txMsg.dstNodeId = meedpdNodeId;
	txMsg.dstSubsId = meedpdSubsId ;
	txMsg.dataBytes = 0;

	if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
	{
		jmsSend(meedpdJms, &txMsg, data, txMsg.dataBytes);
		lastRequestTimeSec = getTimeSeconds();
	}
}

////////////////////////////////////////////////////////////////////////////////
// Function used to set up a message sent to the primitive manipulator to obtain
// current effort values.
////////////////////////////////////////////////////////////////////////////////
void meedpdQueryPmJointEffort(void)
{
	jmh_t txMsg;
	unsigned char data[DATASIZE];
	double requestTimeOutSec = 0.1;
	static double lastRequestTimeSec = 0;

	// Setup Transmit msg
	txMsg.isExpMsg = 0;
	txMsg.priority = 6;
	txMsg.acknCtrl = 0;
	txMsg.isSvcMsg = 0;
	txMsg.version = 1;
	txMsg.reserved = 0;
	txMsg.srcSubsId = meedpdSubsId;
	txMsg.srcNodeId = meedpdNodeId;
	txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER;
	txMsg.srcInstId = meedpdInstId;
	txMsg.dataCtrl = 0;
	txMsg.seqNumber = 0;
	txMsg.cmdCode = QUERY_JOINT_EFFORT;
	txMsg.dstInstId = 1;
	txMsg.dstCompId = PRIMITIVE_MANIPULATOR;
	txMsg.dstNodeId = meedpdNodeId;
	txMsg.dstSubsId = meedpdSubsId ;
	txMsg.dataBytes = 0;

	if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
	{
		jmsSend(meedpdJms, &txMsg, data, txMsg.dataBytes);
		lastRequestTimeSec = getTimeSeconds();
	}
}

////////////////////////////////////////////////////////////////////////////////
// Function used to set up a message sent to the primitive manipulator to obtain
// manipulator specifications.
////////////////////////////////////////////////////////////////////////////////
void meedpdQueryPmManipulatorSpecifications(void)
{
```

```
        jmh_t txMsg;
        unsigned char data[DATASIZE];
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;

        // Setup Transmit msg
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcSubsId = meedpdSubsId;
        txMsg.srcNodeId = meedpdNodeId;
        txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER;
        txMsg.srcInstId = meedpdInstId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = QUERY_MANIPULATOR_SPECIFICATIONS;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = PRIMITIVE_MANIPULATOR;
        txMsg.dstNodeId = meedpdNodeId;
        txMsg.dstSubsId = meedpdSubsId ;
        txMsg.dataBytes = 0;

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(meedpdJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}

////////////////////////////////////////////////////////////////////////////////
// Function used to set up a message sent to the primitive manipulator with new
// joint effort values.
////////////////////////////////////////////////////////////////////////////////
void meedpdSetPmJointEffort(void)
{
        jmh_t txMsg;
        double requestTimeOutSec = 0.1;
        static double lastRequestTimeSec = 0;
        unsigned char data[DATASIZE];

        //Setup Transmit Message
        txMsg.isExpMsg = 0;
        txMsg.priority = 6;
        txMsg.acknCtrl = 0;
        txMsg.isSvcMsg = 0;
        txMsg.version = 1;
        txMsg.reserved = 0;
        txMsg.srcInstId = meedpdInstId;
        txMsg.srcCompId = MANIPULATOR_END_EFFECTOR_DISCRETE_POSE_DRIVER;
        txMsg.srcNodeId = meedpdNodeId;
        txMsg.srcSubsId = meedpdSubsId;
        txMsg.dataCtrl = 0;
        txMsg.seqNumber = 0;
        txMsg.cmdCode = SET_JOINT_EFFORT;
        txMsg.dstInstId = 1;
        txMsg.dstCompId = PRIMITIVE_MANIPULATOR;
        txMsg.dstNodeId = meedpdNodeId;
        txMsg.dstSubsId = meedpdSubsId;
        txMsg.dataBytes = convertJointEffort(data, &setJointEffort, DATASIZE, PACK);

        if( (getTimeSeconds() - lastRequestTimeSec) >= requestTimeOutSec)
        {
                jmsSend(meedpdJms, &txMsg, data, txMsg.dataBytes);
                lastRequestTimeSec = getTimeSeconds();
        }
}

////////////////////////////////////////////////////////////////////////////////
```

```
// Function used to perform closed loop end-effector position control
////////////////////////////////////////////////////////////////////////////////
void endEffectorDiscretePoseDriver(double *cmd_position, double *curr_position, double
*cmd_effort)
{
	int i;
	double k[NUM_JOINTS];

	for (i=0; i<NUM_JOINTS; i++)
	{
		if (i < 3) k[i] = 34;
		else if (i == 3) k[i] = 34;
		else if (i == 4) k[i] = 15;
		else k[i] = 14;

		if (i == 0)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J1_RAD_TO_ENC * 100 / 112000;
		else if (i == 1)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J2_RAD_TO_ENC * 100 / 112000;
		else if (i == 2)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J3_RAD_TO_ENC * 100 / 112000;
		else if (i == 3)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J4_RAD_TO_ENC * 100 / 35000;
		else if (i == 4)
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J5_RAD_TO_ENC * 100 / 16000;
		else
			cmd_effort[i] = ((cmd_position[i] - curr_position[i]) * k[i]) *
J6_RAD_TO_ENC * 100 / 15500;
	}
}

void PmatrixToQuaternion(double *orientA, double *orientS, double *curr_orientation)
{
	double T, S;
	double W, X, Y, Z;
	double M0, M1, M2;
	double M4, M5, M6;
	double M8, M9, M10;

	M0 = orientA[0];
	M4 = orientA[1];
	M8 = orientA[2];

	M1 = orientS[1]*orientA[2] - orientS[2]*orientA[1];
	M5 = orientS[2]*orientA[0] - orientS[0]*orientA[2];
	M9 = orientS[0]*orientA[1] - orientS[1]*orientA[0];

	M2 = orientS[0];
	M6 = orientS[1];
	M10 = orientS[2];

	T = M0 + M5 + M10 + 1;
	S = 0.5/sqrt(T);

	W = 0.25/S;
	X = (M9 - M6)*S;
	Y = (M2 - M8)*S;
	Z = (M4 - M1)*S;

	curr_orientation[0] = W;
	curr_orientation[1] = X;
	curr_orientation[2] = Y;
	curr_orientation[3] = Z;
}
```

```
void PquaternionToMatrix(double *cmd_orientation,double *des_orientS, double
*des_orientA)
{
        double W, X, Y, Z;
        double M0, M1, M2;
        double M4, M5, M6;
        double M8, M9, M10;

        W = cmd_orientation[0];
        X = cmd_orientation[1];
        Y = cmd_orientation[2];
        Z = cmd_orientation[3];

        M0 = 1 - 2*Y*Y - 2*Z*Z;
        M4 = 2*X*Y + 2*Z*W;
        M8 = 2*X*Z - 2*Y*W;

        M1 = 2*X*Y - 2*Z*W;
        M5 = 1 - 2*X*X - 2*Z*Z;
        M9 = 2*Y*Z + 2*X*W;

        M2 = 2*X*Z + 2*Y*W;
        M6 = 2*Y*Z - 2*X*W;
        M10 = 1 - 2*X*X - 2*Y*Y;

        des_orientA[0] = M0;
        des_orientA[1] = M4;
        des_orientA[2] = M8;

        des_orientS[0] = M2;
        des_orientS[1] = M6;
        des_orientS[2] = M10;

}


////////////////////////////////////////////////////////////////////////////////
// File:               meedpd.h
// Version:            0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:        Tom Galluzzo (galluzzt@ufl.edu)
// Date:               09/28/2004
//
// Description:        This file contains the skeleton C header file code for implementing
//                             the meedpd.c file
////////////////////////////////////////////////////////////////////////////////

#ifndef MEEDPD_H
#define MEEDPD_H

#include "jaus.h"

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE 1
#endif

#define MEEDPD_NODE_MANAGER_CHECKIN_ERROR           -1
#define MEEDPD_JMS_OPEN_ERROR                               -2
#define MEEDPD_STARTUP_BEFORE_SHUTDOWN_ERROR -3
#define MEEDPD_STATE_THREAD_CREATE_ERROR            -4
#define MEEDPD_NODEMGR_THREAD_CREATE_ERROR   -5

#define MEEDPD_THREAD_TIMEOUT_SEC                           1.0

// Public
int meedpdStartup(void);
int meedpdShutdown(void);
int meedpdGetState(void);
```

```
unsigned char meedpdGetInstanceId(void);
unsigned char meedpdGetCompId(void);
unsigned char meedpdGetNodeId(void);
unsigned char meedpdGetSubsystemId(void);
double meedpdGetUpdateRate(void);
int meedpdGetPmState(void);

// Query messages requested by this component
void meedpdQueryPmComponentStatus(void);
void meedpdQueryPmJointEffort(void);
void meedpdQueryMjpsJointPosition(void);
void meedpdQueryPmManipulatorSpecifications(void);

// Set messages that this component commands
void meedpdSetPmJointEffort(void);

// Accessors to incoming information
toolPoint_t * meedpdSetToolPoint(void);
endEffectorPathMotion_t * meedpdSetEndEffectorPathMotion(void);
jointEffort_t * meedpdReportJointEffort(void);
manipulatorSpecifications_t * meedpdReportManipulatorSpecifications(void);
jointPosition_t * meedpdReportJointPosition(void);

// Accessors to outgoing information
toolPoint_t * meedpdReportToolPoint(void);
jointEffort_t * meedpdSetJointEffort(void);

double * meedpdGetCurrentOrientation(void);
int meedpdGetPose(void);
double meedpdGetPoseTime(void);
double meedpdGetCurrentTime(void);
double * meedpdGetCurrentEndEffectorPosition(void);
double * meedpdGetCurrentEndEffectorOrientation(void);
void PmatrixToQuaternion(double *, double *, double *);
void PquaternionToMatrix(double *, double *, double *);

void endEffectorDiscretePoseDriver(double *, double *, double *);

#endif // MEEDPD_H
```

B.6 The Mc.c File and the Corresponding Header Mc.h

```
////////////////////////////////////////////////////////////////////////////////
// File:              mc.c
// Version:           0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Date:              09/28/2004
//
// Description: This file contains node startup and shutdown functions. "Manipulator
//             Control" (mc) node is used to spawn all the neccessary threads. Due to
//             very low bandwith to the Galil controller, only one of the driver
//             components can be running at the time.
////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <time.h>
#include <unistd.h>

#include "mc.h"

int roboWorksReady = 0;

int mcStartup(void)
{
       char choice;
       int i;

       // Check if RoboWorks software is being used
       do {
```

```
        printf("Are you using RoboWorks 3D visualization tool? (y or n): ");
        scanf("%c",&choice);
} while (choice != 'y' && choice != 'n');

if (choice == 'y')
        roboWorksReady = 1;
else if (choice == 'n')
        roboWorksReady = 0;

// Start controllers
startGALILController();
if (getGalilReady() == 0)
{
        printf("Could not establish communications with GALIL controller.\n");
        while (getGalilReady() == 0)
        {
                startGALILController();
                usleep(10000);
        }
        printf("Communications with GALIL controller established.\n");
}
else
        printf("Communications with GALIL controller established.\n");

// Reset Galil
i = resetGalil();
if (i == 0)
        printf("GALIL reset successful.\n");
else
        printf("GALIL reset failed.\n");

startVALController();
if (getValReady() == 1)
{
        printf("VAL controller is not on. Switch main power ON.\n");
        while (getValReady() == 1)
        {
                startVALController();
                usleep(10000);
        }
        printf("VAL controller is on.\n");
}
else
        printf("VAL controller is on.\n");

// Allows for remote operation of the manipulator
startARM();
if (getArmReady() == 1)
{
        printf("Switch PUMA 762 on.\n");
        while (getArmReady() == 1)
        {
                startARM();
                usleep(1000);
        }
        printf("PUMA 762 is on.\n");
}
else
        printf("PUMA 762 is on.\n");


// Start up the interface thread
interfaceStartup();

// Start component threads
// Primitive manipulator and sensor components should always be running
// Comment out startup routines of driver components you are not planning to use
if(pmStartup())
{
        cError("mc: pmStartup failed\n");
        mcShutdown();
```

```
                return MC_PM_STARTUP_FAILED;
        }

        if (mjpsStartup())
        {
                cError("mc: mjpsStartup failed\n");
                mjpsShutdown();

                return MC_MJPS_STARTUP_FAILED;
        }

        if (mjvsStartup())
        {
                cError("mc: mjvsStartup failed\n");
                mjvsShutdown();

                return MC_MJVS_STARTUP_FAILED;
        }

        if (mjftsStartup())
        {
                cError("mc: mjftsStartup failed\n");
                mjftsShutdown();

                return MC_MJFTS_STARTUP_FAILED;
        }

        if (mjpdStartup())
        {
                cError("mc: mjpdStartup failed\n");
                mjpdShutdown();

                return MC_MJPD_STARTUP_FAILED;
        }

        if (mjvdStartup())
        {
                cError("mc: mjvdStartup failed\n");
                mjvdShutdown();

                return MC_MJVD_STARTUP_FAILED;
        }


        if (mjmdStartup())
        {
                cError("mc: mjmdStartup failed\n");
                mjmdShutdown();

                return MC_MJMD_STARTUP_FAILED;
        }

        if (meepdStartup())
        {
                cError("mc: meepdStartup failed\n");
                meepdShutdown();

                return MC_MEEPD_STARTUP_FAILED;
        }

        if (meedpdStartup())
        {
                cError("mc: meedpdStartup failed\n");
                meedpdShutdown();

                return MC_MEEDPD_STARTUP_FAILED;
        }

        return 0;
}
```

```
int mcShutdown(void)
{
        pmShutdown();
        mjpsShutdown();
        mjvsShutdown();
        mjftsShutdown();
        mjpdShutdown();
        mjvdShutdown();
        mjmdShutdown();
        meepdShutdown();
        meedpdShutdown();
        interfaceShutdown();

        return 0;
}

int getRoboWorksReady(void) { return roboWorksReady; }

const char *mcGetName(void)
{
        static const char *mcName = MC_NAME_STRING;

        return mcName;
}


////////////////////////////////////////////////////////////////////////////////
// File:              mc.h
// Version:           0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Date:              09/28/2004
//
// Description:       This file contians function prototypes required for mc.c
////////////////////////////////////////////////////////////////////////////////

#ifndef NODE_H
#define NODE_H

#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif

#define MC_NAME_STRING               "Manipulator Control"

#define MC_PM_STARTUP_FAILED  -1
#define MC_MJPS_STARTUP_FAILED       -2
#define MC_MJVS_STARTUP_FAILED  -3
#define MC_MJFTS_STARTUP_FAILED      -4
#define MC_MJPD_STARTUP_FAILED       -5
#define MC_MEEPD_STARTUP_FAILED -6
#define MC_MJVD_STARTUP_FAILED  -7
#define MC_MJMD_STARTUP_FAILED  -9
#define MC_MEEDPD_STARTUP_FAILED -10

int mcStartup(void);
int mcShutdown(void);

int getRoboWorksReady(void);

const char *mcGetName(void);

#include "pm.h"
#include "mjps.h"
#include "mjvs.h"
#include "mjfts.h"
#include "mjpd.h"
#include "mjvd.h"
```

```
#include "mjmd.h"
#include "meepd.h"
#include "meedpd.h"

#include "galilInterface.h"
#include "mcConstants.h"
#include "cppInterface.h"

#include "logLib.h"
#include "timeLib.h"

#include "dmclnx.h"
#include "RoboTalk.h"

#endif // MC_H
```

B.7 The Main.c File

```
////////////////////////////////////////////////////////////////////////////////
// File:               main.c
// Version:            0.1 Original Creation
// Written by: Ognjen Sosa (ognjensosa@hotmail.com)
// Template by:        Tom Galluzzo (galluzzt@ufl.edu)
// Date:               09/28/2004
//
// Description: This file contains the C code used to initiate node startup and display
//                            component information.
////////////////////////////////////////////////////////////////////////////////

#include <curses.h>
#include <ctype.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <termios.h>
#include <time.h>
#include <unistd.h>

#include "pm.h"
#include "galilInterface.h"
#include "cppInterface.h"
#include "logLib.h"
#include "mc.h"
#include "timeLib.h"
#include "mcConstants.h"
#include "mjvs.h"
#include "mjps.h"
#include "mjfts.h"
#include "mjpd.h"
#include "mjvd.h"
#include "mjmd.h"
#include "meepd.h"

#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif

void updateScreen(int page);
void signalInterventionHandler(int);
const char *jausStateString(int);

int mainRunning;
```

```
int main(int argCount, char **argString)
{
	int i;
	char choice=0;
	int verbose = FALSE;
	int debugLevel = 0;
	struct termios newTermios;
	struct termios stored;
	FILE *logFile = NULL;
	char logFileStr[128] = {0};
	time_t timeStamp;
	char timeString[64];
	int page = 0;

	//Get and Format Time String
	time(&timeStamp);
	strftime(timeString, 63, "%m-%d-%Y %X", localtime(&timeStamp));

	system("clear");

	printf("main: CIMAR Core Executable: %s\n", timeString);

	if(mkdir("/var/log/CIMAR/", 0777) < 0)
	{
		if(errno != EEXIST)
		{
			perror("main: Error");
			return -1;
		}
	}

	for(i=1; i<argCount; i++)
	{
		if(argString[i][0] == '-')
		{
			switch(argString[i][1])
			{
				case 'v':
					verbose = TRUE;
					setLogVerbose(TRUE);
					break;

				case 'd':
					if(argString[i][2] == '+')
					{
						setDebugLogic(DEBUG_GREATER_THAN);
						debugLevel = atoi(&argString[i][3]);
					}
					else if(argString[i][2] == '-')
					{
						setDebugLogic(DEBUG_LESS_THAN);
						debugLevel = atoi(&argString[i][3]);
					}
					else if(argString[i][2] == '=')
					{
						setDebugLogic(DEBUG_EQUAL_TO);
						debugLevel = atoi(&argString[i][3]);
					}
					else if(argString[i][2] >= '0' && argString[i][2] <=
'9')
					{
						debugLevel = atoi(&argString[i][2]);
					}
					else
					{
						printf("main: Incorrect use of
arguments\n");
						break;
					}
```

```
                                        printf("main: Entering debug level: %d\n",
debugLevel);
                                        setDebugLevel(debugLevel);
                                        break;

                                case 'l':
                                        if(argCount > i+1 && argString[i+1][0] != '-')
                                        {
                                                logFile = fopen(argString[i+1], "w");
                                                if(logFile != NULL)
                                                {
                                                        fprintf(logFile, "CIMAR %s Log --
%s\n", argString[0], timeString);
                                                        setLogFile(logFile);
                                                }
                                                else printf("main: Error creating log file,
switching to default\n");
                                        }
                                        else
                                                printf("main: Incorrect use of
arguments\n");

                                        break;

                                default:
                                        printf("main: Incorrect use of arguments\n");
                                        break;
                        }
                }
        }

        if(logFile == NULL)
        {
                i = strlen(argString[0]) - 1;
                while(i>0 && argString[0][i-1] != '/') i--;

                sprintf(logFileStr, "/var/log/CIMAR/%s.log", &argString[0][i]);
                printf("main: Creating log: %s\n", logFileStr);
                logFile = fopen(logFileStr, "w");
                if(logFile != NULL)
                {
                        fprintf(logFile, "CIMAR %s Log -- %s\n", argString[0], timeString);
                        setLogFile(logFile);
                }
                else printf("main: ERROR: Could not create log file\n");
                // BUG: Else pError here
        }

        signal(SIGINT, signalInterventionHandler);

        cDebug(1, "main: Starting Up %s Node Software\n", mcGetName());
        if(mcStartup())
        {
                cError("main: %s Node Startup failed\n", mcGetName());
                cDebug(1, "main: Exiting %s Node Software\n", mcGetName());
                return 0;
        }

        if(verbose)
        {
                tcgetattr(0,&stored);
                memcpy(&newTermios,&stored,sizeof(struct termios));

                // Disable canonical mode, and set buffer size to 0 byte(s)
                newTermios.c_lflag &= (~ICANON);
                newTermios.c_lflag &= (~ECHO);
                newTermios.c_cc[VTIME] = 0;
                newTermios.c_cc[VMIN] = 0;
                tcsetattr(0,TCSANOW,&newTermios);
        }
        else
```

```
	{
		// Start up Curses window
		initscr();
		cbreak();
		noecho();
		nodelay(stdscr, 1);	// Don't wait at the getch() function if the user
hasn't hit a key
	}

	mainRunning = TRUE;

	while(mainRunning)
	{
		if(verbose)
		{
			choice = getc(stdin);
			switch(choice)
			{
				case 27: // Escape Key Pressed
					mainRunning = FALSE;
					break;
				default:
					usleep(1000);
					break;
			}
		}
		else
		{
			choice = getch(); // Get the key that the user has selected
			switch(choice)
			{
				case 27:
					mainRunning = FALSE;
					break;
				default:
					usleep(1000);
					if(choice >= '0' && choice <= '9') page = choice -
'0';
					break;
			}
			updateScreen(page);
		}

		usleep(25000);
	}
	if(verbose)
		tcsetattr(0,TCSANOW,&stored);
	else
	{
		// Stop Curses
		clear();
		endwin();
	}

	cDebug(1, "main: Shutting Down %s Node Software\n", mcGetName());
	mcShutdown();

	if(logFile != NULL)
	{
		fclose(logFile);
	}
	return 0;
}

const char *jausStateString(int jausState)
{
	static const char *jausStateString[7] =	{	"Initialize",

	"Ready",

	"Standby",
```

```
        "Shutdown",

        "Failure",

        "Emergency",

        "Unknown"
                                                                                };


        if(jausState < 0 || jausState > 5) return jausStateString[6];
        else return jausStateString[jausState];
}

void updateScreen(int page)
{
        int row = 0, column = 0;
        int pose;
        int numSoln, optSoln;
        double totalCost;
        jointEffort_t *effort;
        jointPosition_t *position;
        jointVelocity_t *velocity;
        jointForceTorque_t *forcetorque;
        manipulatorSpecifications_t *specs;
        jointMotion_t *motion;
        toolPoint_t *toolpoint;
        endEffectorPose_t *endeffectorpose;
        endEffectorPathMotion_t *endeffectorpathmotion;
        double *posePosition, *maxVelocity, *maxAcceleration, *maxDeceleration;
        double currentTime, poseTime;
        double *currentOrientation;
        double *currentEndEffectorPosition;
        double *currentEndEffectorOrientation;

        clear();

        switch(page)
        {
                case 0:
                        mvprintw(row++,column,"CIMAR %s Node Main Screen", mcGetName() );
                        row++;
                        mvprintw(row++,column,"Low Level Manipulator Control Components:");
                        row++;
                        mvprintw(row++,column," 1) Primitive Manipulator");
                        row++;
                        mvprintw(row++,column,"Manipulator Sensor Components:");
                        row++;
                        mvprintw(row++,column," 2) Manipulator Joint Position and Velocity
Sensors");
                        mvprintw(row++,column," 3) Manipulator Joint Force/Torque Sensor");
                        row++;
                        mvprintw(row++,column,"Low Level Position and Velocity Driver
Components:");
                        row++;
                        mvprintw(row++,column," 4) Manipulator Joint Positions Driver");
                        mvprintw(row++,column," 5) Manipulator End-Effector Pose Driver");
                        mvprintw(row++,column," 6) Manipulator Joint Velocities Driver");
                        mvprintw(row++,column," 7) Manipulator End_Effector Velocity State
Driver");
                        row++;
                        mvprintw(row++,column,"Mid Level Position and Velocity Driver
Components:");
                        row++;
                        mvprintw(row++,column," 8) Manipulator Joint Move Driver");
                        mvprintw(row++,column," 9) Manipulator End-Effector Discrete Pose
Driver Component");
                        row++;
                        mvprintw(row++,column," Press Escape To Shutdown Anytime");
                        break;
```

```
               case 1:
                      column++;
                      mvprintw(1,column,"Primitive Manipulator Screen");
                      mvprintw(3,column,"PM State:         %s",
jausStateString(pmGetState()) );
                      mvprintw(4,column,"PM Instance ID:   %3.0u", pmGetInstanceId());
                      mvprintw(5,column,"PM Component ID:  %3.0u", pmGetCompId());
                      mvprintw(6,column,"PM Node ID:              %3.0u",
pmGetNodeId());
                      mvprintw(7,column,"PM Subsystem ID:  %3.0u", pmGetSubsystemId());
                      mvprintw(8,column,"PM Update Rate:   %5.2f", pmGetUpdateRate());
                      mvprintw(9,column,"Interface Update Rate:  %5.2f",
getInterfaceUpdateRate());
                      mvaddstr(10,column,"----------------------------------------------
--------------------------------");

                   mvaddstr(1,65,"Commanded Effort");
                   mvaddstr(3,65,"Joint 1:");
                   mvaddstr(4,65,"Joint 2:");
                   mvaddstr(5,65,"Joint 3:");
                   mvaddstr(6,65,"Joint 4:");
                   mvaddstr(7,65,"Joint 5:");
                   mvaddstr(8,65,"Joint 6:");

                      effort = pmSetJointEffort();
                   mvprintw(3,75,"% 6.2f",effort->jointEffort[0]);
                   mvprintw(4,75,"% 6.2f",effort->jointEffort[1]);
                   mvprintw(5,75,"% 6.2f",effort->jointEffort[2]);
                   mvprintw(6,75,"% 6.2f",effort->jointEffort[3]);
                   mvprintw(7,75,"% 6.2f",effort->jointEffort[4]);
                   mvprintw(8,75,"% 6.2f",effort->jointEffort[5]);

                   mvaddstr(11,column,"Manipulator Specifications");
                   mvaddstr(16,1,"Joint 1:");
                   mvaddstr(17,1,"Joint 2:");
                   mvaddstr(18,1,"Joint 3:");
                   mvaddstr(19,1,"Joint 4:");
                   mvaddstr(20,1,"Joint 5:");
                   mvaddstr(21,1,"Joint 6:");

                      specs = pmReportManipulatorSpecifications();
                      mvaddstr(13,9,"Type");
                   mvprintw(16,10,"% 1.0d",specs->jointSpecifications[0].jointType);
                   mvprintw(17,10,"% 1.0d",specs->jointSpecifications[1].jointType);
                   mvprintw(18,10,"% 1.0d",specs->jointSpecifications[2].jointType);
                   mvprintw(19,10,"% 1.0d",specs->jointSpecifications[3].jointType);
                   mvprintw(20,10,"% 1.0d",specs->jointSpecifications[4].jointType);
                   mvprintw(21,10,"% 1.0d",specs->jointNtype);

                      mvaddstr(13,15,"Lnk_L");
                      mvaddstr(14,16,"(mm)");
                   mvprintw(16,15,"% 4.0f",specs->jointSpecifications[0].linkLength);
                   mvprintw(17,15,"% 4.0f",specs->jointSpecifications[1].linkLength);
                   mvprintw(18,15,"% 4.0f",specs->jointSpecifications[2].linkLength);
                   mvprintw(19,15,"% 4.0f",specs->jointSpecifications[3].linkLength);
                   mvprintw(20,15,"% 4.0f",specs->jointSpecifications[4].linkLength);

                      mvaddstr(13,21,"Twist_Ang");
                      mvaddstr(14,23,"(deg)");
                   mvprintw(16,22,"% 6.1f",specs->jointSpecifications[0].twistAngle *
RAD_TO_DEG / 1000);
                   mvprintw(17,22,"% 6.1f",specs->jointSpecifications[1].twistAngle *
RAD_TO_DEG / 1000);
                   mvprintw(18,22,"% 6.1f",specs->jointSpecifications[2].twistAngle *
RAD_TO_DEG / 1000);
                   mvprintw(19,22,"% 6.1f",specs->jointSpecifications[3].twistAngle *
RAD_TO_DEG / 1000);
                   mvprintw(20,22,"% 6.1f",specs->jointSpecifications[4].twistAngle *
RAD_TO_DEG / 1000);
```

```
                    mvaddstr(13,31,"Offset");
                    mvaddstr(14,32,"(mm)");
                 mvprintw(16,31,"% 4.0f",specs-
>jointSpecifications[0].jointOffsetOrAngle);
                 mvprintw(17,31,"% 4.0f",specs-
>jointSpecifications[1].jointOffsetOrAngle);
                 mvprintw(18,31,"% 4.0f",specs-
>jointSpecifications[2].jointOffsetOrAngle);
                 mvprintw(19,31,"% 4.0f",specs-
>jointSpecifications[3].jointOffsetOrAngle);
                 mvprintw(20,31,"% 4.0f",specs-
>jointSpecifications[4].jointOffsetOrAngle);
                 mvprintw(21,31,"% 4.0f",specs->jointNoffsetOrAngle);

                    mvaddstr(13,38,"Pos_Limit");
                    mvaddstr(14,40,"(enc)");
                 mvprintw(16,39,"%7.0f",specs->jointSpecifications[0].jointMaximumValue
* J1_RAD_TO_ENC / 1000);
                 mvprintw(17,39,"%7.0f",specs->jointSpecifications[1].jointMaximumValue
* J2_RAD_TO_ENC / 1000);
                 mvprintw(18,39,"%7.0f",specs->jointSpecifications[2].jointMaximumValue
* J3_RAD_TO_ENC / 1000);
                 mvprintw(19,39,"%7.0f",specs->jointSpecifications[3].jointMaximumValue
* J4_RAD_TO_ENC / 1000);
                 mvprintw(20,39,"%7.0f",specs->jointSpecifications[4].jointMaximumValue
* J5_RAD_TO_ENC / 1000);
                 mvprintw(21,39,"%7.0f",specs->jointNmaximumValue * J6_RAD_TO_ENC /
1000);

                    mvaddstr(13,48,"Vel_Limit");
                    mvaddstr(14,48,"(enc/s)");
                 mvprintw(16,49,"%5.0f",specs-
>jointSpecifications[0].jointMaximumVelocity * J1_RAD_TO_ENC / 1000);
                 mvprintw(17,49,"%5.0f",specs-
>jointSpecifications[1].jointMaximumVelocity * J2_RAD_TO_ENC / 1000);
                 mvprintw(18,49,"%5.0f",specs-
>jointSpecifications[2].jointMaximumVelocity * J3_RAD_TO_ENC / 1000);
                 mvprintw(19,49,"%5.0f",specs-
>jointSpecifications[3].jointMaximumVelocity * J4_RAD_TO_ENC / 1000);
                 mvprintw(20,49,"%5.0f",specs-
>jointSpecifications[4].jointMaximumVelocity * J5_RAD_TO_ENC / 1000);
                 mvprintw(21,49,"%5.0f",specs->jointNmaximumVelocity * J6_RAD_TO_ENC /
1000);

                    mvaddstr(23,1,"Manipulator Type:");
                    mvaddstr(24,3,"PUMA 762 Serial 6DOF");
                    mvaddstr(26,1,"Controller Type:");
                    mvaddstr(27,6,"GALIL MC, DMC2100");

                 mvaddstr(23,27,"Manipulator Coordinate System:");
                 mvaddstr(24,30, "Position:");
                 mvaddstr(24,45, "Orientation:");
                 mvaddstr(25,30,"X");
                 mvaddstr(26,30,"Y");
                 mvaddstr(27,30,"Z");
                 mvaddstr(25,45,"D");
                 mvaddstr(26,45,"A");
                 mvaddstr(27,45,"B");
                 mvaddstr(28,45,"C");

                 mvprintw(25,32,"% 4.1f",specs->manipulatorCoordinateSystemX);
                 mvprintw(26,32,"% 4.1f",specs->manipulatorCoordinateSystemY);
                 mvprintw(27,32,"% 4.1f",specs->manipulatorCoordinateSystemZ);
                 mvprintw(25,47,"% 4.1f",specs->quaternionQcomponentD);
                 mvprintw(26,47,"% 4.1f",specs->quaternionQcomponentA);
                 mvprintw(27,47,"% 4.1f",specs->quaternionQcomponentB);
                 mvprintw(28,47,"% 4.1f",specs->quaternionQcomponentC);

                    mvaddstr(11,67,"Current Effort");
                 mvaddstr(13,65,"Joint 1:");
                 mvaddstr(14,65,"Joint 2:");
```

```
                    mvaddstr(15,65,"Joint 3:");
                    mvaddstr(16,65,"Joint 4:");
                    mvaddstr(17,65,"Joint 5:");
                    mvaddstr(18,65,"Joint 6:");

                        effort = pmReportJointEffort();
                    mvprintw(13,75,"% 6.2f",effort->jointEffort[0]);
                    mvprintw(14,75,"% 6.2f",effort->jointEffort[1]);
                    mvprintw(15,75,"% 6.2f",effort->jointEffort[2]);
                    mvprintw(16,75,"% 6.2f",effort->jointEffort[3]);
                    mvprintw(17,75,"% 6.2f",effort->jointEffort[4]);
                    mvprintw(18,75,"% 6.2f",effort->jointEffort[5]);

                    break;

                case 2:
                        column++;
                        mvprintw(1,column,"Manipulator Joint Position and Velocity Sensors
Screen:", mcGetName() );
                        mvprintw(3,column,"MJPS State:               %s",
jausStateString(mjpsGetState()) );
                        mvprintw(4,column,"MJPS Instance ID:  %3.0u", mjpsGetInstanceId());
                        mvprintw(5,column,"MJPS Component ID: %3.0u", mjpsGetCompId());
                        mvprintw(6,column,"MJPS Node ID:             %3.0u",
mjpsGetNodeId());
                        mvprintw(7,column,"MJPS Subsystem ID: %3.0u",
mjpsGetSubsystemId());
                        mvprintw(8,column,"MJPS Update Rate:  %5.2f", mjpsGetUpdateRate());

                        mvprintw(3,40,"MJVS State:           %s",
jausStateString(mjvsGetState()) );
                        mvprintw(4,40,"MJVS Instance ID:     %3.0u", mjvsGetInstanceId());
                        mvprintw(5,40,"MJVS Component ID:    %3.0u", mjvsGetCompId());
                        mvprintw(6,40,"MJVS Node ID:         %3.0u", mjvsGetNodeId());
                        mvprintw(7,40,"MJVS Subsystem ID:    %3.0u",
mjvsGetSubsystemId());
                        mvprintw(8,40,"MJVS Update Rate:     %5.2f", mjvsGetUpdateRate());

                        mvaddstr(10,column,"-----------------------------------------------
---------------------------------");

                    mvaddstr(11,column,"Current position in:");

                    mvaddstr(13,13,"Joint 1:");
                    mvaddstr(14,13,"Joint 2:");
                    mvaddstr(15,13,"Joint 3:");
                    mvaddstr(16,13,"Joint 4:");
                    mvaddstr(17,13,"Joint 5:");
                    mvaddstr(18,13,"Joint 6:");

                        position = mjpsReportJointPosition();
                        mvprintw(11,25,"radians");
                        mvprintw(13,25,"% 6.4f",position->jointPosition[0]);
                    mvprintw(14,25,"% 6.4f",position->jointPosition[1]);
                    mvprintw(15,25,"% 6.4f",position->jointPosition[2]);
                    mvprintw(16,25,"% 6.4f",position->jointPosition[3]);
                    mvprintw(17,25,"% 6.4f",position->jointPosition[4]);
                    mvprintw(18,25,"% 6.4f",position->jointPosition[5]);

                        mvprintw(11,40,"degrees");
                        mvprintw(13,40,"% 6.2f",position->jointPosition[0] * RAD_TO_DEG);
                    mvprintw(14,40,"% 6.2f",position->jointPosition[1] * RAD_TO_DEG);
                    mvprintw(15,40,"% 6.2f",position->jointPosition[2] * RAD_TO_DEG);
                    mvprintw(16,40,"% 6.2f",position->jointPosition[3] * RAD_TO_DEG);
                    mvprintw(17,40,"% 6.2f",position->jointPosition[4] * RAD_TO_DEG);
                    mvprintw(18,40,"% 6.2f",position->jointPosition[5] * RAD_TO_DEG);


                        mvprintw(11,55,"enc_cnts");
                        mvprintw(13,55,"% 6.0f",position->jointPosition[0] *
J1_RAD_TO_ENC);
```

```
                    mvprintw(14,55,"% 6.0f",position->jointPosition[1] * J2_RAD_TO_ENC);
                    mvprintw(15,55,"% 6.0f",position->jointPosition[2] * J3_RAD_TO_ENC);
                    mvprintw(16,55,"% 6.0f",position->jointPosition[3] * J4_RAD_TO_ENC);
                    mvprintw(17,55,"% 6.0f",position->jointPosition[4] * J5_RAD_TO_ENC);
                    mvprintw(18,55,"% 6.0f",position->jointPosition[5] * J6_RAD_TO_ENC);


                    mvaddstr(20,column,"Current velocity in:");

                    mvaddstr(22,13,"Joint 1:");
                    mvaddstr(23,13,"Joint 2:");
                    mvaddstr(24,13,"Joint 3:");
                    mvaddstr(25,13,"Joint 4:");
                    mvaddstr(26,13,"Joint 5:");
                    mvaddstr(27,13,"Joint 6:");

                       velocity = mjvsReportJointVelocity();
                       mvprintw(20,25,"radians/s");
                       mvprintw(22,25,"% 6.4f",velocity->jointVelocity[0]);
                    mvprintw(23,25,"% 6.4f",velocity->jointVelocity[1]);
                    mvprintw(24,25,"% 6.4f",velocity->jointVelocity[2]);
                    mvprintw(25,25,"% 6.4f",velocity->jointVelocity[3]);
                    mvprintw(26,25,"% 6.4f",velocity->jointVelocity[4]);
                    mvprintw(27,25,"% 6.4f",velocity->jointVelocity[5]);

                       mvprintw(20,40,"degrees/s");
                       mvprintw(22,40,"% 6.2f",velocity->jointVelocity[0] * RAD_TO_DEG);
                    mvprintw(23,40,"% 6.2f",velocity->jointVelocity[1] * RAD_TO_DEG);
                    mvprintw(24,40,"% 6.2f",velocity->jointVelocity[2] * RAD_TO_DEG);
                    mvprintw(25,40,"% 6.2f",velocity->jointVelocity[3] * RAD_TO_DEG);
                    mvprintw(26,40,"% 6.2f",velocity->jointVelocity[4] * RAD_TO_DEG);
                    mvprintw(27,40,"% 6.2f",velocity->jointVelocity[5] * RAD_TO_DEG);


                       mvprintw(20,55,"enc_cnts/s");
                       mvprintw(22,55,"% 6.0f",velocity->jointVelocity[0] *
J1_RAD_TO_ENC);
                    mvprintw(23,55,"% 6.0f",velocity->jointVelocity[1] * J2_RAD_TO_ENC);
                    mvprintw(24,55,"% 6.0f",velocity->jointVelocity[2] * J3_RAD_TO_ENC);
                    mvprintw(25,55,"% 6.0f",velocity->jointVelocity[3] * J4_RAD_TO_ENC);
                    mvprintw(26,55,"% 6.0f",velocity->jointVelocity[4] * J5_RAD_TO_ENC);
                    mvprintw(27,55,"% 6.0f",velocity->jointVelocity[5] * J6_RAD_TO_ENC);


                    break;
                case 3:
                       column++;
                       row++;
                       mvprintw(row++,column,"Manipulator Joint Force/Torque Sensor
Screen", mcGetName() );
                       row++;
                       mvprintw(row++,column,"MJFTS State:          %s",
jausStateString(mjftsGetState()) );
                       mvprintw(row++,column,"MJFTS Instance ID:    %3.0u",
mjftsGetInstanceId());
                       mvprintw(row++,column,"MJFTS Component ID:   %3.0u",
mjftsGetCompId());
                       mvprintw(row++,column,"MJFTS Node ID:        %3.0u",
mjftsGetNodeId());
                       mvprintw(row++,column,"MJFTS Subsystem ID:   %3.0u",
mjftsGetSubsystemId());
                       mvprintw(row++,column,"MJFTS Update Rate:    %5.2f",
mjftsGetUpdateRate());
                       row++;
                       mvaddstr(row++,column,"----------------------------------------
-----------------------------------");
                    mvaddstr(row++,column,"Current torque in:");
                       row++;
                    mvaddstr(13,11,"Joint 1:");
                    mvaddstr(14,11,"Joint 2:");
                    mvaddstr(15,11,"Joint 3:");
```

```
                    mvaddstr(16,11,"Joint 4:");
                    mvaddstr(17,11,"Joint 5:");
                    mvaddstr(18,11,"Joint 6:");

                       forcetorque = mjftsReportJointForceTorque();
                       mvprintw(11,25,"N-m");
                       mvprintw(13,23,"% 6.4f",forcetorque->jointForceTorque[0]);
                    mvprintw(14,23,"% 6.4f",forcetorque->jointForceTorque[1]);
                    mvprintw(15,23,"% 6.4f",forcetorque->jointForceTorque[2]);
                    mvprintw(16,23,"% 6.4f",forcetorque->jointForceTorque[3]);
                    mvprintw(17,23,"% 6.4f",forcetorque->jointForceTorque[4]);
                    mvprintw(18,23,"% 6.4f",forcetorque->jointForceTorque[5]);


                       mvprintw(11,38,"ft_lbs");
                       mvprintw(13,38,"% 6.2f",forcetorque-
>jointForceTorque[0]/FTLB_TO_NM);
                    mvprintw(14,38,"% 6.2f",forcetorque->jointForceTorque[1]/FTLB_TO_NM);
                    mvprintw(15,38,"% 6.2f",forcetorque->jointForceTorque[2]/FTLB_TO_NM);
                    mvprintw(16,38,"% 6.2f",forcetorque->jointForceTorque[3]/FTLB_TO_NM);
                    mvprintw(17,38,"% 6.2f",forcetorque->jointForceTorque[4]/FTLB_TO_NM);
                    mvprintw(18,38,"% 6.2f",forcetorque->jointForceTorque[5]/FTLB_TO_NM);


                    break;

                case 4:
                       column++;
                       mvprintw(1,column,"Manipulator Joint Positions Driver Screen",
mcGetName() );
                       mvprintw(3,column,"MJPD State:               %s",
jausStateString(mjpdGetState()) );
                       mvprintw(4,column,"MJPD Instance ID:  %3.0u", mjpdGetInstanceId());
                       mvprintw(5,column,"MJPD Component ID: %3.0u", mjpdGetCompId());
                       mvprintw(6,column,"MJPD Node ID:             %3.0u",
mjpdGetNodeId());
                       mvprintw(7,column,"MJPD Subsystem ID: %3.0u",
mjpdGetSubsystemId());
                       mvprintw(8,column,"MJPD Update Rate:  %5.2f", mjpdGetUpdateRate());
                       mvaddstr(10,column,"-----------------------------------------------
---------------------------------");

                    mvaddstr(1,63,"Commanded Position");
                    mvaddstr(3,64,"Joint 1:");
                    mvaddstr(4,64,"Joint 2:");
                    mvaddstr(5,64,"Joint 3:");
                    mvaddstr(6,64,"Joint 4:");
                    mvaddstr(7,64,"Joint 5:");
                    mvaddstr(8,64,"Joint 6:");

                       position = mjpdSetJointPosition();
                       mvprintw(3,75,"% 6.0f",position->jointPosition[0] * J1_RAD_TO_ENC);
                    mvprintw(4,75,"% 6.0f",position->jointPosition[1] * J2_RAD_TO_ENC);
                    mvprintw(5,75,"% 6.0f",position->jointPosition[2] * J3_RAD_TO_ENC);
                    mvprintw(6,75,"% 6.0f",position->jointPosition[3] * J4_RAD_TO_ENC);
                    mvprintw(7,75,"% 6.0f",position->jointPosition[4] * J5_RAD_TO_ENC);
                    mvprintw(8,75,"% 6.0f",position->jointPosition[5] * J6_RAD_TO_ENC);


                    mvaddstr(11,column,"Queried Information:");
                    mvaddstr(13,13,"Joint 1:");
                    mvaddstr(14,13,"Joint 2:");
                    mvaddstr(15,13,"Joint 3:");
                    mvaddstr(16,13,"Joint 4:");
                    mvaddstr(17,13,"Joint 5:");
                    mvaddstr(18,13,"Joint 6:");

                       effort = mjpdReportJointEffort();
                       mvaddstr(11,26,"effort");
                    mvprintw(13,24,"% 7.2f",effort->jointEffort[0]);
                    mvprintw(14,24,"% 7.2f",effort->jointEffort[1]);
```

```
                    mvprintw(15,24,"% 7.2f",effort->jointEffort[2]);
                    mvprintw(16,24,"% 7.2f",effort->jointEffort[3]);
                    mvprintw(17,24,"% 7.2f",effort->jointEffort[4]);
                    mvprintw(18,24,"% 7.2f",effort->jointEffort[5]);

                       position = mjpdReportJointPosition();
                       mvaddstr(11,35,"position");
                       mvprintw(13,36,"% 6.0f",position->jointPosition[0] *
J1_RAD_TO_ENC);
                    mvprintw(14,36,"% 6.0f",position->jointPosition[1] * J2_RAD_TO_ENC);
                    mvprintw(15,36,"% 6.0f",position->jointPosition[2] * J3_RAD_TO_ENC);
                    mvprintw(16,36,"% 6.0f",position->jointPosition[3] * J4_RAD_TO_ENC);
                    mvprintw(17,36,"% 6.0f",position->jointPosition[4] * J5_RAD_TO_ENC);
                    mvprintw(18,36,"% 6.0f",position->jointPosition[5] * J6_RAD_TO_ENC);


                    mvaddstr(11,65,"Resulting Effort");
                    mvaddstr(13,64,"Joint 1:");
                    mvaddstr(14,64,"Joint 2:");
                    mvaddstr(15,64,"Joint 3:");
                    mvaddstr(16,64,"Joint 4:");
                    mvaddstr(17,64,"Joint 5:");
                    mvaddstr(18,64,"Joint 6:");

                       effort = mjpdSetJointEffort();
                    mvprintw(13,73,"% 8.2f",effort->jointEffort[0]);
                    mvprintw(14,73,"% 8.2f",effort->jointEffort[1]);
                    mvprintw(15,73,"% 8.2f",effort->jointEffort[2]);
                    mvprintw(16,73,"% 8.2f",effort->jointEffort[3]);
                    mvprintw(17,73,"% 8.2f",effort->jointEffort[4]);
                    mvprintw(18,73,"% 8.2f",effort->jointEffort[5]);

                    break;

                case 5:
                       column++;
                       mvprintw(1,column,"Manipulator End-Effector Pose Driver Screen",
mcGetName() );
                       mvprintw(3,column,"MEEPD State:              %s",
jausStateString(meepdGetState()) );
                       mvprintw(4,column,"MEEPD Instance ID: %3.0u",
meepdGetInstanceId());
                       mvprintw(5,column,"MEEPD Component ID:       %3.0u",
meepdGetCompId());
                       mvprintw(6,column,"MEEPD Node ID:            %3.0u",
meepdGetNodeId());
                       mvprintw(7,column,"MEEPD Subsystem ID:       %3.0u",
meepdGetSubsystemId());
                       mvprintw(8,column,"MEEPD Update Rate: %5.2f",
meepdGetUpdateRate());
                       mvaddstr(10,column,"-----------------------------------------------
---------------------------------");

                    mvaddstr(1,56,"Commanded Pose:");
                    mvaddstr(3,56,"position");
                    mvaddstr(5,56,"X:");
                    mvaddstr(6,56,"Y:");
                    mvaddstr(7,56,"Z:");

                       mvaddstr(3,70,"orientation");
                       mvaddstr(5,70,"D:");
                       mvaddstr(6,70,"A:");
                       mvaddstr(7,70,"B:");
                       mvaddstr(8,70,"C:");

                       endeffectorpose = meepdSetEndEffectorPose();
                       mvprintw(5,59,"% 6.2f",endeffectorpose->X*1000);
                    mvprintw(6,59,"% 6.2f",endeffectorpose->Y*1000);
                    mvprintw(7,59,"% 6.2f",endeffectorpose->Z*1000);

                       mvprintw(5,74,"% 6.4f",endeffectorpose->quaternionQcomponentD);
```

```
                    mvprintw(6,74,"% 6.4f",endeffectorpose->quaternionQcomponentA);
                    mvprintw(7,74,"% 6.4f",endeffectorpose->quaternionQcomponentB);
                    mvprintw(8,74,"% 6.4f",endeffectorpose->quaternionQcomponentC);

                    mvaddstr(11,column,"Queried Information:");
                    mvaddstr(13,13,"Joint 1:");
                    mvaddstr(14,13,"Joint 2:");
                    mvaddstr(15,13,"Joint 3:");
                    mvaddstr(16,13,"Joint 4:");
                    mvaddstr(17,13,"Joint 5:");
                    mvaddstr(18,13,"Joint 6:");

                       effort = meepdReportJointEffort();
                       mvaddstr(11,26,"effort");
                    mvprintw(13,24,"% 7.2f",effort->jointEffort[0]);
                    mvprintw(14,24,"% 7.2f",effort->jointEffort[1]);
                    mvprintw(15,24,"% 7.2f",effort->jointEffort[2]);
                    mvprintw(16,24,"% 7.2f",effort->jointEffort[3]);
                    mvprintw(17,24,"% 7.2f",effort->jointEffort[4]);
                    mvprintw(18,24,"% 7.2f",effort->jointEffort[5]);

                       position = meepdReportJointPosition();
                       mvaddstr(11,35,"position");
                       mvprintw(13,35,"% 7.0f",position->jointPosition[0] *
J1_RAD_TO_ENC);
                    mvprintw(14,35,"% 7.0f",position->jointPosition[1] * J2_RAD_TO_ENC);
                    mvprintw(15,35,"% 7.0f",position->jointPosition[2] * J3_RAD_TO_ENC);
                    mvprintw(16,35,"% 7.0f",position->jointPosition[3] * J4_RAD_TO_ENC);
                    mvprintw(17,35,"% 7.0f",position->jointPosition[4] * J5_RAD_TO_ENC);
                    mvprintw(18,35,"% 7.0f",position->jointPosition[5] * J6_RAD_TO_ENC);


                       toolpoint = meepdReportToolPoint();
                       mvaddstr(20,27,"pose");

                    mvaddstr(22,19,"X:");
                    mvaddstr(23,19,"Y:");
                    mvaddstr(24,19,"Z:");

                       mvaddstr(22,33,"D:");
                       mvaddstr(23,33,"A:");
                       mvaddstr(24,33,"B:");
                       mvaddstr(25,33,"C:");

                       mvprintw(22,24,"% 4.2f",toolpoint->X);
                       mvprintw(23,24,"% 4.2f",toolpoint->Y);
                       mvprintw(24,24,"% 4.2f",toolpoint->Z);

                       currentOrientation = meepdGetCurrentOrientation();
                       mvprintw(22,35,"% 6.4f",currentOrientation[0]);
                       mvprintw(23,35,"% 6.4f",currentOrientation[1]);
                       mvprintw(24,35,"% 6.4f",currentOrientation[2]);
                       mvprintw(25,35,"% 6.4f",currentOrientation[3]);

                       numSoln = cppGetNumSoln();
                       mvaddstr(22,55,"Number of solutions:");
                       mvprintw(22,79,"% 1.0d",numSoln);

                       optSoln = cppGetOptSoln();
                       mvaddstr(24,55,"Optimal solution is:");
                       mvprintw(24,79," %1.0d",optSoln);

                       totalCost = cppGetCost();
                       mvaddstr(26,55,"Current cost:");
                       mvprintw(26,73," %7.0f",totalCost);

                    mvaddstr(11,65,"Resulting Effort");
                    mvaddstr(13,64,"Joint 1:");
                    mvaddstr(14,64,"Joint 2:");
                    mvaddstr(15,64,"Joint 3:");
                    mvaddstr(16,64,"Joint 4:");
```

```
                    mvaddstr(17,64,"Joint 5:");
                    mvaddstr(18,64,"Joint 6:");

                       effort = meepdSetJointEffort();
                    mvprintw(13,73,"% 8.2f",effort->jointEffort[0]);
                    mvprintw(14,73,"% 8.2f",effort->jointEffort[1]);
                    mvprintw(15,73,"% 8.2f",effort->jointEffort[2]);
                    mvprintw(16,73,"% 8.2f",effort->jointEffort[3]);
                    mvprintw(17,73,"% 8.2f",effort->jointEffort[4]);
                    mvprintw(18,73,"% 8.2f",effort->jointEffort[5]);

                    break;



                case 6:
                       column++;
                       mvprintw(1,column,"Manipulator Joint Velocities Driver Screen",
mcGetName() );
                       mvprintw(3,column,"MJVD State:               %s",
jausStateString(mjvdGetState()) );
                       mvprintw(4,column,"MJVD Instance ID:  %3.0u", mjvdGetInstanceId());
                       mvprintw(5,column,"MJVD Component ID: %3.0u", mjvdGetCompId());
                       mvprintw(6,column,"MJVD Node ID:             %3.0u",
mjvdGetNodeId());
                       mvprintw(7,column,"MJVD Subsystem ID: %3.0u",
mjvdGetSubsystemId());
                       mvprintw(8,column,"MJVD Update Rate:  %5.2f", mjvdGetUpdateRate());
                       mvaddstr(10,column,"-----------------------------------------------
---------------------------------");

                    mvaddstr(1,63,"Commanded Velocity");
                    mvaddstr(3,64,"Joint 1:");
                    mvaddstr(4,64,"Joint 2:");
                    mvaddstr(5,64,"Joint 3:");
                    mvaddstr(6,64,"Joint 4:");
                    mvaddstr(7,64,"Joint 5:");
                    mvaddstr(8,64,"Joint 6:");

                       velocity = mjvdSetJointVelocity();
                       mvprintw(3,75,"% 6.0f",velocity->jointVelocity[0] * J1_RAD_TO_ENC);
                    mvprintw(4,75,"% 6.0f",velocity->jointVelocity[1] * J2_RAD_TO_ENC);
                    mvprintw(5,75,"% 6.0f",velocity->jointVelocity[2] * J3_RAD_TO_ENC);
                    mvprintw(6,75,"% 6.0f",velocity->jointVelocity[3] * J4_RAD_TO_ENC);
                    mvprintw(7,75,"% 6.0f",velocity->jointVelocity[4] * J5_RAD_TO_ENC);
                    mvprintw(8,75,"% 6.0f",velocity->jointVelocity[5] * J6_RAD_TO_ENC);


                    mvaddstr(11,column,"Queried Information:");
                    mvaddstr(13,13,"Joint 1:");
                    mvaddstr(14,13,"Joint 2:");
                    mvaddstr(15,13,"Joint 3:");
                    mvaddstr(16,13,"Joint 4:");
                    mvaddstr(17,13,"Joint 5:");
                    mvaddstr(18,13,"Joint 6:");

                       effort = mjvdReportJointEffort();
                       mvaddstr(11,26,"effort");
                    mvprintw(13,24,"% 7.2f",effort->jointEffort[0]);
                    mvprintw(14,24,"% 7.2f",effort->jointEffort[1]);
                    mvprintw(15,24,"% 7.2f",effort->jointEffort[2]);
                    mvprintw(16,24,"% 7.2f",effort->jointEffort[3]);
                    mvprintw(17,24,"% 7.2f",effort->jointEffort[4]);
                    mvprintw(18,24,"% 7.2f",effort->jointEffort[5]);

                       velocity = mjvdReportJointVelocity();
                       mvaddstr(11,35,"velocity");
                       mvprintw(13,36,"% 6.0f",velocity->jointVelocity[0] *
J1_RAD_TO_ENC);
                    mvprintw(14,36,"% 6.0f",velocity->jointVelocity[1] * J2_RAD_TO_ENC);
                    mvprintw(15,36,"% 6.0f",velocity->jointVelocity[2] * J3_RAD_TO_ENC);
```

```
                    mvprintw(16,36,"% 6.0f",velocity->jointVelocity[3] * J4_RAD_TO_ENC);
                    mvprintw(17,36,"% 6.0f",velocity->jointVelocity[4] * J5_RAD_TO_ENC);
                    mvprintw(18,36,"% 6.0f",velocity->jointVelocity[5] * J6_RAD_TO_ENC);


                    mvaddstr(11,65,"Resulting Effort");
                    mvaddstr(13,64,"Joint 1:");
                    mvaddstr(14,64,"Joint 2:");
                    mvaddstr(15,64,"Joint 3:");
                    mvaddstr(16,64,"Joint 4:");
                    mvaddstr(17,64,"Joint 5:");
                    mvaddstr(18,64,"Joint 6:");

                       effort = mjvdSetJointEffort();
                    mvprintw(13,75,"% 6.2f",effort->jointEffort[0]);
                    mvprintw(14,75,"% 6.2f",effort->jointEffort[1]);
                    mvprintw(15,75,"% 6.2f",effort->jointEffort[2]);
                    mvprintw(16,75,"% 6.2f",effort->jointEffort[3]);
                    mvprintw(17,75,"% 6.2f",effort->jointEffort[4]);
                    mvprintw(18,75,"% 6.2f",effort->jointEffort[5]);

                    break;

                case 8:
                       column++;
                       mvprintw(1,column,"Manipulator Joint Move Driver Screen",
mcGetName() );
                       mvprintw(3,column,"MJMD State:               %s",
jausStateString(mjmdGetState()) );
                       mvprintw(4,column,"MJMD Instance ID:  %3.0u", mjmdGetInstanceId());
                       mvprintw(5,column,"MJMD Component ID: %3.0u", mjmdGetCompId());
                       mvprintw(6,column,"MJMD Node ID:             %3.0u",
mjmdGetNodeId());
                       mvprintw(7,column,"MJMD Subsystem ID: %3.0u",
mjmdGetSubsystemId());
                       mvprintw(8,column,"MJMD Update Rate:  %5.2f", mjmdGetUpdateRate());
                       mvaddstr(10,column,"----------------------------------------------
--------------------------------");

                    mvaddstr(1,65,"Commanded Motion");
                    mvaddstr(3,64,"Joint 1:");
                    mvaddstr(4,64,"Joint 2:");
                    mvaddstr(5,64,"Joint 3:");
                    mvaddstr(6,64,"Joint 4:");
                    mvaddstr(7,64,"Joint 5:");
                    mvaddstr(8,64,"Joint 6:");

                       posePosition = mjmdGetCurrentPosePosition();
                       mvprintw(3,75,"% 6.0f",posePosition[0] * J1_RAD_TO_ENC);
                       mvprintw(4,75,"% 6.0f",posePosition[1] * J2_RAD_TO_ENC);
                       mvprintw(5,75,"% 6.0f",posePosition[2] * J3_RAD_TO_ENC);
                       mvprintw(6,75,"% 6.0f",posePosition[3] * J4_RAD_TO_ENC);
                       mvprintw(7,75,"% 6.0f",posePosition[4] * J5_RAD_TO_ENC);
                       mvprintw(8,75,"% 6.0f",posePosition[5] * J6_RAD_TO_ENC);


                    mvaddstr(11,column,"Commanded Joint Values:");
                    mvaddstr(13,13,"Joint 1:");
                    mvaddstr(14,13,"Joint 2:");
                    mvaddstr(15,13,"Joint 3:");
                    mvaddstr(16,13,"Joint 4:");
                    mvaddstr(17,13,"Joint 5:");
                    mvaddstr(18,13,"Joint 6:");

                       maxVelocity = mjmdGetCurrentPoseMaxVelocity();
                       mvaddstr(11,26,"max_vel");
                       mvprintw(13,24,"% 7.0f",maxVelocity[0] / JOINT1_ENC_TO_DEG /
DEG_TO_RAD);
                    mvprintw(14,24,"% 7.0f",maxVelocity[1] * J2_RAD_TO_ENC);
                    mvprintw(15,24,"% 7.0f",maxVelocity[2] * J3_RAD_TO_ENC);
                    mvprintw(16,24,"% 7.0f",maxVelocity[3] * J4_RAD_TO_ENC);
```

```
                    mvprintw(17,24,"% 7.0f",maxVelocity[4] * J5_RAD_TO_ENC);
                    mvprintw(18,24,"% 7.0f",maxVelocity[5] * J6_RAD_TO_ENC);

                       maxAcceleration = mjmdGetCurrentPoseMaxAcceleration();
                       mvaddstr(11,36,"max_acc");
                       mvprintw(13,36,"% 7.0f",maxAcceleration[0] * J1_RAD_TO_ENC);
                    mvprintw(14,36,"% 7.0f",maxAcceleration[1] * J2_RAD_TO_ENC);
                    mvprintw(15,36,"% 7.0f",maxAcceleration[2] * J3_RAD_TO_ENC);
                    mvprintw(16,36,"% 7.0f",maxAcceleration[3] * J4_RAD_TO_ENC);
                    mvprintw(17,36,"% 7.0f",maxAcceleration[4] * J5_RAD_TO_ENC);
                    mvprintw(18,36,"% 7.0f",maxAcceleration[5] * J6_RAD_TO_ENC);

                       maxDeceleration = mjmdGetCurrentPoseMaxDeceleration();
                       mvaddstr(11,46,"max_dec");
                       mvprintw(13,46,"% 7.0f",maxDeceleration[0] * J1_RAD_TO_ENC);
                    mvprintw(14,46,"% 7.0f",maxDeceleration[1] * J2_RAD_TO_ENC);
                    mvprintw(15,46,"% 7.0f",maxDeceleration[2] * J3_RAD_TO_ENC);
                    mvprintw(16,46,"% 7.0f",maxDeceleration[3] * J4_RAD_TO_ENC);
                    mvprintw(17,46,"% 7.0f",maxDeceleration[4] * J5_RAD_TO_ENC);
                    mvprintw(18,46,"% 7.0f",maxDeceleration[5] * J6_RAD_TO_ENC);

                       pose = mjmdGetPose();
                       motion = mjmdSetJointMotion();
                       poseTime = mjmdGetPoseTime();
                       currentTime = mjmdGetCurrentTime();
                       mvaddstr(12,57,"Current pose:");
                       mvprintw(12,78,"%3.0d",pose);
                       mvaddstr(14,57,"Number of poses:");
                       mvprintw(14,78,"%3.0d",motion->numPoses);
                       mvaddstr(16,57,"Next pose time:");
                       mvprintw(16,78,"%3.0f",poseTime);
                       mvaddstr(18,57,"Relative time:  ");
                       mvprintw(18,75,"%6.2f",currentTime);

                    mvaddstr(20,column,"Queried Information:");
                    mvaddstr(22,13,"Joint 1:");
                    mvaddstr(23,13,"Joint 2:");
                    mvaddstr(24,13,"Joint 3:");
                    mvaddstr(25,13,"Joint 4:");
                    mvaddstr(26,13,"Joint 5:");
                    mvaddstr(27,13,"Joint 6:");

                       effort = mjmdReportJointEffort();
                       mvaddstr(20,26,"effort");
                    mvprintw(22,24,"% 7.2f",effort->jointEffort[0]);
                    mvprintw(23,24,"% 7.2f",effort->jointEffort[1]);
                    mvprintw(24,24,"% 7.2f",effort->jointEffort[2]);
                    mvprintw(25,24,"% 7.2f",effort->jointEffort[3]);
                    mvprintw(26,24,"% 7.2f",effort->jointEffort[4]);
                    mvprintw(27,24,"% 7.2f",effort->jointEffort[5]);

                       position = mjmdReportJointPosition();
                       mvaddstr(20,35,"position");
                       mvprintw(22,36,"% 6.0f",position->jointPosition[0] *
J1_RAD_TO_ENC);
                    mvprintw(23,36,"% 6.0f",position->jointPosition[1] * J2_RAD_TO_ENC);
                    mvprintw(24,36,"% 6.0f",position->jointPosition[2] * J3_RAD_TO_ENC);
                    mvprintw(25,36,"% 6.0f",position->jointPosition[3] * J4_RAD_TO_ENC);
                    mvprintw(26,36,"% 6.0f",position->jointPosition[4] * J5_RAD_TO_ENC);
                    mvprintw(27,36,"% 6.0f",position->jointPosition[5] * J6_RAD_TO_ENC);


                       velocity = mjmdReportJointVelocity();
                       mvaddstr(20,46,"velocity");
                       mvprintw(22,46,"% 6.0f",velocity->jointVelocity[0] *
J1_RAD_TO_ENC);
                    mvprintw(23,46,"% 6.0f",velocity->jointVelocity[1] * J2_RAD_TO_ENC);
                    mvprintw(24,46,"% 6.0f",velocity->jointVelocity[2] * J3_RAD_TO_ENC);
                    mvprintw(25,46,"% 6.0f",velocity->jointVelocity[3] * J4_RAD_TO_ENC);
                    mvprintw(26,46,"% 6.0f",velocity->jointVelocity[4] * J5_RAD_TO_ENC);
                    mvprintw(27,46,"% 6.0f",velocity->jointVelocity[5] * J6_RAD_TO_ENC);
```

```
                    mvaddstr(20,65,"Resulting Effort");
                    mvaddstr(22,64,"Joint 1:");
                    mvaddstr(23,64,"Joint 2:");
                    mvaddstr(24,64,"Joint 3:");
                    mvaddstr(25,64,"Joint 4:");
                    mvaddstr(26,64,"Joint 5:");
                    mvaddstr(27,64,"Joint 6:");

                        effort = mjmdSetJointEffort();
                    mvprintw(22,73,"% 8.2f",effort->jointEffort[0]);
                    mvprintw(23,73,"% 8.2f",effort->jointEffort[1]);
                    mvprintw(24,73,"% 8.2f",effort->jointEffort[2]);
                    mvprintw(25,73,"% 8.2f",effort->jointEffort[3]);
                    mvprintw(26,73,"% 8.2f",effort->jointEffort[4]);
                    mvprintw(27,73,"% 8.2f",effort->jointEffort[5]);

                    break;

                case 9:
                        column++;
                        mvprintw(1,column,"Manipulator End-Effector DiscretePose Driver
Screen", mcGetName() );
                        mvprintw(3,column,"MEEPD State:              %s",
jausStateString(meedpdGetState()) );
                        mvprintw(4,column,"MEEPD Instance ID: %3.0u",
meedpdGetInstanceId());
                        mvprintw(5,column,"MEEPD Component ID:       %3.0u",
meedpdGetCompId());
                        mvprintw(6,column,"MEEPD Node ID:            %3.0u",
meedpdGetNodeId());
                        mvprintw(7,column,"MEEPD Subsystem ID:       %3.0u",
meedpdGetSubsystemId());
                        mvprintw(8,column,"MEEPD Update Rate: %5.2f",
meedpdGetUpdateRate());
                        mvaddstr(10,column,"----------------------------------------------
--------------------------------");

                    mvaddstr(1,56,"Commanded Pose:");
                    mvaddstr(3,56,"position");
                    mvaddstr(5,56,"X:");
                    mvaddstr(6,56,"Y:");
                    mvaddstr(7,56,"Z:");

                        mvaddstr(3,70,"orientation");
                        mvaddstr(5,70,"D:");
                        mvaddstr(6,70,"A:");
                        mvaddstr(7,70,"B:");
                        mvaddstr(8,70,"C:");

                        pose = meedpdGetPose();
                        endeffectorpathmotion = meedpdSetEndEffectorPathMotion();
                        poseTime = meedpdGetPoseTime();
                        currentTime = meedpdGetCurrentTime();
                        mvaddstr(22,55,"Current pose:");
                        mvprintw(22,78,"%3.0d",pose);
                        mvaddstr(23,55,"Number of poses:");
                        mvprintw(23,78,"%3.0d",endeffectorpathmotion->numPoses);
                        mvaddstr(24,55,"Next pose time:");
                        mvprintw(24,78,"%3.0f",poseTime);
                        mvaddstr(25,55,"Relative time:  ");
                        mvprintw(25,75,"%6.2f",currentTime);

                        currentEndEffectorPosition = meedpdGetCurrentEndEffectorPosition();
                        currentEndEffectorOrientation =
meedpdGetCurrentEndEffectorOrientation();
                        mvprintw(5,59,"% 6.2f",currentEndEffectorPosition[0]);
                    mvprintw(6,59,"% 6.2f",currentEndEffectorPosition[1]);
                    mvprintw(7,59,"% 6.2f",currentEndEffectorPosition[2]);

                        mvprintw(5,74,"% 6.4f",currentEndEffectorOrientation[0]);
```

```
                    mvprintw(6,74,"% 6.4f",currentEndEffectorOrientation[1]);
                    mvprintw(7,74,"% 6.4f",currentEndEffectorOrientation[2]);
                    mvprintw(8,74,"% 6.4f",currentEndEffectorOrientation[3]);

                    mvaddstr(11,column,"Queried Information:");
                    mvaddstr(13,13,"Joint 1:");
                    mvaddstr(14,13,"Joint 2:");
                    mvaddstr(15,13,"Joint 3:");
                    mvaddstr(16,13,"Joint 4:");
                    mvaddstr(17,13,"Joint 5:");
                    mvaddstr(18,13,"Joint 6:");

                       effort = meedpdReportJointEffort();
                       mvaddstr(11,26,"effort");
                    mvprintw(13,24,"% 7.2f",effort->jointEffort[0]);
                    mvprintw(14,24,"% 7.2f",effort->jointEffort[1]);
                    mvprintw(15,24,"% 7.2f",effort->jointEffort[2]);
                    mvprintw(16,24,"% 7.2f",effort->jointEffort[3]);
                    mvprintw(17,24,"% 7.2f",effort->jointEffort[4]);
                    mvprintw(18,24,"% 7.2f",effort->jointEffort[5]);

                       position = meedpdReportJointPosition();
                       mvaddstr(11,35,"position");
                       mvprintw(13,35,"% 7.0f",position->jointPosition[0] *
J1_RAD_TO_ENC);
                    mvprintw(14,35,"% 7.0f",position->jointPosition[1] * J2_RAD_TO_ENC);
                    mvprintw(15,35,"% 7.0f",position->jointPosition[2] * J3_RAD_TO_ENC);
                    mvprintw(16,35,"% 7.0f",position->jointPosition[3] * J4_RAD_TO_ENC);
                    mvprintw(17,35,"% 7.0f",position->jointPosition[4] * J5_RAD_TO_ENC);
                    mvprintw(18,35,"% 7.0f",position->jointPosition[5] * J6_RAD_TO_ENC);


                       toolpoint = meedpdReportToolPoint();
                       mvaddstr(20,27,"pose");

                       numSoln = cppGetNumSoln();
                       mvaddstr(26,55,"Number of solutions:");
                       mvprintw(26,79,"% 1.0d",numSoln);

                       optSoln = cppGetOptSoln();
                       mvaddstr(27,55,"Optimal solution is:");
                       mvprintw(27,79," %1.0d",optSoln);

                       totalCost = cppGetCost();
                       mvaddstr(28,55,"Current cost:");
                       mvprintw(28,73," %7.0f",totalCost);

                    mvaddstr(11,65,"Resulting Effort");
                    mvaddstr(13,64,"Joint 1:");
                    mvaddstr(14,64,"Joint 2:");
                    mvaddstr(15,64,"Joint 3:");
                    mvaddstr(16,64,"Joint 4:");
                    mvaddstr(17,64,"Joint 5:");
                    mvaddstr(18,64,"Joint 6:");

                       effort = meedpdSetJointEffort();
                    mvprintw(13,73,"% 8.2f",effort->jointEffort[0]);
                    mvprintw(14,73,"% 8.2f",effort->jointEffort[1]);
                    mvprintw(15,73,"% 8.2f",effort->jointEffort[2]);
                    mvprintw(16,73,"% 8.2f",effort->jointEffort[3]);
                    mvprintw(17,73,"% 8.2f",effort->jointEffort[4]);
                    mvprintw(18,73,"% 8.2f",effort->jointEffort[5]);

                    mvaddstr(22,19,"X:");
                    mvaddstr(23,19,"Y:");
                    mvaddstr(24,19,"Z:");

                       mvaddstr(22,33,"D:");
                       mvaddstr(23,33,"A:");
                       mvaddstr(24,33,"B:");
                       mvaddstr(25,33,"C:");
```

```
                        mvprintw(22,24,"% 4.2f",toolpoint->X);
                        mvprintw(23,24,"% 4.2f",toolpoint->Y);
                        mvprintw(24,24,"% 4.2f",toolpoint->Z);

                        currentOrientation = meedpdGetCurrentOrientation();
                        mvprintw(22,35,"% 6.4f",currentOrientation[0]);
                        mvprintw(23,35,"% 6.4f",currentOrientation[1]);
                        mvprintw(24,35,"% 6.4f",currentOrientation[2]);
                        mvprintw(25,35,"% 6.4f",currentOrientation[3]);

                        break;
        }

        move(25,0);
        refresh();
}

void signalInterventionHandler(int signal)
{
        mainRunning = FALSE;
}
```

APPENDIX C
SOURCE CODE FOR THE USER DEFINED JAUS MESSAGES

C.1 The JointEffort.c File and the Corresponding Header JointEffort.h

```
/*-----------------------------------------------------------------------*/
/* File        : jointEffort.c                                           */
/* Programmer  : Jeff Wit, Ralph English, Nate Mathews, Rommel Mandapat  */
/*             : Copyright (c) 2003 by WINTEC, Inc.                      */
/*-----------------------------------------------------------------------*/
/* Date        : 03/13/2003 original creation                            */
/*-----------------------------------------------------------------------*/
/* Description :                                                         */
/*-----------------------------------------------------------------------*/
/* Rev History :                                                         */
/*-----------------------------------------------------------------------*/
#include <string.h>
#include <math.h>
#include <jaus/jausNet.h>
#include <jaus/msg/jointEffort.h>

/*********************************************************************
  Function : convertJointEffort
  Input    : unsigned char*, jointEffort_t*, unsigned int, unsigned char
  Output   : int - number of bytes in buf
  Synopsis : This function formats the JAUS
             messages using the jointEffort_t structure.
*********************************************************************/
int convertJointEffort(unsigned char *buf, jointEffort_t *data,
               unsigned int size, unsigned char request)
{
       int i;
       short tempShort;
       unsigned short count=0;

       if (request == PACK) {
              if (size < (count+1)) return -1;
              buf[count] = data->numJoints;
              count++;

              for (i=0; ((i<data->numJoints) && (i<MAX_JOINTS)); i++) {
                     if (size < (count+2)) return -1;
                     tempShort = hdtojs(data->jointEffort[i],-100.0,100.0);
                     memcpy(&(buf[count]),&tempShort,2);
                     count += 2;
              }

              return count;
       }
       else if (request == UNPACK) {
              data->numJoints = buf[count];
              count++;

              for (i=0; ((i<data->numJoints) && (i<MAX_JOINTS)); i++) {
                     memcpy(&(tempShort),&(buf[count]),2);
                     data->jointEffort[i] = jstohd(tempShort,-100.0,100.0);
                     count += 2;
              }

              return count;
       }
       else
```

```
                return -1;
}

/*!     \file jointEffort.h
*       \author Jeff Wit
*       \author Ralph English
*       \author Nate Mathews
*       \author Rommel Mandapat
*       \par Copyright:
*       Copyright (c) 2003 by WINTEC, Inc.
*       \date 03/13/2003
*       \version 0.02
*/
#ifndef __jointEffort_h
#define __jointEffort_h

#include <jaus/msg/requests.h>
#include <jaus/msg/manipulatorSpecifications.h> /* defines MAX_JOINTS */

#ifdef __cplusplus
extern "C" {
#endif

/*! This message is used to set the joint effort for a serial mainipulator */
#define SET_JOINT_EFFORT             0x0601
/*! This message shall cause the receiving component to reply to the requester
* with the message \a REPORT_JOINT_EFFORT.
*/
#define QUERY_JOINT_EFFORT           0x2601
/*! This message provides the receiver the current values of the commanded
* joint efforts.
*/
#define REPORT_JOINT_EFFORT          0x4601

/*! \struct _jointEffort
* This structure contains the data associated with the command codes
* \a SET_JOINT_EFFORT and \a REPORT_JOINT_EFFORT.
*/
typedef struct _jointEffort
{
        unsigned char numJoints; /*!< 1 to MAX_JOINTS, 0 reserved */
        double jointEffort[MAX_JOINTS]; /*!< -100 to 100 percent */
} jointEffort_t;


/*!     \fn int convertJointEffort(unsigned char *buf, jointEffort_t *data, unsigned int
size, unsigned char request);
*       \brief Packs/Unpacks jointEffort_t to/from the format defined in JAUS Referenc
Architecture 3.0
*       \param buf The buffer that the JAUS data is unpacked from or packed into.
*       \param data The data structure that the JAUS data is packed from or unpacked into.
*       \param size The size of buf.
*       \param request The requested action, \a PACK or \a UNPACK.
*       \return The number of bytes in buf on success, -1 on error
*/
int convertJointEffort(unsigned char *buf, jointEffort_t *data, unsigned int size,
unsigned char request);

/*!     \fn     int printJointEffort(jointEffort_t *data);
*       \brief prints the structure jointEffort_t
*       \param data The data structure that is to be printed.
*       \return 0 on success, -1 on error
*/
int printJointEffort(jointEffort_t *data);

#ifdef __cplusplus
}
#endif

#endif
```

C.2 The JointPosition.c File and the Corresponding Header JointPosition.h

```
/*-----------------------------------------------------------------------*/
/* File        : jointPosition.c                                         */
/* Programmer  : Ognjen Sosa                                              */
/*             : Copyright (c) 2004 by University of Florida             */
/*-----------------------------------------------------------------------*/
/* Date        : 09/14/2004 original creation                            */
/*-----------------------------------------------------------------------*/
/* Description :                                                         */
/*-----------------------------------------------------------------------*/
/* Rev History :                                                         */
/*-----------------------------------------------------------------------*/
#include <string.h>
#include <math.h>
#include <jaus/jausNet.h>
#include <jaus/msg/jointPosition.h>
#include <jaus/msg/manipulatorSpecifications.h> //defines jointType

/*********************************************************************
  Function : convertJointPosition
  Input    : unsigned char*, jointPosition_t*, unsigned int, unsigned char
  Output   : int - number of bytes in buf
  Synopsis : This function formats the JAUS
             messages using the jointPosition_t structure.
*********************************************************************/
int convertJointPosition(unsigned char *buf, jointPosition_t *data,
               unsigned int size, unsigned char request)
{
       int i;
       int tempInt;
       unsigned short count=0;

       if (request == PACK) {
              if (size < (count+1)) return -1;
              buf[count] = data->numJoints;
              count++;

              for (i=0; ((i<data->numJoints) && (i<MAX_JOINTS)); i++) {
                     if (size < (count+4)) return -1;
//                   if (jointType[i] == 1)
                            tempInt = hdtoji(data->jointPosition[i],-25.1327,25.1327);
// for revolute, from -8_PI to 8_PI rad
//                   else
//                            tempInt = hdtojs(data->jointPosition[i],-10.0, 10.0); //
for prismatic, from -10m to 10m
                     memcpy(&(buf[count]),&tempInt,4);
                     count += 4;
              }

              return count;
       }
       else if (request == UNPACK) {
              data->numJoints = buf[count];
              count++;

              for (i=0; ((i<data->numJoints) && (i<MAX_JOINTS)); i++) {
                     memcpy(&(tempInt),&(buf[count]),4);
                     data->jointPosition[i] = jitohd(tempInt,-25.1327,25.1327);
                     count += 4;
              }

              return count;
       }
       else
              return -1;
}

/*!     \file jointPosition.h
```

```
*       \author Ognjen Sosa
*       \par Copyright:
*       Copyright (c) 2004 by University of Florida.
*       \date 09/14/2004
*       \version 0.02
*/
#ifndef __jointPosition_h
#define __jointPosition_h

#include <jaus/msg/requests.h>
#include <jaus/msg/manipulatorSpecifications.h> /* defines MAX_JOINTS */

#ifdef __cplusplus
extern "C" {
#endif

/*! This message is used to set the joint position for a serial mainipulator */
#define SET_JOINT_POSITION           0x0602

/*! This message shall cause the receiving component to reply to the requester
* with the message \a REPORT_JOINT_POSITION.
*/
#define QUERY_JOINT_POSITION  0x2602
/*! This message provides the receiver the current values of the set
* joint positions.
*/
#define REPORT_JOINT_POSITION        0x4602

/*! \struct _jointPosition
* This structure contains the data associated with the command codes
* \a SET_JOINT_POSITION and \a REPORT_JOINT_POSITION.
*/
typedef struct _jointPosition
{
       unsigned char numJoints; /*!< 1 to MAX_JOINTS, 0 reserved */
       double jointPosition[MAX_JOINTS]; /* -8_PIrad to 8_PIrad for revolute and -10m to
10m for prismatic */
} jointPosition_t;


/*!     \fn int convertJointPosition(unsigned char *buf, jointPosition_t *data, unsigned
int size, unsigned char request);
*       \brief Packs/Unpacks jointPosition_t to/from the format defined in JAUS Referenc
Architecture 3.0
*       \param buf The buffer that the JAUS data is unpacked from or packed into.
*       \param data The data structure that the JAUS data is packed from or unpacked into.
*       \param size The size of buf.
*       \param request The requested action, \a PACK or \a UNPACK.
*       \return The number of bytes in buf on success, -1 on error
*/
int convertJointPosition(unsigned char *buf, jointPosition_t *data, unsigned int size,
unsigned char request);

/*!     \fn     int printJointPosition(jointPosition_t *data);
*       \brief prints the structure jointPosition_t
*       \param data The data structure that is to be printed.
*       \return 0 on success, -1 on error
*/
int printJointPosition(jointPosition_t *data);

#ifdef __cplusplus
}
#endif

#endif
```

C.3 The EndEffectorPose.c File and the Corresponding Header EndEffectorPose.h

```
/*-----------------------------------------------------------------------*/
/* File        : endEffectorPose.c                                        */
/* Programmer  : Ognjen Sosa                                              */
/*             : Copyright (c) 2004 by University of Florida              */
/*-----------------------------------------------------------------------*/
/* Date        : 09/14/2004 original creation                             */
/*-----------------------------------------------------------------------*/
/* Description :                                                          */
/*-----------------------------------------------------------------------*/
/* Rev History :                                                          */
/*-----------------------------------------------------------------------*/
#include <string.h>
#include <math.h>
#include <jaus/jausNet.h>
#include <jaus/msg/endEffectorPose.h>

/*****************************************************************
  Function : convertEndEffectorPose
  Input    : unsigned char*, endEffectorPose_t*, unsigned int, unsigned char
  Output   : int - number of bytes in buf
  Synopsis : This function formats the JAUS
             messages using the endEffectorPose_t structure.
*****************************************************************/
int convertEndEffectorPose(unsigned char *buf, endEffectorPose_t *data,
               unsigned int size, unsigned char request)
{
       int tempInt;
       unsigned short count=0;

       if (request == PACK) {
              if (size < (count+4)) return -1;
              tempInt = hdtoji(data->X,-30.0,30.0); // -30m to 30m
              memcpy(&(buf[count]),&tempInt,4);
              count += 4;

              if (size < (count+4)) return -1;
              tempInt = hdtoji(data->Y,-30.0,30.0); // -30m to 30m
              memcpy(&(buf[count]),&tempInt,4);
              count += 4;

              if (size < (count+4)) return -1;
              tempInt = hdtoji(data->Z,-30.0,30.0); // -30m to 30m
              memcpy(&(buf[count]),&tempInt,4);
              count += 4;

              if (size < (count+4)) return -1;
              tempInt = hdtoji(data->quaternionQcomponentD,-1.0,1.0); // -1m to 1m
              memcpy(&(buf[count]),&tempInt,4);
              count += 4;

              if (size < (count+4)) return -1;
              tempInt = hdtoji(data->quaternionQcomponentA,-1.0,1.0); // -1m to 1m
              memcpy(&(buf[count]),&tempInt,4);
              count += 4;

              if (size < (count+4)) return -1;
              tempInt = hdtoji(data->quaternionQcomponentB,-1.0,1.0); // -1m to 1m
              memcpy(&(buf[count]),&tempInt,4);
              count += 4;

              if (size < (count+4)) return -1;
              tempInt = hdtoji(data->quaternionQcomponentC,-1.0,1.0); // -1m to 1m
              memcpy(&(buf[count]),&tempInt,4);
              count += 4;
```

```
                return count;
        }

        else if (request == UNPACK) {

                memcpy(&(tempInt),&(buf[count]),4);
                data->X = jitohd(tempInt,-30.0,30.0);
                count += 4;

                memcpy(&(tempInt),&(buf[count]),4);
                data->Y = jitohd(tempInt,-30.0,30.0);
                count += 4;

                memcpy(&(tempInt),&(buf[count]),4);
                data->Z = jitohd(tempInt,-30.0,30.0);
                count += 4;

                memcpy(&(tempInt),&(buf[count]),4);
                data->quaternionQcomponentD = jitohd(tempInt,-1.0,1.0);
                count += 4;

                memcpy(&(tempInt),&(buf[count]),4);
                data->quaternionQcomponentA = jitohd(tempInt,-1.0,1.0);
                count += 4;

                memcpy(&(tempInt),&(buf[count]),4);
                data->quaternionQcomponentB = jitohd(tempInt,-1.0,1.0);
                count += 4;

                memcpy(&(tempInt),&(buf[count]),4);
                data->quaternionQcomponentC = jitohd(tempInt,-1.0,1.0);
                count += 4;

                return count;
        }
        else
                return -1;
}

/*!     \file endEffectorPose.h
*       \author Ognjen Sosa
*       \par Copyright:
*       Copyright (c) 2004 by University of Florida.
*       \date 09/14/2004
*       \version 0.02
*/
#ifndef __endEffectorPose_h
#define __endEffectorPose_h

#include <jaus/msg/requests.h>

#ifdef __cplusplus
extern "C" {
#endif

/*! This message is used to set the end effector pose for a serial mainipulator */
#define SET_END_EFFECTOR_POSE        0x0605

/*! \struct _endEffectorPose
* This structure contains the data associated with the command code
* \a SET_END_EFFECTOR_POSE
*/
typedef struct _endEffectorPose
{
        double X; /* -30m to 30m */
        double Y; /* -30m to 30m */
        double Z; /* -30m to 30m */

        double quaternionQcomponentD; /* -1m to 1m */
        double quaternionQcomponentA; /* -1m to 1m */
        double quaternionQcomponentB; /* -1m to 1m */
```

```
        double quaternionQcomponentC; /* -1m to 1m */
} endEffectorPose_t;


/*!     \fn int convertEndEffectorPose(unsigned char *buf, endEffectorPose_t *data,
unsigned int size, unsigned char request);
*       \brief Packs/Unpacks endEffectorPose_t to/from the format defined in JAUS Referenc
Architecture 3.0
*       \param buf The buffer that the JAUS data is unpacked from or packed into.
*       \param data The data structure that the JAUS data is packed from or unpacked into.
*       \param size The size of buf.
*       \param request The requested action, \a PACK or \a UNPACK.
*       \return The number of bytes in buf on success, -1 on error
*/
int convertEndEffectorPose(unsigned char *buf, endEffectorPose_t *data, unsigned int
size, unsigned char request);

/*!     \fn     int printEndEffectorPose(endEffectorPose_t *data);
*       \brief prints the structure endEffectorPose_t
*       \param data The data structure that is to be printed.
*       \return 0 on success, -1 on error
*/
int printEndEffectorPose(endEffectorPose_t *data);

#ifdef __cplusplus
}
#endif

#endif
```

LIST OF REFERENCES

1. Das H., Bao X. Bar-Cohed Y., Bonitz R., Lindemann R., Maimone M., Nesnas I., Voorhees C., Robot Manipulator Technologies for Planetary Exploration, Proceedings of the 6th Annual International Symposium on Smart Structures and Materials, Newport Beach, CA, March 1999.

2. Matthies L., Stereo Vision for Planetary Rovers: Stochastic Modelling to Near Real-time Implementation, International Journal of Computer Vision, Volume 8, pp 71-91, July 1992.

3. Das H., Maimone M., Nesnas I., Autonomous Rock Tracking and Acquisition from a Mars Rover, Proceedings to the 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space, Noordwijk, Netherlands, June 1999.

4. Yuh J., Underwater Robotic Vehicles: Design and Control, TSI Press Series, Albuquerque, NM, 1995.

5. Sarkar N., Podder T., Motion Coordination of Underwater Vehicle-Manipulator Systems Subject to Drag Optimization, IEEE International Conference on Robotics and Automation, Detroit, MI, pp 387-392, May 1999.

6. Cui Y., Sarkar N., A Unified Force Control Approach to Autonomous Underwater Manipulation, Robotica, Cambridge University Press, pp 255-266, September 2000.

7. JAUS Working Group, 2004, Joint Architecture for Unmanned Systems (JAUS): Reference Architecture Specification, Version 3.2, Volume II, The Joint Architecture for Unmanned Systems, retrieved October 5, 2004, from http://www.jauswg.org, October 2004.

8. Vinch, P., Design and Implementation of an Intelligent Primitive Driver, M.S. Thesis, 2003.

9. Crane C., Duffy J., Kinematic Analysis of Robot Manipulators, Cambridge University Press, University of Florida, Gainesville, FL, 1998.

10. Duffy, J., Crane C., Screw Theory Based Analysis of Robot Manipulators, Center for Intelligent Machines and Robotics, University of Florida, Gainesville, FL, 2000.

11. Dowling, K., What's Available for Puma Manipulators?, retrieved October 13, 2004, from http://www.frc.ri.cmu.edu/robotics-faq/12.html, August, 1996.

12. JAUS Working Group, 2004, Joint Architecture for Unmanned Systems (JAUS): Compliance Specification, Version 1.0, The Joint Architecture for Unmanned Systems, retrieved October 6, 2004, from http://www.jauswg.org, July 2004.

13. Unimate Industrial Robots, PUMA 700 Series Mark III – VAL II Equipment Manual, Unimation Inc, Danbury, Connecticut, May 1986.

14. Galil MC, DMC-2x00 Manual Rev. 1.3, Galil Motion Control Inc., Rocklin, California, June 2001.

15. Galil MC, Linux Driver and C/C++ API for PCI/ISA/Ethernet Controllers, Galil Motion Control Inc., Rocklin, California, September 2004.

BIOGRAPHICAL SKETCH

Ognjen Sosa was born on February 5, 1979, in Sarajevo, Bosnia and Herzegovina, to Darko and Jadranka Sosa. He received his Bachelor of Science degree in mechanical engineering (graduating with honors) from the University of Florida in May 2001. He then enrolled in the Graduate School of the University of Florida, and obtained a Master of Science degree in management in June 2003; and a Master of Business Administration degree in May of 2004. After completing his Master of Engineering degree in mechanical engineering, he plans to work in the field that will use both technical and business skills acquired during his studies.