

To the Graduate Council:

I am submitting herewith a dissertation written by Mark William Noakes entitled "Telerobotic Sensor-based Tool Control Derived From Behavior-based Robotics Concepts." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Mechanical Engineering.

William R. Hamel , Major Professor

We have read this dissertation
and recommend its acceptance:

J. Wesley Hines

Lynne E. Parker

Gary V. Smith

Accepted for the Council:

Carolyn R. Hodges
Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

TELEROBOTIC SENSOR-BASED TOOL CONTROL DERIVED FROM
BEHAVIOR-BASED ROBOTICS CONCEPTS

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Mark William Noakes
May 2011

Copyright © 2011 by Mark William Noakes
All rights reserved.

Acknowledgements

I would like to thank William R. Hamel, my advisor for his patience and guidance through this too long process. I would like to thank my committee—J. Wesley Hines, Lynne E. Parker, and Gary V. Smith—for their direction in, review of, and input to my dissertation. I would like to thank fellow graduate student, Andrzej Nycz, for his input to my work and for his extensive ownership of the baseline system as “keeper of the high level controller” (HLC) that I used for my work. I would also like to thank Rob Gibbons for his valuable Linux counsel during my implementation of the new HLC. I am grateful to the Oak Ridge National Laboratory for their permission to use some of the photographs contained in this document. The Division of Work and Industry, National Museum of American History, Behring Center, Smithsonian Institution provided permission to use robot historical photographs. My management at Oak Ridge National Laboratory, François Pin and Ken Tobin, supported my decision to pursue school even though it wasn’t convenient to their needs. To my parents, I would like to say thank you for an upbringing that insisted on hard work and a refusal to give up. Finally I would like to thank my family—my wife Clemence and my twin nine year old sons Sean and David—for their support and patience in the pursuit of this degree even though it meant time not spent with them that can never be recovered and far too many things left undone for too long.

Abstract

Teleoperated task execution for hazardous environments is slow and requires highly skilled operators. Attempts to implement telerobotic assists to improve efficiency have been demonstrated in constrained laboratory environments but are not being used in the field because they are not appropriate for use on actual remote systems operating in complex unstructured environments using typical operators. This work describes a methodology for combining select concepts from behavior-based systems with telerobotic tool control in a way that is compatible with existing manipulator architectures used by remote systems typical to operations in hazardous environment. The purpose of the approach is to minimize the task instance modeling in favor of a priori task type models while using sensor information to register the task type model to the task instance. The concept was demonstrated for two tools useful to decontamination & dismantlement type operations—a reciprocating saw and a powered socket tool. The experimental results demonstrated that the approach works to facilitate traded control telerobotic tooling execution by enabling difficult tasks and by limiting tool damage. The role of the tools and tasks as drivers to the telerobotic implementation was better understood in the need for thorough task decomposition and the discovery and examination of the tool process signature. The contributions of this work include: (1) the exploration and evaluation of select features of behavior-based robotics to create a new methodology for integrating telerobotic tool control with positional teleoperation in the execution of complex tool-centric remote tasks, (2) the simplification of task decomposition and the implementation of sensor-based tool control in such a way that eliminates the need for the creation of a task instance model for telerobotic task execution, and (3) the discovery, demonstrated use, and documentation of characteristic tool process signatures that have general value in the investigation of other tool control, tool maintenance, and tool development strategies above and beyond the benefit sustained for the methodology described in this work.

Table of Contents

Chapter 1 Introduction.....	1
1.1 Motivation.....	1
1.2 Contributions.....	2
1.3 Outline of the Dissertation.....	4
Chapter 2 Background.....	5
2.1 Introduction.....	5
2.2 Teleoperation.....	5
2.3 Robotics	14
2.4 Telerobotics.....	15
2.5 Behavior-based Robotics (BBR).....	17
2.6 Application Areas.....	20
Chapter 3 Relevant Work.....	22
3.1 Introduction.....	22
3.2 Teleoperated Tooling Tasks.....	22
3.3 Telerobotic Tooling Tasks	24
3.4 BBR Tooling Tasks	32
3.5 Tool Disturbances.....	36
3.6 Summary.....	37
Chapter 4 Testbed Description, Capabilities, and Limitations.....	39
4.1 Introduction.....	39
4.2 Hardware Overview.....	39
4.3 Software Architecture and Implementation.....	42
4.4 HLC Interface.....	46
4.5 Tooling Interfaces.....	47
4.6 System Limitations.....	48
Chapter 5 Telerobotic Tool Control Methodology Derived From Behavior-based Concepts	49
5.1 Introduction.....	49
5.2 Identification of the Tool Set and Applicability of Technique	51
5.3 Behavior Selection Methods and Impact on Technique Development	55
5.4 Description of Methodology	57
5.5 Implementation Guidelines.....	63
5.6 Managing Human or Robot to Telerobotic Interaction	66
Chapter 6 Functional Implementation.....	67
6.1 Introduction.....	67
6.2 Cutting a Horizontal Pipe With a Reciprocating Saw	67
6.2.1 Task Definition.....	67
6.2.2 Tool Selection and Description	70
6.2.3 Subtask Definition	74
6.2.4 Sensor Selection	75
6.2.5 Saw Experimentation, Function Definition, and Implementation.....	76

6.2.6 Testing to Establish Saw Thresholds and Control Approaches.....	79
6.3 Removing a Bolt With a Powered Socket Tool	93
6.3.1 Task Definition.....	93
6.3.2 Tool Selection and Description	94
6.3.3 Subtask Definition	98
6.3.4 Sensor Selection	98
6.3.5 Socket Experimentation, Function Definition, and Implementation	99
6.3.6 Testing to Establish Socket Thresholds and Control Approaches.....	100
6.4 A Note on Expansion to Other Tools	102
Chapter 7 Experimental Results.....	105
7.1 Discussion of Overall Telerobotic Reciprocating Saw Results	105
7.2 Examination of Specific Saw Tool Representative Test Cases	109
7.3 Discussion of Overall Telerobotic Socket Tool Results.....	113
7.4 Examination of Specific Representative Socket Tool Test Cases.....	114
Chapter 8 Summary and Future Work	118
8.1 Summary	118
8.2 Review of Contributions	119
8.3 Future Work	121
Chapter 9 Conclusions.....	123
List of References	126
Appendices	134
Appendix A Software.....	135
Appendix B Mechanical Drawings	229
Appendix C Schematics	237
Vita.....	241

List of Tables

Table 1. Remote Systems Efficiencies.....	6
Table 2. D&D Tool Summary.	52
Table 3. Reciprocating Saw Specifications Summary.	72
Table 4. Reciprocating Saw Event Tabulation.	80
Table 5. Socket Tool (Drill) Specifications Summary.....	96
Table 6. Socket Tool Event Tabulation.....	100
Table 7. Reciprocating Saw Data.....	107
Table 8. bCut128S Internal Performance Data.....	108
Table 9. bApproachB Socket Tool Composite Results.....	115

List of Figures

Figure 1. Early Long Handle Tools.....	7
Figure 2. Early Mechanical Manipulator Prototype.....	8
Figure 3. Commercial Through-the-Wall Manipulators.	8
Figure 4. Commercial Power Manipulators Resemble Robots.	9
Figure 5. M-2 Servomanipulator.....	10
Figure 6. Telerob State-of-the-art Commercial Teleoperator.	11
Figure 7. Advanced Servomanipulator Remotely Maintainable Manipulator.	11
Figure 8. Advanced Integrated Maintenance System Master Control Station.	12
Figure 9. Dual Arm Work Platform Using Schilling Hydraulic Manipulators.	12
Figure 10. Schilling Smart Tooling Demonstration.....	13
Figure 11. Barrett Wraptor Mounted on Schilling Manipulator at UTK.	14
Figure 12. Unimate Robot.	16
Figure 13. Machina Speculatrix Cybernetic Tortoise Replica.....	18
Figure 14. Telerobotics Test Bed.....	41
Figure 15. PC/104 Manipulator Controller.....	41
Figure 16. Telerobotics Operator station.....	42
Figure 17. Test Bed System Level Block Diagram.	43
Figure 18. HLC Graphical Monitor.	47
Figure 19. Smart Tool Behavior Development Methodology Block Diagram.	58
Figure 20. Concept Block Diagram.	64
Figure 21. Behavior Selection Sequencing.....	65
Figure 22. Horizontal Pipe Task.	68
Figure 23. Real World Piping Arrays and Viewing Limitations.	68
Figure 24. Pipe End Section.	69
Figure 25. Hand Held Reciprocating Saw.....	71
Figure 26. Reciprocating Saw Smart Tool.	73
Figure 27. Reciprocating Saw Mounted in Gripper.....	73
Figure 28. Smart Tool Force-Torque Sensor Axes.....	76
Figure 29. Sample bApproachH Plot of Forces and Torques.....	82
Figure 30. Sample bBackH Plot of Forces and Torques.....	84
Figure 31. Sample bApproachV Plot of Forces and Torques.....	86
Figure 32. Sample bBackV Plot of Forces and Torques.....	87
Figure 33. Unfiltered Cut Forces and Torques.	90
Figure 34. Example 1 Filtered Ry.....	91
Figure 35. Example 2 Filtered Ry.....	91
Figure 36. Disassembly Mockup.	94
Figure 37. Electric Drill for Socket Tool.....	95
Figure 38. Smart Socket Tool.....	97
Figure 39. Smart Socket Tool Mounted in Gripper.	97
Figure 40. Cut Data From Shortest Duration Cut.....	111
Figure 41. Cut Data From Longest Duration Cut.....	112

Figure 42. Sample Approach Forces and Torques, Fx Used for Event Monitoring.	116
Figure 43. Sample Unbolt Forces and Torques, fxfilt Used for Event Monitoring.	117

Chapter 1

Introduction

1.1 Motivation

The US Department of Energy (DOE) has a stated need for improved remote systems technology that will assist in removing workers from hazardous environments while improving productivity [1], [2]. Due to current limitations of remotely operated systems and autonomous robotics, the vast majority of hazardous material operations is still performed by human workers dressed in protective equipment and sent into the hazardous environment to complete activities manually. One of the most pressing hazardous operations categories is the decontamination and decommissioning (D&D) of contaminated DOE nuclear facilities. Remote technology has been used successfully, but many D&D operation organizations have complained that the equipment available today is not sufficiently suited to their needs [1]. Remote systems as they now exist are too costly in terms of procurement, facility burden, and the requirement for skilled operators. Remote systems are also typically described as being too slow in task completion time and not capable of matching human dexterity. These same criticisms expressed by DOE operations organizations also apply to remote systems everywhere in use: space exploration, sub-sea exploration and oil rig maintenance and accident response, military explosive ordnance disposal, and homeland security, to name a few.

Remote equipment dismantlement is a common theme as a need in the D&D community. Contaminated process equipment and structural steel are common. Where possible, suited humans are used to complete unbolting and cutting tasks, but there have proved to be significant safety, health, and cost issues involved. Teleoperated remote systems have also been used where radiation levels eliminate the possibility of using humans; however system cost and task time completion are major issues in overall operating costs. A time-efficient, cost-effective approach to safely complete D&D operations without placing humans in the hazardous environment is a direct need. Telerobotic systems (teleoperated

remote systems that incorporate added automation to improve operational efficiency) are one solution.

This dissertation addresses the problem of tool control and uncompensated errors in teleoperated or robotic motion via the creation of a sensor-actuator control strategy by identifying and using select relevant concepts from classical behavior-based robotics (BBR) techniques to permit task execution in unstructured environments. The focus is not on the advancement of or a rigid adherence to BBR techniques but rather on the exploration of the “first principles” of behavior-based systems as a means to facilitate tool control for improved viability of telerobotic manipulation in unstructured environments from the perspective of the remote systems community. The research includes experimental data collection and verification of theoretical development for multiple tools for both human interactive and robotic task execution assists.

1.2 Contributions

The fundamental contributions of this dissertation are:

1. The exploration and evaluation of behavior-based robotics for concepts to create a new methodology for integrating telerobotic tool control with positional teleoperation in the execution of complex tool-centric remote tasks such as those associated with remote nuclear operations. Successful experimental results with selected power tools and a full-scale telerobotics test bed have revealed the attractive combination of simple implementation and efficient/effective tooling operations.

This methodology provides a workable clear path to implementation relevant to the existing architectures of typical teleoperator systems while addressing tasks that are currently difficult to automate due to complexity and limited registration to

actual task hardware. Once the first couple of tool tasks were programmed, it was quite obvious that this technique has created a set of primitives that may be assembled in different ways or with slight modification to quickly produce new automated tooling tasks. This work represents the first known application of these techniques to power tooling tasks.

2. The creation of a new tooling task modeling process that is general in nature and applicable to a wide range of power tools used in typical remote operations. This task type modeling can replace task instance modeling to reduce and simplify the application of the new behavior-based methods to complex telerobotic tooling applications. It was demonstrated that the task type model could be reliably encoded in a sequence of simple behavior-like reactive functions thereby alleviating the need for extensive a priori generation of a task instance model for each task execution. This reduces the modeling time needed for individual task automation making telerobotics more time competitive even with proficient operators.

3. The generation of specific characteristic tooling data for reciprocating saw cutting and removal of bolts with a powered socket tool. These results have general value in that they are relevant to extensions of this work and in the pursuit of other tool control strategies. In particular, the force profile generated for pipe cutting produces a well-defined characteristic signature that should be broadly useful even outside of the telerobotics community. Progressive variation in the tool signature profiles over repeated test instances indicate that tool wear, maintenance prediction, and fault detection can probably be deduced from further study of the process signature.

1.3 Outline of the Dissertation

The relevant definitions, history, and background of remote systems, teleoperation, robotics, telerobotics, and behavior-based systems are presented in Chapter 2 along with a remote systems perspective on applications. A survey of the relevant work is then presented in Chapter 3. Chapter 4 provides a discussion of the testbed description, capabilities, and limitations. Chapter 5 addresses the development of the methodology. Chapter 6 describes the functional implementation of the two example test cases. The experimental work is presented in Chapter 7. Chapter 8 outlines a summary of the work presented and provides a discussion of future work. Chapter 9 provides a final conclusion to the work. The appendices provide software, mechanical, and electrical/electronic background documentation.

Chapter 2

Background

2.1 Introduction

This chapter presents the definitions and relevant history of remote systems—teleoperation and robotics—from the vantage point of teleoperation. Since this dissertation is concerned with enhanced dexterous manipulation, only minimal attention as necessary will be given to the vast territory of mobile remote systems and robotics. Application areas will also be discussed to frame the context of the rest of the dissertation.

2.2 Teleoperation

Sheridan's definition of teleoperation states:

Teleoperation is the extension of a person's sensing and manipulation capability to a remote location. A teleoperator includes at the minimum artificial sensors, arms and hands, a vehicle for carrying these, and communications channels to and from the human operator. The term "teleoperation" refers most commonly to direct and continuous human control of the teleoperator, but can also be used generally to encompass "telerobotics"...as well [3].

For the purposes of this dissertation, high fidelity teleoperation will be further defined as teleoperated manipulation receiving operator commands from a positional master controller instead of from a high level supervisory control graphical operator interface or from rate control joysticks.

Remotely operated systems have an inherent inefficiency of operations due to the limited dexterity of the machine and the limited ability of the operator interface to support the sensory needs of the operator. Table 1 communicates these operator inefficiencies measured in task completion time ratios using varieties of remote systems compared to bare “hands-on” task completion for various remote equipment and operator interface configurations. High fidelity teleoperation is considered to be the best remote system currently in use; however there is still great disparity between the performance of a “good” teleoperator and human hands-on task execution.

Modern remote systems were developed out of the extreme needs of the World War II Manhattan Project’s radioactive materials handling. The technology developmental progression was from long handled tools to mechanical “master-slave” manipulators and switchbox-controlled electric manipulators (the direct ancestor of industrial robot manipulators) to analog servomanipulators and finally to digital servomanipulators.

Long-handled tools, such as is shown in Figure 1, have simple end-effectors and control handles along with limited capability. While long-handled tools are slow, have limited reach, and are not articulate enough for many tasks, they are still used today in some cases.

Table 1. Remote Systems Efficiencies.

(used by permission of the author) [4]

Manipulator Type	Task Completion Time Ratios
Skilled human operator (unencumbered)	1:1
Suited human (air suit or equal)	8:1
Force-reflecting servomanipulator or master/slave manipulator (i.e., through the wall type)	8:1
Non-force-reflecting electromechanical manipulator (i.e., power-arm type)	20:1 – 50:1
Crane/impact wrench	50:1 – 500:1

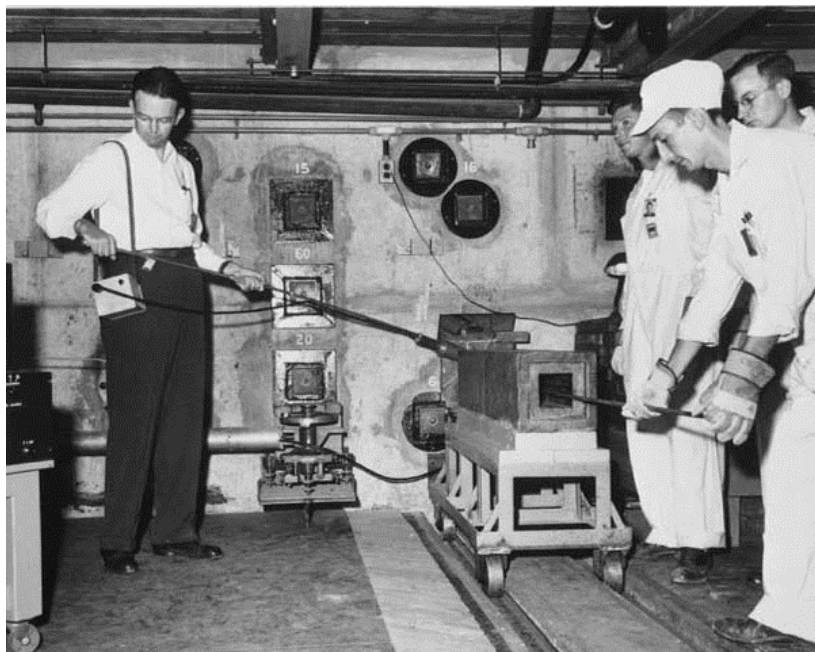


Figure 1. Early Long Handle Tools.

(Courtesy of Oak Ridge National Laboratory)

The first real innovation was the development of mechanical “master-slave” manipulators during the mid-1940s [5], [6], [7]. An early prototype is shown in Figure 2. These systems could work through significant shielding (attenuating walls with oil-filled viewing windows) to remove the operator from hazard exposure. Figure 3 shows a commercial mechanical manipulator system; these types of systems are still used today for stationary tasks such as in small hot cells where direct human access is not possible.

Remotely controlled electric manipulators were also developed by the late 1940s to remove the working envelope constraints of the mechanical manipulators [6], [7]. These systems used a switch box to control each individual joint, and motion was extremely slow. The same control philosophy later became the commercial power manipulators shown in Figure 4. These systems bear strong resemblance to robot manipulators except that there is no computer control; an operator directly controls all joint motions.

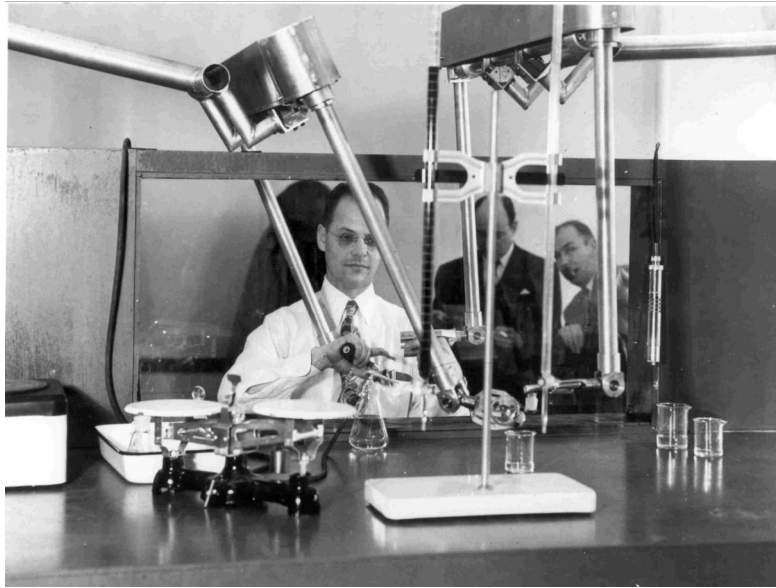


Figure 2. Early Mechanical Manipulator Prototype.

(Courtesy of Oak Ridge National Laboratory)

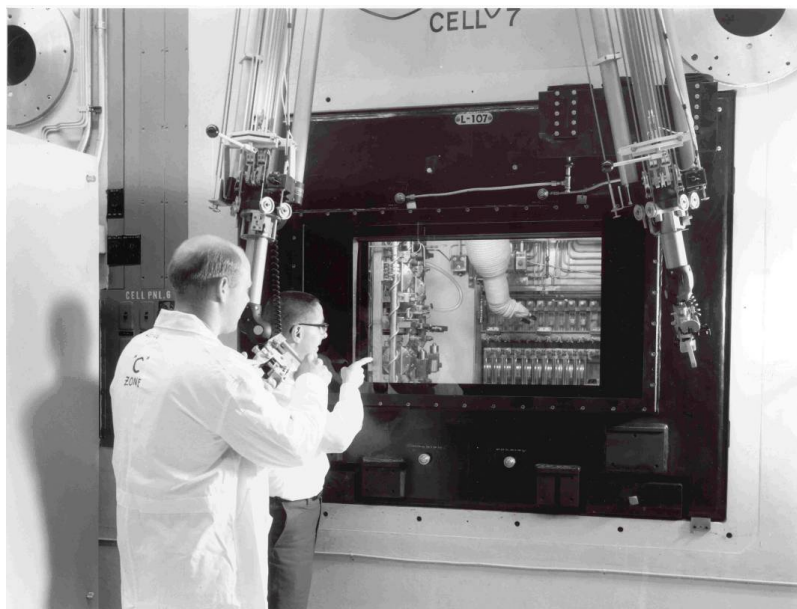


Figure 3. Commercial Through-the-Wall Manipulators.

(Courtesy of Oak Ridge National Laboratory)

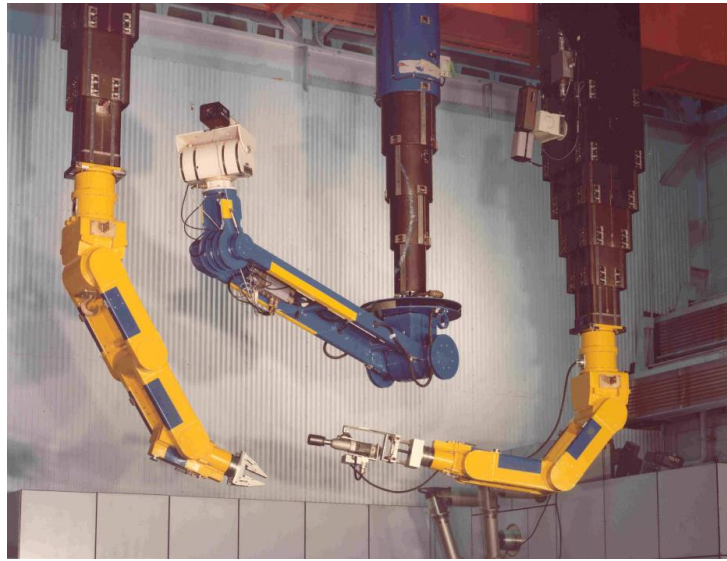


Figure 4. Commercial Power Manipulators Resemble Robots.

(Courtesy of Oak Ridge National Laboratory)

By the early 1950s, master-manipulator systems based on analog electric servomechanisms were developed [6], [7], [8]. Typical deployment modes used overhead transporters similar to bridge cranes. Some were mounted on mobile platforms. Analog electronics-based teleoperation became highly developed and remained the state-of-the-art baseline until about 1980 [9], [10], [11]. The systems worked well but were prone to amplifier drift and had to be retuned regularly. Teleoperated manipulation began to proliferate from the nuclear application area to space and subsea exploration from the 1950s through the 1980s and to medical use in the 1990s.

Oak Ridge National Laboratory (ORNL) worked with Central Research Laboratories to produce what is believed to be the first microprocessor-based teleoperated system shown in Figure 5. The technology has not significantly changed since that time. A current commercial state-of-the-art system is shown in Figure 6. Most teleoperators of this type have 6-DOF positional master controllers driving identical scale and configuration (kinematic replica) manipulator systems. Force reflection, reflecting the contact forces

from the manipulator back to the master controller, is common but by no means universal. Figures 7 and 8 illustrate a typical remote task execution and master controller station.

In the 1990s the DOE seriously began to address its contaminated facilities and hazardous waste problems. Two specific requirements were substantially different from the high radiation hot cell environments for which the first electric servomanipulators were developed. First the radiation environments were orders of magnitude weaker in most (not all) hazardous waste sites. Second the tools needed for dismantlement and cleanup were heavy and reflected large forces back into the manipulator systems during operation. Electric teleoperators were too fragile for use with these tools. High payload hydraulically-actuated manipulators developed for subsea teleoperation began to be used in the 1990s at the various DOE sites for hazardous waste cleanup tasks that were too hot for direct human hands-on work. One such application used for demolition of a nuclear research reactor is shown in Figure 9.

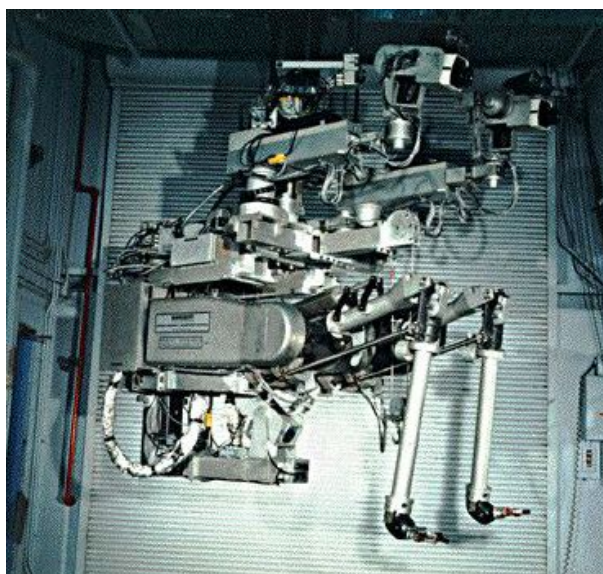


Figure 5. M-2 Servomanipulator.

(Courtesy of Oak Ridge National Laboratory)

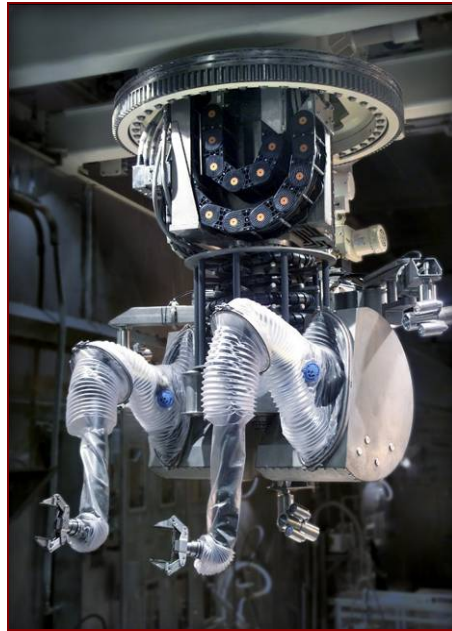


Figure 6. Telerob State-of-the-art Commercial Teleoperator.

(Courtesy of Oak Ridge National Laboratory)

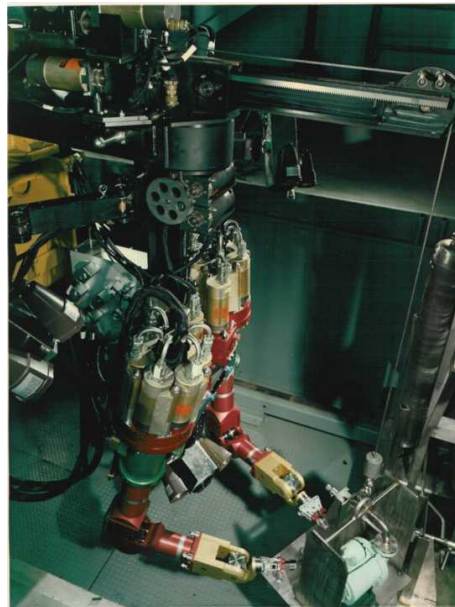


Figure 7. Advanced Servomanipulator Remotely Maintainable Manipulator.

(Courtesy of Oak Ridge National Laboratory)



Figure 8. Advanced Integrated Maintenance System Master Control Station.

(Courtesy of Oak Ridge National Laboratory)



Figure 9. Dual Arm Work Platform Using Schilling Hydraulic Manipulators.

(Courtesy of Oak Ridge National Laboratory)

A major limitation of all real world teleoperators is the use of the two-finger parallel jaw gripper with no or minimal sensing for grasping tasks. This dictates that tooling used by the manipulator be modified with special fixturing to allow firm grasping. Tool operation often must be completed without sensing useful to optimal operation. The use of smart tooling to place some actuation and sensing on the tool has been a relatively recent development that relieves the manipulator of some of the task dexterity requirements [12]. Figure 10 shows a plasma torch smart tool application to cut structural steel. Another approach that has been demonstrated is to modify the manipulators with multi-finger end-effectors to improve dexterity such as in Figure 11; however robustness and control issues have kept these types of manipulator hands from widespread use in D&D-type applications to date, and multi-fingered end-effectors typically do not yet have adequate sensing to support task completion [13].

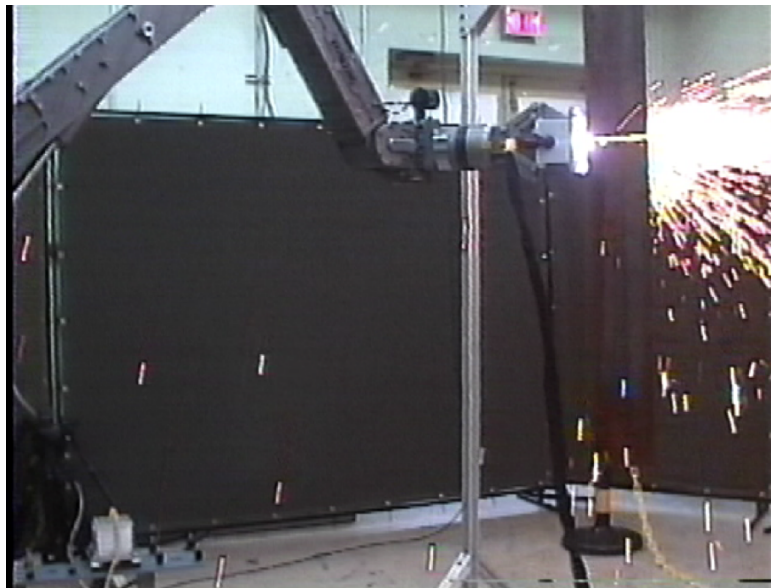


Figure 10. Schilling Smart Tooling Demonstration.

(Courtesy of Oak Ridge National Laboratory)

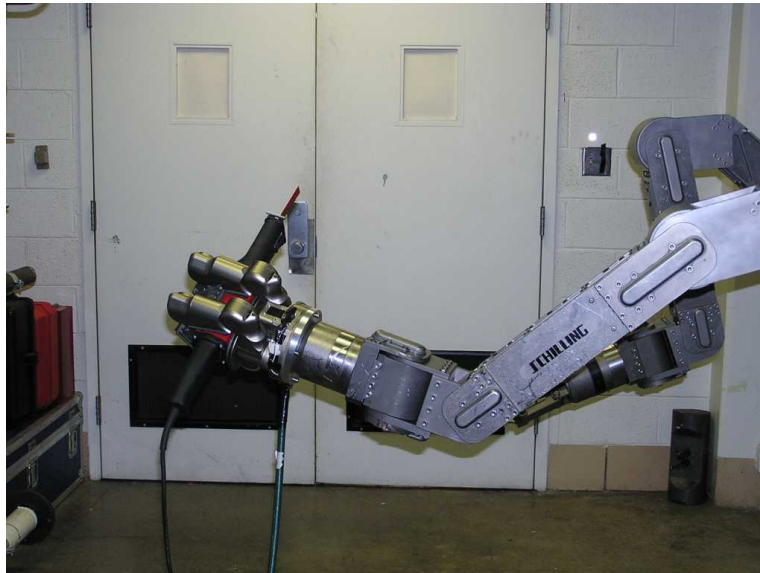


Figure 11. Barrett Wraptor Mounted on Schilling Manipulator at UTK.

2.3 Robotics

Sheridan's definition of a robot states:

A robot is an automatic apparatus or device that performs functions ordinarily ascribed to human beings, or operates with what appears to be almost human intelligence (adapted from Webster's Third International Dictionary.) ...The Robot Institute of America has defined a robot as a reprogrammable multi-functional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks [3].

Discounting mechanical toys and novelties that date back to ancient civilizations, the first useful industrial robot manipulator was created by Engelberger and Devol in the 1950s. Their thinking was directly inspired by nuclear manipulator systems, early numerical control machining techniques, and Isaac Asimov's science fiction stories of the 1940s

and 1950s [14]. Their Unimate[®] robot manipulator as shown in Figure 12 was the first commercially available robot manipulator. It was completely pre-programmed and automated for repetitive tasks. The robot manipulator as originally conceived is essentially a teleoperated manipulator with a preprogrammed front end dictating all motions in a predetermined sequence. Previously mentioned Figure 4 show remote manipulators that could have been or could be used as robots with the addition of a suitable front-end computer interface.

2.4 Telerobotics

Sheridan states that “a telerobot is an advanced form of teleoperator the behavior of which a human operator supervises through a computer intermediary.” [3] This implies an intermittent level of communication. However the approach and degree of emphasis on either teleoperation or robotics can vary significantly. Hamel presented a notation to describe this variation in emphasis [2]. Telerobotics can be defined as the fusion of teleoperation (T) and robotics (R) to complete a task. Telerobotics expressed as “tR” emphasizes robotics and is presented from a robot-centric perspective. This variety of telerobotics tends to be oriented towards the use of industrial robots as the target manipulator and generally relies on higher-level commands in a more supervisory control mode where the operator is not in continuous control of the motions of the manipulator. This is consistent with the Sheridan interpretation of telerobotics. “Tr” telerobotics emphasizes teleoperation finesse but adds robotic functionality to the teleoperator for improved task completion performance. Robotic functions in Tr typically use traded or shared control in some form of operator assist. Shared control combines human-controlled motions with robotic motions at the same time. Traded control sequences human controlled motion and robotic motion with one or the other having control at any one time [15], [16]. The approach presented in this dissertation best fits the Tr category of telerobotics using traded control.

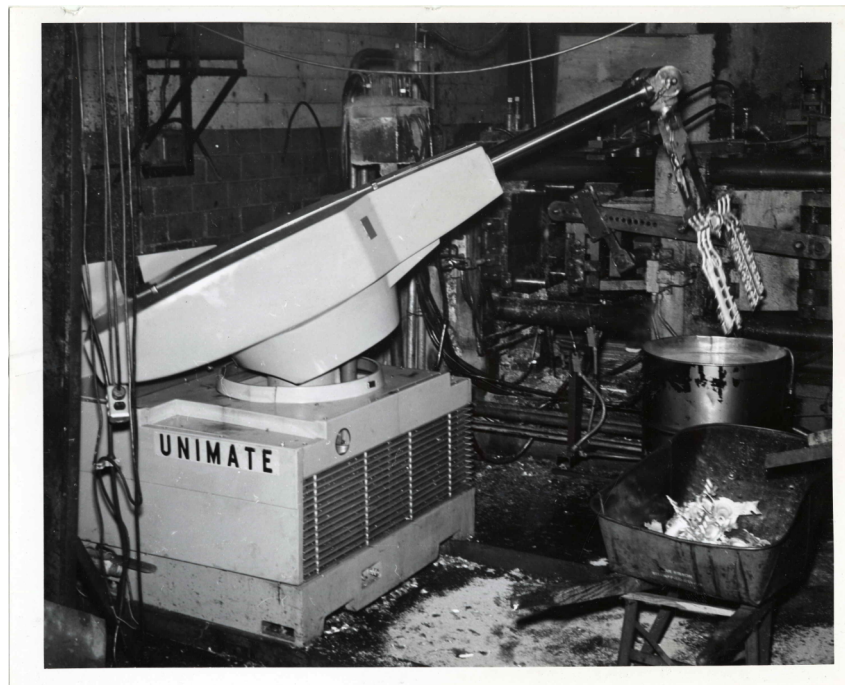


Figure 12. Unimate Robot.

(Courtesy of Division of Work & Industry, National Museum of American History, Behring Center, Smithsonian Institution)

Beginning in the 1980s the hazardous materials handling community began to explore the use of telerobotics in attempts to provide refined capability and reduced task completion times. These capabilities added various automated robotic functions to human-guided teleoperation. Typical functions include “software fixturing” to constrain manipulator motions to a plane or line of motion (a form of shared control where the human operator manages some aspects of motion while autonomous control manages others), traded control where the human operator hands off control to automated execution of narrowly defined sequences of tasks for a time and then receives it back after task execution is completed, and supervisory control where the operator manages tasks at the higher level instead of making every motion personally [3]. Except for some of the more simple software fixturing, telerobotics is rarely used in real world D&D manipulator applications

due primarily to incompatibilities and implementation issues with both the manipulator systems used and the unstructured environments encountered.

Smart tooling, a category of telerobotics whereby additional sensing and/or actuation is added to manipulation in the tooling acquired by the end-effector to improve task execution, has its roots in pick-and-place specialized remote tooling used by the nuclear industry since its inception. Smart tooling, when grasped in an end-effector, adds capability to limited manipulator systems. To date smart tooling systems are normally highly task specialized.

2.5 Behavior-based Robotics (BBR)

A concise definition of BBR provided by Arkin follows:

Behavior-based systems are composed of multiple behaviors (stimulus/response pairs suitable for a given environmental setting that is modulated by attention and determined by intention) that tightly couple perception and action to produce timely response in dynamic and unstructured worlds. These behaviors are coordinated through many possible mechanisms, including arbitration, fusion, and sequencing [17].

BBR is most typically associated with autonomous systems and sometimes with supervisory control-oriented (type tR) telerobotic systems. To date, BBR is also more often implemented on mobile platforms than with manipulation though manipulation has been a component of BBR since the 1980s [18].

BBR grew out of the realization and frustration that the traditional artificial intelligence (AI) schemes for robot control were not working outside of simplified laboratory test environments. Recent research has expanded the definition of BBR significantly and

created a hybrid form by incorporating more traditional AI concepts as well as new developments. However, this dissertation returns to the early foundations to explore initial development in support of traded control of smart tooling for telerobotic assists.

The earliest true autonomous robots were actually mobile platforms designed for psychological studies. These systems used what could be called a behavior-based control scheme implemented directly in analog electronics. The earliest design concepts were published in the 1930s [19], [20]. Contemporary concepts of the parallels and the intertwining between machine intelligence, control systems, and the human nervous system were expounded by Weiner as a new field of study, cybernetics, in 1948 [21]. The Machina Speculatrix cybernetic tortoise, shown in Figure 13, was first implemented in the late 1940s by W. Grey Walter for psychological studies [22], [23], [24].

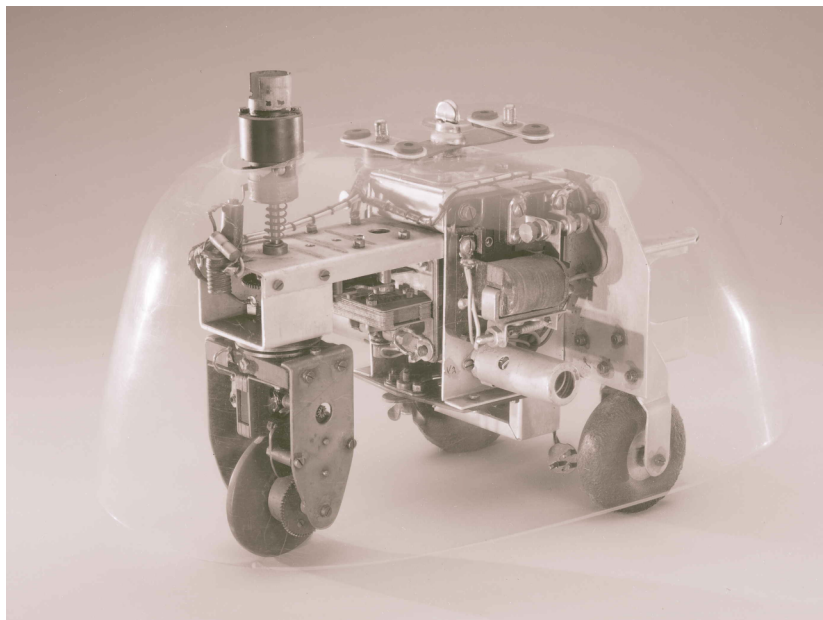


Figure 13. Machina Speculatrix Cybernetic Tortoise Replica.

*(Courtesy of Division of Work and Industry, National Museum of American History,
Behring Center, Smithsonian Institution)*

Behavior-based approaches then disappeared from the forefront of robotics research until the 1980s when they resurfaced in similar form migrating to higher-level computer control. The earliest modern implementation of the behavior-based approach was by Brooks at MIT although Braitenberg also published some psychological mental experiments in 1984 that appear to have been inspired by the earlier work [25], [26].

Several critical postulates can be put forward to describe the core of BBR. One of the most important is that "the world is its own best model" [27]. The plan should not be to model everything in the "world" and then attempt to calibrate the robot to that artificial world when the exact representation of what the robot needs to interact with is right in front of the robot. Sensors then become critical but the range of interaction is generally localized permitting more accurate ranging on simpler object fields and accommodating real-time updates which address flexibility and imprecision in the mobility/delivery system. At its simplest, BBR is sensor-based reactive control. However, BBR, while founded upon sensor-based reactive control, also requires an architecture of arbitration of the various behaviors necessary to complete a task. Brooks used a layered approach, labeled subsumption, of higher-level behaviors built on top of fundamental low-level behaviors [25]. The higher-level behaviors subsume (override) the lower level behaviors unless they fail for some reason; then the lower level behaviors can stand alone without any of the higher level functionality. Interaction or prioritization between behaviors may be via arbitration, fusion, and/or sequencing. Arkin labels behaviors as schema; each schema has a characteristic artificial potential field associated with its function. The fusion of behaviors is achieved by summing all of the schema potential fields into one overall potential field [28]. Pin's fuzzy logic-based BBR represents an approach to arbitration common in both Europe and Japan [29].

Additional core concepts to the BBR philosophy include situatedness and embodiment. Situatedness means that the robot is located in the world in which it is interacting; there is only a minimal abstract description of that world. The environment directly affects the actions of the robot. Embodiment means that the robots use sensors to "experience" the

world directly. Actions have direct consequence on the feedback of the robot's sensor systems. From the terminology used, it is fairly obvious that BBR was devised around a focus on autonomous robotics and not on human interactive telerobotics that is the focus of this dissertation.

Although they have morphed considerably into more complex architectures than the original concept, behavior-based systems have since become mainstream and taken over the more practical autonomous robotic mobile platform implementations in the field. Companies that sell small robotic devices, such as robot vacuum cleaners typically use BBR approaches [30]. The primary application for behavior-based systems has been autonomous robots, but they have also been applied to telerobotic systems of the supervisory control variety (tR) [28], [31].

2.6 Application Areas

The application area for this dissertation is anywhere positional teleoperation is used and especially where the manipulators need to handle substantial tooling to execute tasks. The initial and key application area for teleoperated manipulators has been the handling of radioactive materials, operational support of processes, and conducting maintenance for nuclear research facilities and nuclear power industries where human access is not possible. Especially within the last 20 years, teleoperated manipulation has been used at the DOE sites for hazardous waste cleanup in areas where radiation levels are too high for human presence, where contamination levels dictate the use of personal protective equipment that limits human mobility, efficiency, and duration of operation, or where chemical or physical hazards create too much of a liability to permit human presence.

Undersea and space applications grew out of the example created by the nuclear industry. Sub-sea manipulation has become crucial to oceanographic and archeological scientific investigations and off-shore oil exploration, oil rig maintenance, and accident mitigation.

A key difference in space-based applications is the significant time delay between operator interface and remote manipulator caused by the extreme distances encountered in space and by switching delays in communications equipment that relay the signals. Real-time high fidelity positional teleoperated manipulation is not currently feasible for space-based applications unless the master controller is in close proximity to the system being controlled. This means that tR telerobotics is more applicable than Tr telerobotics for most space-based applications.

Most recently teleoperated surgery or telesurgery has become a major application area. Minimally invasive robotic laparoscopic surgery removes the head surgeon from the operating table to an operator station directly adjacent to the surgery while the rest of the surgical staff directly tends to the operation hands-on. These systems are commercially available and expanding in use at hospitals across the U.S. Telesurgery where the surgeon is separated a great distance from the operation and support staff has been demonstrated, and full remote site telesurgery with no surgical staff on hand has been demonstrated by the DARPA TraumaPod project where a nurse robot provided the surgical support staff function [32], [33]. The result of this work should be applicable to power tool use in telesurgery as well as the core focus of manipulation for any hazardous environment.

Chapter 3

Relevant Work

3.1 Introduction

This chapter examines previous work and the resulting literature in order to establish the foundation and direction for this work. Unfortunately there is limited previous published intersecting work that ties teleoperation, telerobotics, or behavior-based robotics techniques to the use of tools and especially to the use of power tools and their interaction with the target task. Also where behavior-based techniques are used in telerobotics, they are typically of the tR type and not of the Tr type that is the focus of this dissertation. Therefore the literature survey is expanded to include the basic topics to establish the necessary foundation and to facilitate an extrapolation to tool-centric Tr-oriented telerobotics enhanced with selected relevant behavior-based concepts.

3.2 Teleoperated Tooling Tasks

The development of teleoperated manipulation was a direct result of the need to handle hazardous materials and to maintain process equipment during the World War II Manhattan Project. Pick and place of objects has always been one aspect of hazardous materials teleoperation, but the use of powered and hand tooling has always been a key and dominant requirement for task completion [14]. Much of this accumulated remote tooling design and application knowledge is not known outside of the DOE community though published guidelines do exist.

The technology for teleoperated force-reflecting 6-DOF manipulators was well sorted out and highly developed through the 1950s [6], [7], [34]. These manipulators primarily used cable- or metal tape-driven joint actuation and a two-finger parallel jaw gripper arrangement to articulate objects and deliver and operate tools to remote tasks. Where

servomanipulators had inadequate capacity or were too fragile to deploy the tooling required, power manipulators resembling crane-deployed inverted industrial robots and overhead crane-hook-deployed tools were used [4], [14].

These servomanipulators used joint-based analog control driven with kinematically similar master controllers. The analog control loops had to be frequently tuned to maintain optimal performance due to amplifier gain and zero offset drift. Since the controls were analog, there was little opportunity to augment these systems with automation. Many systems provided force reflection using a control loop scheme called position-position bilateral force reflection [34]. There were no force sensors used in the generation of force reflection. The per joint force reflected back to the master was generated by controller response to the position difference between the joint position of the remote manipulator and the corresponding joint position of the master controller [8].

Since teleoperated servomanipulators used a parallel jaw gripper end-effector that was not compatible with the irregular cylindrical shapes of most tools, custom tool fixturing was typically required to grasp and articulate the various tools. Grip pads that captured the fingers of the parallel jaw grippers were added. Depending on the reaction forces of the tools and the inability of the operator to precisely align and position that tool, compliant rubber links/pads were added to the tool fixturing. If the tool in question was powered, remote actuation was then adapted to operate the tool. These modifications drove cost and availability for remote tooling—more and more complicated modifications meant that fewer tool instances could be afforded. As previously mentioned, detailed guidelines have long existed for how to design, fixture, deploy, and use remote tooling for teleoperation [4].

Sometimes particularly large tooling would be of the pick-and-place variety whereby the manipulator system with the aid of an overhead crane would set a tool package in place on a task. The automated or semi-automated remote tool (a predecessor to current concepts of smart tooling) would then complete its specific task via remote control. Any

issues of tooling dynamics and control would be handled directly in the tool mechanical design and would not impact the manipulator [4], [14].

In the 1970s analog servomanipulators were converted first to minicomputer and then to microprocessor-based control [2], [35]. This minimized the analog drift problems and allowed rudimentary automation (telerobotics) for the first time. Features that were enhanced or added included motion scaling, variable force reflection ratios, and enhanced master controller indexing. Commercial digital manipulator systems to this day are based on the same control concepts as these first systems.

3.3 Telerobotic Tooling Tasks

As previously mentioned a telerobot is a system that beneficially combines human interaction and automation in a single robot system; the fusion of teleoperation and robotics is telerobotics. The key benefits typically sought are faster and/or better task completion, and lower operator fatigue that permits longer operation and better efficiency than would be possible with a pure teleoperated system. These desires all have relevance in tool usage along with the need to minimize tool and manipulator system damage.

Early work included the addition of subtask automation to traditional (compliant) teleoperated systems and had limited success [36], [37], [38]. To permit position-based force reflection in traditional joint control teleoperation, the manipulator and master controller joints require low actuation friction that tends toward high backlash and makes overall joint control compliant and imprecise. The resultant positional errors are not an issue for a human operator but are problematic for precise robotic positioning [36], [39]. Much telerobotics work after this time made use of industrial robots instead of teleoperators to gain precision of positioning at the cost of quality of teleoperation.

The earliest useful telerobotics work appears to have been completed by Vertut et al. and published in the mid-1980s [37]. Along with teach/playback-recorded motion, they also implemented software jigs and fixtures to constrain teleoperation motions to make it easier for an operator to use tools requiring precise alignment such as saws and drills.

Also in the 1980s there was a growing interest in breaking joint level control and kinematically identical master controllers with a move to Cartesian control. Khatib provided a thorough mathematical development of his operational space that has been foundational ever since [40]. Researchers began to try to use industrial robots for teleoperation and dissimilar master control schemes and multi-axis joysticks were tried with varying levels of success [41], [42]. Much of what drove this was that research communities did not have access to high fidelity servomanipulators due to their high cost. (A high fidelity digital dual arm master-manipulator electric teleoperator system cost approximately \$1.5M in 2010 [43].) In general these dissimilar kinematic systems do not compare favorably to traditional kinematic replica joint level teleoperation; however work in this area done by the French Commissariat à l'Énergie Atomique et aux Énergies Alternatives has made serious improvements in dissimilar Cartesian control through the use of traditional teleoperator master controllers driving industrial robots with a force-torque sensor in the slave manipulator base [44], [45], [46].

Chan et. al. at the University of Tennessee at Knoxville (UTK) attempted to expand on Vertut's work by focusing on various kinds of operator assists for tooling [47]. This work required that complex compliance matrices be set up by hand for each task. Everett later expanded on the operator assist efforts to include available sensor and model-based data to improve the quality of operation [16], [48]. This work also required complex setup procedures for each task. There is no question that operator assists add value to the precision of operation. The difficulty comes in setting up parameters to execute these tasks in a way that makes them useful and accessible. A key issue here is that the programming and engineering intuition required to implement task automation is beyond that of typical remote systems operators, and the amount of time required to configure the

system for a task may be longer than that required to struggle through the task via pure teleoperation. While the use of tooling was the focus of some of this work, it did not specifically incorporate tool/task interactions.

Space-based systems seem to be the only application area that has broadly adopted joysticks for their highest grade of teleoperation, but they have unusual work space constraints, motions must be slow to avoid imparting reactive forces in space-based systems, and great distances induce time delay into control making traditional positional teleoperation difficult [49]. Under these constraints a “fly-the-end-effector” approach to control, which is also more natural for the typical astronaut with a pilot background, is the most practical control architecture [42]. While mission specialists are no longer typically pilots, they undergo extensive training on task mockups to achieve proficiency with a limited set of tasks using the available control modes. D&D remote operators generally receive little to no system level training or practice. Under these circumstances, positional master controllers that function as an extension of the human operators hands provide more natural teleoperation.

The US National Aeronautics and Space Administration (NASA) has always maintained active research in teleoperation, telerobotics, and autonomous robotics [50], [51]. Early in their planning stages NASA acknowledged that moving from teleoperated systems to telerobotics (Tr) appeared to be the better approach although the National Bureau of Standards had determined to start with industrial robots and move back towards telerobotics (tR) by adding flexibility in operations and task programming. Hertzinger et. al. developed and flew a series of telerobotic dextrous manipulation experiments called the robot technology experiment (ROTEX) to explore master controller and control system control modes [52], [53].

Backes et. al., Hyati, and Lee worked at NASA to address issues of telerobotic shared and traded control for teleoperators [54], [15], [55]. This fundamental work does not

appear to have extended to the use of tooling for task completion or appreciably distinguished whether one mode was better than the other.

More recent NASA work in telerobotics has focused on creating the anthropomorphic Robonaut capable of articulating hand tools for space-based operations and potentially geological surveys on other planets. While highly capable, Robonaut has different operating parameters from those of earth-based D&D-type operations. It is relatively slow moving and is not designed to handle power tools capable of reflecting large forces back into the system [56]. Additionally, time-delayed operation issues due to distance and communications relay technology place constraints on space-based teleoperation and telerobotics that are not typically issues with earth-based D&D type operations. They are addressing a different set of task constraints.

End-effector tooling has always been a focus in the use of industrial robots where welding, painting, and various machine type operations such as deburring are common. Whitney et. al. did early work on robotic deburring solutions [57]. Solutions often did not transfer well to telerobotics, however, since industrial robots are stiff and the majority of teleoperators are not. In general Tr-oriented telerobotics requires solutions that accommodate the flexibility of the manipulator and its delivery system.

The DOE pursued telerobotics throughout the 1990s with the purpose of improving the efficiency of remediation operations where remote systems were required to protect people from hazardous environments. The Robotics Technology Development Program and later the Robotics Crosscut Program addressed issues in D&D, tank waste retrieval, buried waste, mixed waste disposal, and laboratory automation [58]. Several of these areas, in particular tanks waste retrieval and D&D, began to investigate relevant telerobotic issues with respect to tooling usage.

One area of application included storage tank waste retrieval and remediation using operator assists developed by Xi et. al. [59], [60]. They were concerned with integration

of human-based corrections into a preplanned robotic path to correct for path flaws and to avoid obstacles. Here the robotic task is the main activity and any operator motion is the assist. This approach was implemented and tested as an improvement for the tedious process of using remote systems to remove hazardous waste from storage tanks including scouring walls. The manipulator system was a large slender hydraulic manipulator with sluicing tooling on the end-effector. This system was controlled by a joystick and moved slowly and so was not a high fidelity teleoperator. In this case, very specific and narrowly defined telerobotic assists were defined and implemented as a means of reducing operator fatigue.

DOE also pursued manipulation, telerobotics, and tooling for typical D&D-type tasks. Since early testing showed that typical D&D tools such as hydraulic shears could reflect more than 300 lbs (1334N) of force back into the manipulator system, hydraulic teleoperated manipulators were substituted for the traditional but more fragile electric servomanipulators. Position-position force reflection was replaced by a force-torque sensor on the hydraulic manipulator in combination with dissimilar kinematic electric force reflecting master controllers [61]. Early work studied with varying success circular saws, band saws, reciprocating saws, sheet metal nibblers, and hydraulic shears with minimal fixturing and no telerobotics in an attempt to dismantle process equipment and the core of a research reactor. Substantial lessons learned on teleoperated tooling implementation issues were collected [1].

Later work included telerobotic plasma torch cutting of structural components that would be located in areas where accurate a priori models of the task would not be available [12]. This work involved smart tooling with both sensing and actuation and incorporated realistic manipulator control constraints such as dealing with a closed “black box” manipulator controller. Telerobotic functions included traditional robotics for pick-and-place of tools, the use of a teleoperated sensor tool (ultrasonic and laser rangefinder to establish edges and standoff distances and correlated with manipulator position) to establish a short term task model with cut paths and standoff distances (plasma torch

cutting requires the maintenance of an approximately 3mm to 7mm of standoff for proper cutting), and automated robotic use of a plasma torch cutting tool to execute the model generated. Capability was demonstrated for flat plates and complex cuts on structural angle iron. Each task instance was completely hand programmed using the generated points.

Hamel at UTK has conducted extensive telerobotics work that has specifically been oriented towards D&D-type cutting tasks and addresses the modeling issues via task-specific sensor-based modeling where an operator used the robot task space analyzer (RTSA) to identify and plan the task; task execution was model-based robotics using the human-machine cooperative telerobotics (HMCT) system [62], [63], [64], [65], [66]. Under the RTSA operation strategy, an operator used sensor data from both video and laser rangefinder to establish an object's location in space to create a task model of the particular D&D task to be completed, a task script was generated, and the task was automatically executed in model-based robotic mode. There was no direct task feedback during execution and no sensor-based registration of the manipulator to the task during execution.

The technique and process has been tested and proven using a manipulator-held bandsaw to cut mockup process piping. There are several remaining issues in this technology. RTSA was one of the earliest techniques to recognize that a local task model would have more utility than a world model. World models can take extreme amounts of time to properly construct and register impacting the efficiency of operation, and the real world is not static, especially in a D&D situation where all of the tasks are dismantling the "world". However RTSA's foundational philosophical shift begs the question as to how much of a task model is actually necessary to complete a task. This has not been fully explored. Other remaining issues include dealing with the error bubble of a sensor system mounted any appreciable distance from the target task that limits task and tool choices and the complexity of dealing with various shapes in the task modeling [67]. The use of tooling was a critical part of the operation, but tool disturbances were not incorporated.

Zhang furthered this work by focusing on tooling dynamics and disturbances of the band saw cutting task to provide stable and more consistent cutting operation [68]. This capability was added to the existing HMCT RTSA system but did not make use of the RTSA capability. The goals of this work included the generation of a “universal tooling interaction force prediction model” and a “grey prediction force/position parallel fuzzy controller...that compensates for tooling interaction forces.” This work dealt with a single hard programmed task in its demonstration and did not accommodate the ability to reprogram tasks, task target locations, or more broadly accommodate other tools.

Working with the same system, Kim noted that “highly unstructured environments and the continuous changing commands needed from the operator to counteract unexpected events make it impossible to develop a force assistance function using control algorithms based on any analytical form [65].” This was addressed with the incorporation of a fuzzy logic compensator narrowly defined for a specific task. This work identified issues with telerobotic tool fault detection that led to a series of efforts to find solutions using fuzzy logic, discrete wavelets, and neural networks.

Most recently UTK has focused on the use of a multi-finger end-effector to provide generic grasping of unfixtured tools [13], [69]. Fixturing has always been an expensive approach especially in situations such as cost-conscious D&D where tools wear out quickly. While generally relevant to this work, a multi-fingered end-effector was considered to be a complication to first attempts at telerobotic tooling control and so was not considered in this work.

Cannon launched a direction of work that examined grasping issues related to hand tools for a version of “point-and-direct” high level telerobotics using “virtual tools” [70], [71], [72], [73]. The ultimate goal was to provide supervisory level control of tools using in manufacturing type tasks including force control. The primary focus of this work was to

define how to grasp tools and did not address how to manage contact with the target task especially in the context of the use of tooling.

It is recognized that virtual fixturing as originally developed for teleoperated use of tooling in the 1980s is not task-flexible. Fixturing is generally based on manipulator coordinates and not on task coordinates. Aarno et. al. examined the use of adaptive virtual fixtures; however the focus was on predicting intended operator motions to define fixture adaptations and did not directly address accommodation of tooling [74].

Yu et. al. explored the possibility of using attractive and repulsive forces to align on a target, avoid an obstacle, or to follow a path using a Hidden Markov Model in an attempt to classify the apparent motions of a human operator to determine, select, and control the manipulator motion [75]. The focus was on determining the intended motions of the operator. The use of tooling contact issues during operation was not a concern or focus of that work.

The advancement of medical manipulation of small surgical tools for the removal of human operator tremor and to compensate for motion of the task is directly relevant to D&D tasks because the task or manipulator deployment system will typically move during task execution. Bebek and Cavusoglu used a whisker sensor to dynamically compensate for tool-to-task motion during surgery on a beating heart [76]. The purpose of the sensor system was to cancel relative motion between the surgical tools and the target of the surgery.

Some medical systems work has recognized that smart tooling is an important aspect of teleoperation and telerobotics. Saha under the guidance of Okamura examined the addition of force sensing directly onto surgical tooling to provide more sensory feedback to the surgeon remotely conducting the surgery with the purpose of improving the quality of task execution [77]. This work focused on force sensing in support of teleoperation

only and did not address telerobotics or the use of power tools which greatly complicates control schemes.

While not specifically telerobotics, the DARPA TraumaPod project to support surgical teleoperation with a robotic nurse developed tool/task interaction strategies for a 7-DOF robot manipulator that had to quickly interact with both compliant manipulators and rigid non-optimally aligned surgical subsystems supplying tools and surgical supplies. Insertion force limiting and incremental force-based calibration of subsystems in an outer control loop around a “black box” robot controller provide relevant control concepts for D&D telerobotics [32], [33].

There is some indication that interest is increasing in the use of smart tooling to facilitate teleoperated task execution. Dario et. al. discussed smart tooling and its impact on telesurgery and minimally invasive surgery [78]. This paper was a survey of potential smart tooling usage and did not specifically address tooling usage itself or control modes. There has been little implementation in this area to date.

3.4 BBR Tooling Tasks

Previous traditional autonomous robotic approaches to unstructured task environments normally used a sense-model-plan-act sequence of events; and though there has been progress there are still difficulties with most of these event stages in the context of real world task execution [27]. In order for a robotic system to interact with its environment, an adequate model must be made of the world or the specific task to be addressed. In the context of early telerobotics, this model was generated manually in a computer-aided drafting package using as-built drawings. This requires that expensive skilled technical labor spend significant time to generate models of the environment to be dismantled.

A better way to do this is to use some sort of sensor system to automatically model the robot's world, and there have been many significant research activities along these lines, and commercial systems now exist that will generate models with human operator assistance [79], [80], [81], [82]. Key problems include cost, physical robustness in the presence of tooling, accuracy, the process requirement for a containment dome over the sensor system that is currently unworkable, and long scanning time or analysis times of the various sensor systems (laser range finders and stereo or monocular video are the two most common). Knowledge representation, or interpretation of the data into a model that the robot can use in real time, is also an area requiring significant progress. Finally, registration and calibration of the position of the robot to the task model to establish where it and all the objects in the task are located is also critical.

Now consider that practical D&D systems are relatively large pieces of hardware, movable and flexible and not rigidly mounted, and operating in highly unstructured environments where complex objects reside in dirty low-contrast, low light environments (vision is necessary but not sufficient). High remote system flexibility means that the robot reference frames, normally taken to be fixed and rigid in a laboratory context, cannot be trusted and dictates that these models must be updated as necessary to maintain positional accuracy of the robot with respect to task objects. This could be nearly real-time depending on the bandwidth of the disruption to the robot base frame location. Dark, complex, and dirty facility environments tax sensors and recognition systems beyond current state of the art. The research community has made relatively little deployable progress in resolving these issues over the years [27].

While the primary focus of the BBR research community appears to be on mobile robot platforms, manipulation has also been addressed. Since most of these systems focus on total autonomy and not on human interaction, most of this work is marginally relevant to the proposed research. However there is some work in telerobotic manipulation and collaboration with human operators or peers.

Arkin, et. al., have participated in BBR research based on both reactive and hybrid deliberative/reactive control approaches [28], [83], [84], [85], [86], [87], [31], [88], [89]. His work documents the evolution of the schema-based approach to reactive control and its migration to a hierarchical hybrid deliberative/reactive architecture to take advantage of a priori task knowledge. This body of work also lays the groundwork for schema-based telerobotics, though the definition of telerobotics is typically kept at a fairly high supervisory level (tR) and is applied primarily to mobility and not manipulation, and especially not to power tooling. The example presented by *Reactive Control as a Substrate for Telerobotic Systems* does present one possible conceptual model to create a substrate for telemanipulation [28]. However this is tR-oriented telerobotics and would require a complete rework of the teleoperation scheme that would be incompatible with commercial positional teleoperation systems.

In work directly related to Arkin, Cameron et. al. and MacKenzie et. al., conducted research related to manipulation and mobility [90], [91]. The focus was on autonomous manipulation and not on interactive telerobotics. The most interesting concept here is the identification of the manipulator Jacobian and its relationship between joint torques and static forces at the end effector with the schema's potential fields used to specify behaviors. However this would require a complete change of approach to teleoperation for implementation.

Connell at MIT appears to have published some of the earliest work related to BBR-based manipulation [18]. The control system is based on Brook's subsumption architecture for behavior selection and is comprised of a collection of state machine-based behaviors. The robot is completely autonomous and optimized for finding and picking up soda cans. The key useful point here is the switching mode provided by the state machines. One of the limitations of schema-based summed potential fields is that they do not provide for mode switching that is provided by the subsumption state machine.

Stein is one of the few that has addressed behavior-based telemanipulation [92]. The primary focus here is time-delayed teleoperation for space-based operations. In this case because of the time delay issues, it is important to make the BBR system the primary mover and to add human level control as a secondary. While Stein refers to this approach as teleoperation, it is in fact supervisory control at a fairly high level and barely even tR. The control system behavior arbitration is based on subsumption.

Park et. al. of Argonne National Laboratory pursued BBR-based techniques for D&D-related manipulation [93], [94], [95], [96], [97]. The context of this work focuses on dual arm manipulation and task execution based on structured light sources and video processing. This work follows the schema-based approach of Arkin and makes use of the manipulator Jacobian in correlating manipulator action to the BBR schema. The sensor scheme is to use structured lighting and video image processing for behavior feedback. The intent of this work is to manipulate objects and tools, and while there is some mention of possibly using force/torque or motor current sensors to detect loads and anomalies, there are no sensors planned to address direct tooling-to-work-piece interactions or optimization of tool action based on proximity and contact information. This work is very much arm-centric, and the aspects of tool interaction are ignored. This approach would encounter difficulties in task execution—tool alignment, wear, and chattering—that would affect efficient task completion. As with almost all BBR type implementations, it also treats teleoperation as a secondary mode and not as the primary mode of operation.

Pettinaro explored the use of behavior-based techniques for the peg-in-hole insertion task [98]. The premise of this work was to consider how a blind human might use sensing to locate a hole and insert a peg. A zigzag and a hopping spiral pattern of motion were used to locate the hole. These approaches may work well to find a hole in a plane but does not translate well to the tooling tasks in three-dimensional space that may be surrounded by similar task objects.

Wasik and Saffiotti explored behavior-based approaches to arm control and examined previous work finding that much prior implementation of behavior-based systems for arm control were based on the sequencing of behaviors which they considered to be too limited to support generic grasping [99], [100]. Their work focused on vision-based grasping for a collection of pick and place task primitives. Their approach is fully autonomous and does not incorporate teleoperation, contact management, or concepts related to tooling interaction with its environment.

Stoytchev noted that studies focusing on robotic tool use were uncommon and had not been well addressed in the autonomous robotics community [101]. This is still true. He examined the use of behavioral approaches to characterize tools with a focus on having the robot learn the use characteristics of tools. The tools identified were simple items such as sticks that could be grasped and used to poke or prod objects. This work is preliminary. The focus was on learning how to use simple tools and not on the efficient use of existing tools. It therefore does not address the use of power tools.

Though not related to tooling, Pin described a minimal modeling approach to mobile robot navigation that used a fuzzy rule-based system [29], [102], [103]. Performance of a small set of 20 fuzzy rules was able to exceed the performance of 30,000 lines of code designed to attempt “crisp” image and sensor processing and navigation. The focus is on automated rule generation. The resultant is that the concept of a minimum model has value for real world implementation and that the use of a simple functional architecture based on behaviors may be able to exceed the performance of a system using more complex engineering models.

3.5 Tool Disturbances

Rapid oscillation of cutting teeth in conjunction with applied cutting force can produce “chattering” between the tool and the work piece. High frequency machine tool and saw

tooth chatter have been extensively studied by many researchers though the process is still not completely understood [104], [105], [106], [107], [108], [109], [110], [111], [112]. It is best if the working frequency of the tool contact can be kept far beyond the frequency that would normally impact manipulator dynamics; however, these tools invariably use universal motors where the motor's tendency to slow under increased load can move its frequency of operation into a range where it will be of concern.

Noakes investigated a chatter/disturbance solution based on prior machine tool chatter techniques that detect chatter with the ratio of variances of low and high accelerometer signals generated by the saw during cutting [113]. This is an empirical approach and thresholds must be established by experimental testing with the particular tool type. Standard digital signal processing techniques are used to split the signal into high and low frequency components for analysis. This approach only works to identify the presence of saw blade chatter and disturbance and does not mitigate chatter. Once the disturbance is detected, a procedure to modify operation to correct problem has to be devised that is dependent on specific task and tool circumstances.

3.6 Summary

In summary, there has been nearly no work that combines telerobotics, behavior-based concepts, and the use of power tooling in a way that is cognizant of the interactions between the tool and the task. However some general direction may be derived from previous work in the various non-intersecting subject areas.

For this work the use of a positional master controller in support of high fidelity teleoperation is a primary goal. Telerobotic assists emphasizing Tr mode of operation are desired so that teleoperation may be maintained as the primary mode of operation since unplanned tasks and events will always occur during operation. This means that supervisory modes of operation or those modes that might use joystick control to modify

an autonomous operation as has been previously done in some behavior-based architectures are not desired. The behavior-based architectures also tend to supplant rather than coexist with existing manipulator controllers which is also undesired.

A desire to maintain a standard teleoperation capability within an existing manipulator controller architecture while integrating telerobotic operator assists points to a traded control approach to permit switching between the control modes. This will also permit coexistence and ready integration between traditional teleoperation, robotic motion, and telerobotic assists. Traded control also affords the operator periodic breaks from concentrated physical motion to relieve fatigue in a way that shared control does not during longer operating sessions.

One of the most promising concepts from behavior-based techniques is to rely on sensor information to capture local model context rather than generating an abstract model. This is the concept of “the world is its own best model.” This offers significant promise in task execution with minimal modeling of each individual task before execution.

While multi-fingered end-effectors are ultimately desirable, they are currently unreliable for long-term operation and testing in the context of D&D tooling needs for this work. The effects of grasp on sensors is also a diversion from the intended goals of this topic. “Traditional” remote system tool fixturing is adopted for this work with the understanding that more generic grasping should be addressed at some future point.

Chapter 4

Testbed Description, Capabilities, and Limitations

4.1 Introduction

This chapter defines the test bed used in this work. Much of this system was pre-existent to this work though it has been extensively reworked. The current iteration of hardware and software owes much to the foundational work of Renbin Zhou and substantial ongoing work by Andrzej Nycz. A hardware description and the software architecture are described. System capabilities and limitations are defined since they impact implementation, performance, and test results.

4.2 Hardware Overview

The manipulator system used in this work, shown in Figure 14, consists of a pair of manipulators that are mounted on a cross beam and then mounted to a pedestal base bolted to the floor. The steel box beam is 1.22 m long and .203 m across the flats of the square. The manipulators are mounted 1.054 m apart between the centers of their base mounting points. The top of the box beam where the manipulators mount is located .845 m above the floor.

The manipulators used are Schilling Titan II hydraulic 6 degree-of-freedom (DOF) manipulators. The shoulder pitch joint uses a linear actuator (hydraulic cylinder). The rest of the joints are proprietary rotary designs. All joints except the gripper use resolvers for position indication; the gripper uses a linear variable differential transformer. The hydraulic system is described as 3000 psi (20,684 kPa) nominal with a flow rate of 1.5 – 5 gallons per minute (5.7 – 18.9 liters per minute). The manipulators are specifically designed for sub-sea use and are designed to withstand underwater depths up to 7,000 m below sea level. They are constructed of titanium for strength and corrosion resistance as

their most common use is off shore oil-rig maintenance. The use of these arms for hazardous waste cleanup is due to their robustness and payload capacity.

From center of the manipulator base to the tip of the parallel jaw gripper, length of the arm is 2.00 m. Payload capacity of the arm while at full extension is 109 kg; the mass of the arm is 79 kg. The parallel jaw grippers open to 0.152 m, have serrated finger faces for firm grasp and include a cylindrical T-shaped notch for positive grasp of tooling if fixturing is designed to support the “T-handle” approach.

The Schilling controller has been replaced with a PC/104-based controller developed by ORNL. The PC/104 controller was designed to provide basic teleoperation while supporting further development; the original Schilling controller was a “black box” that could not be modified and had limited means of control access. The controller, shown in Figure 15, is an open architecture unit based on the QNX4 real time operating system. The controller runs at a 200 Hz loop rate. It is essentially a joint position controller. UTK previously modified the controller to communicate with external systems via Ethernet; the original used a serial link to connect to the Schilling mini-master operator interface.

The operator station is shown in Figure 16. It consists of an Agile Engineering-supplied compact remote operator console with control chair, viewing system, and computer monitors. A Barrett Whole Arm Manipulator (WAM) configured as a 7-DOF master controller is mounted on the left side of the console.



Figure 14. Telerobotics Test Bed.



Figure 15. PC/104 Manipulator Controller.



Figure 16. Telerobotics Operator station.

4.3 Software Architecture and Implementation

The system level block diagram is provided in Figure 17. The system resides on a total of five computers interconnected with a dedicated Ethernet network. The system has no external connection to the Internet; therefore there is no traffic on the network that is not directly related to control. The collection of computers is a variety of hardware configurations and run various operating systems running software at various loop rates.

The central machine is the high level controller (HLC). This desktop PC manages all communications between the other machines, manages the Ethernet loop timing, coordinates the passing of variables between systems and programs via shared memory, and provides the forward kinematics for the WAM and the forward and inverse kinematics for the Schilling. The interface for manual teleoperation and the BBR-inspired controls also reside in the HLC.

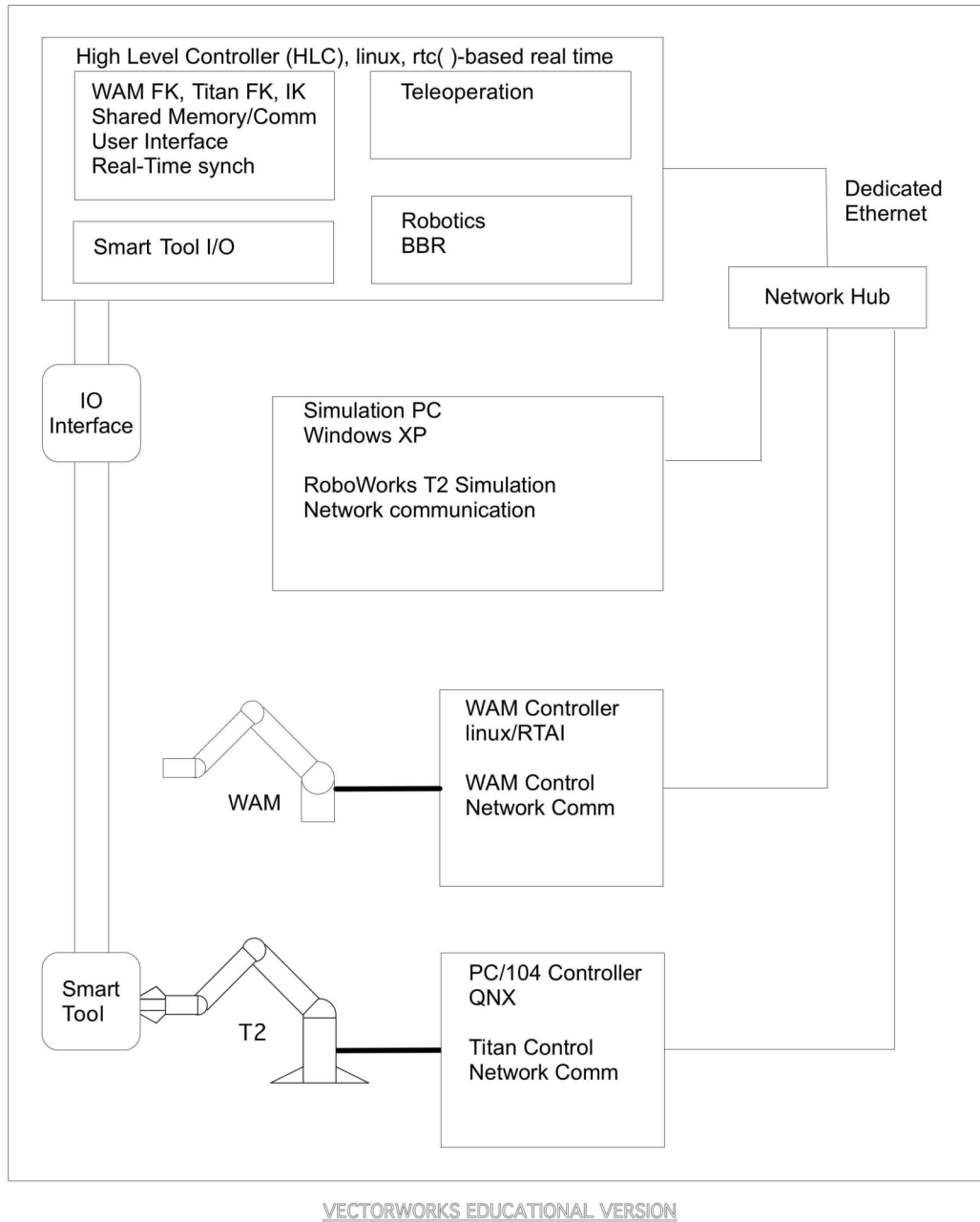


Figure 17. Test Bed System Level Block Diagram.

The operating system for the HLC is based on CentOS linux. Real time loop timing is synchronized via the `rtc()` system function call. Prior testing at UTK has indicated that this approach is valid to at least 128 Hz loop rate [114]. The intersystem Ethernet loop rate runs at approximately 32 Hz. The `rtc()` is provided to the main HLC program `server_hlcx()` since it is the point of coordination and timing between all processes on all of the networked systems.

The include file is `rtc.h`. The `rtc()` is configured as follows:

```
// required for the real time clock (rtc)
// rtc device file descriptor
    int rtc_fd;
    unsigned long dummy;
// variable for status response from /dev/rtc when interrupt returns
    unsigned long rtc_status;
// open the /dev/rtc device file
    rtc_fd = open("/dev/rtc", O_RDONLY);
    if(rtc_fd < 0) return -1;
// enable periodic interrupts, and set interval
if(ioctl(rtc_fd, RTC_PIE_ON, 0) < 0) return -1;
if(ioctl(rtc_fd, RTC_IRQP_SET, 128) < 0) return -1; // set to power of 2 up to 8196
// sets the loop rate directly in Hz; currently set for 128Hz.
```

It is used in the loop as follows:

```
// LOOP
    while (1)
    {
        code inside loop here
// trigger the periodic rtc interrupt
        read(rtc_fd, &dummy, sizeof(unsigned long));
    }
```

Unfortunately only one process on the computer can have the `rtc()` at runtime, and the `server_hlcx` process absorbs its full availability. This means that all other processes that

need to run in a timed loop must run using `nanosleep()`. Since loop timing is based on the process run time plus the sleep time, it must be set empirically using an iterative process, but this is not difficult to determine.

Usage of `nanosleep()` is managed as follows. The include file is `time.h`. Preliminary code outside of the timed loop is:

```
// Loop timing management using nanosleep( )

struct timespec ts;
ts.tv_sec = 0;
//ts.tv_nsec = 31250000; // 32 hz, not calibrated
ts.tv_nsec = 24400000; // calibrated for actual runtime 32 hz
```

At the end of each loop the function is called as follows to delay the initiation of the next loop as follows:

```
nanosleep(&ts, NULL);
```

The PC/104-based arm controller was described in the previous section. It is only responsible for the Schilling arm control and communications to the network.

The WAM controller is a Linux[®] box running the open source real time application interface (RTAI). It manages WAM control and its network interface only. Joint information and gravity compensation data are collected at a 500 Hz rate. Since the WAM runs as a master controller, joint motors are only used for the gravity compensation on the four lower driven joints of the manipulator. The three wrist joints are passive with position feedback only.

The WAM master controller and the Schilling manipulator are kinematically dissimilar; therefore traditional joint-to-joint teleoperation is not viable. A Cartesian-to-Cartesian control scheme is used to manage the dissimilar kinematics. This particular system has

been described previously in [115], [116]. The original system used an open loop forward kinematic-reverse kinematic scheme that did not reflect actual position of the Schilling manipulator. While it worked well for teleoperation since the operator provides additional perception feedback, it was found to be problematic with robotics and was changed to a closed loop kinematic approach by feeding back the Schilling Titan positions.

A separate Windows-based PC is used to run the RoboWorks[®] application that provides a simulation of the Schilling manipulator. The HLC is capable of connecting to either the actual Schilling manipulator or to the RoboWorks simulation of the Schilling. Using this interface, the WAM master or robotics routines can run either simulation or real manipulator. This feature is used only for checking software and visualization during operation of the real hardware.

4.4 HLC Interface

The HLC program `server_hlcx` uses a keyboard interface for commands and displays values on the screen indicating operating status of the system. See Figure 18. Important commands include:

- I Idle mode and Index mode for the master manipulator
- C Cartesian teleoperation
- M Toggles between real arm control and control of the RoboWorks simulation
- H Toggles between teleoperation and behavior/robotics modes

Additionally there are a similar series of commands for individual joint or Cartesian space motions. The original system did not have the capability for robotic motion; this was implemented as part of this work.

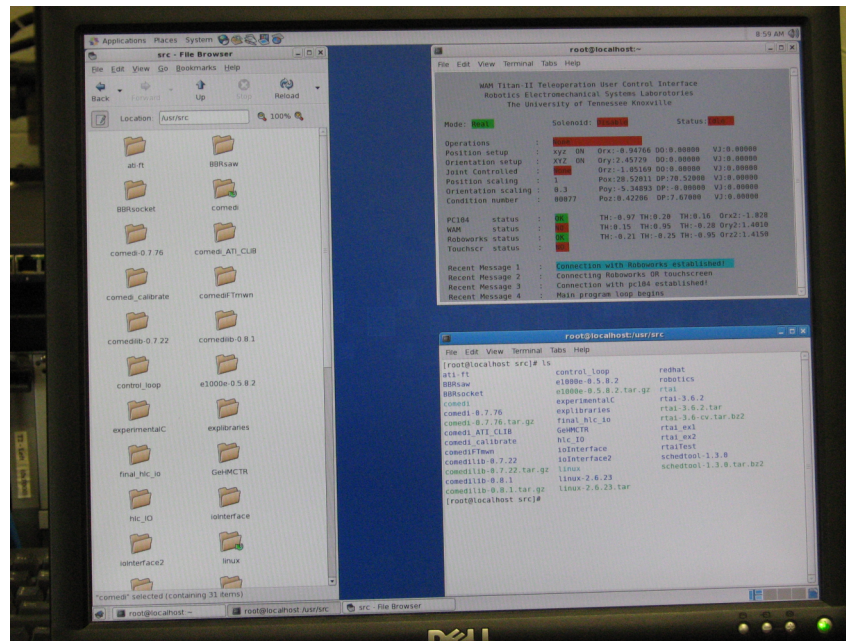


Figure 18. HLC Graphical Monitor.

4.5 Tooling Interfaces

Tool control and sensor interfaces are managed by two National Instruments PCI-6034E data acquisition cards located in the HLC. These cards have 16 single-ended analog input channels (or 8 differential input channels) and eight bits of digital I/O programmable as input or output bitwise. One card is dedicated to the ATI force/torque sensor interface. The other card is available for the analog and digital I/O necessary for tool interfacing. There is a custom built interface installed between the I/O cards and the smart tool. Block diagrams and schematics are located in the Appendix.

Software interface to the cards is provided through the open source Comedi data acquisition library for Linux. Comedi is also used to support reading of the force/torque sensor along with a library of routines supplied by force/torque sensor vendor (ATI). Software listings for the system support functions are provided in the Appendix.

4.6 System Limitations

There are several physical limitations to the test bed as implemented. The lab where the manipulator system is located is small and the workspace is constrained. The Schilling manipulators have several weaknesses in terms of their use for robotics. The test bed is workable for that which it was used; however there are limits to the level of finesse that can be demonstrated.

The lab where the manipulator system is located is a temporary installation. The room is too small to manage the proper reach between the manipulators and the mockups available for testing. While the setup appeared cramped on installation, issues did not show up until testing. The manipulator was having difficulty reaching tasks while maintaining full manipulability. The Schilling has an exceptionally long wrist link chain instead of a spherical wrist. This means that the manipulator should not have been mounted as close as it was to the mockups. However there was not additional space to move the system back from the mockups.

The Schilling manipulators have a high payload; however, they also have fairly high compliance, but the key weakness of the Schilling manipulators when used for robotics is position resolution. At full extension with the resolution of the joint resolvers, one bit change is equal to approximately 3mm. Therefore at best the controller can be expected to manage $\pm 3\text{mm}$ of positioning resolution with the arm at full extension.

Referring back to the Figure 17 block diagram and prior discussion it should be noted that the smart tool force torque sensor is limited to reading at approximately 128 Hz and that the network control update is limited to about 32 Hz. While this situation is highly realistic in terms of systems that would actually be used in the D&D world, it also reveals the limitations in terms of what can be done with various control techniques. Control strategies and proposed solutions that require high feedback loop rates are not possible.

Chapter 5

Telerobotic Tool Control Methodology Derived From Behavior-based Concepts

5.1 Introduction

In the very general context of remote tool-based operations, power tools contact surfaces, interact with, and change their environment in ways that normal grasping does not. Much of this interaction is variable depending on materials used in the task components, assembly torques of the target components, condition of the target components (such as the existence of rust/corrosion), and wear of the tool as part of its process of acting upon its environment. In general, these processes are not well understood, and previous research used comparatively complex solutions that have implementation issues for fieldable systems. The important issue is that the fundamental nature of the tooling and the associated processes are the dominant elements of basic task execution.

Most previous attempts have been based on model-based approaches. These assume that the task and tool delivery system may be completely and accurately modeled before the task is executed, that task objects are located where they are supposed to be, and that the manipulator system positioning the tooling goes where it is supposed to go. In actuality sensor systems working at a distance from their target object have error bubbles (a volume of measurement uncertainty) around the supposed target point. Manipulator systems, especially teleoperators that tend to be more compliant, may have substantial differences between where the control system intends to send the end-effector and where it actually goes. Finally the physical objects of interest in the task model must be rendered in such a way as to capture necessary manufacturing and installation details and variances.

The D&D “real world” is not composed of simple structures in orderly arrays of high contrast objects. Lighting is often minimal. Target tasks are typically dirty and/or corroded. As-built installations often use components that were not on the original

drawings or are installed in a more approximate fashion than design drawings might imply. For a D&D system operating in a contaminated environment where human access would not be possible, direct measurement of all of the variables necessary to define a tooling task may not be practical or even possible. This is not to say that models are unnecessary, or not useful, but rather that there is significant motivation to explore simpler approaches to telerobotic tool usage in environments such as D&D that directly measure the location of task objects while managing tool contact and the tool process.

An alternative and perhaps more desirable approach is to simplify the understanding of tool interactions through task decomposition, to characterize each particular step, to identify interactions that must be controlled, and to identify events that must be noted for successful operations. Behavior-based systems provide one perspective for task decomposition and a focus on interaction with the actual target task. Behavior-based approaches use local sensor systems to interact directly with the target task object where possible. Tasks are broken into simple sense-react motions that typically do only one thing. Behaviors are then grouped together to complete more complex overall tasks. This decomposition makes the overall approach simpler and readily implementable due to the inherent iterative nature of the process/philosophy. Task complexity may be addressed by adding additional behaviors to the existing set. Based on the literature review included in this research, behavior-based methods have not previously been used in tooling-centric situations and/or systems such as those used in remote handling and maintenance.

Specifically, the hypothesis for this research is that behavior-based methods offer a simple and effective way to implement telerobotic tool control within positional master controller-based teleoperation of complex remote tasks. The goal is to identify and use relevant concepts in behavior-based robotics to build task type models without the need to build a task instance model and to execute the task type model with the resulting implementation. A generalized methodology using selected behavior-based concepts appropriate for telerobotics and applicable across a wide range of tools is described here in terms of procedures and implementation rules.



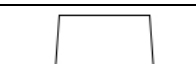
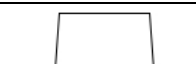


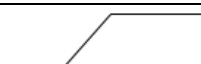
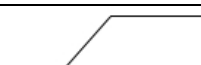
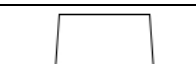


5.2 Identification of the Tool Set and Applicability of Technique

The primary focus of this work is improvement in the use of remote power tooling for D&D of contaminated facilities. While the tool set is continually being reevaluated for improvement, there are specific tools that are normally used for certain types of D&D operations. A typical set of tools and their function is listed in Table 2. A majority of the tools are cutting or disassembly tools of some type. Entire categories such as the range of abrasive blasting processes have been excluded for now because large quantities of individual particles moving in a wave against a task object cannot be individually measured or controlled.

In the course of exploring this topic, it was discovered that all tools that interact with their work piece have characteristic process signatures that are generally repeatable. The tool signature is particularly identifiable if the tool process is operated at a constant rate rather than by trying to control to a particular process variable. This signature may be used to monitor task progress, to infer quality of operation, and to identify task completion. The methodology pursued in this work requires a tool process that can be readily monitored for feedback and control. Contact and force are the most likely controllable tool parameters.

However, not all contact tools would be appropriate for this technique. The air chisel, jack hammer, and plasma torch are examples of tools that would be a poor fit for this technique. The air chisel, jack hammer, and sheet metal nibbler make high frequency high impact contact with a target surface to break up or break loose the target object for removal. Contact sensing and interpretation of impacts and generation of any type of response trajectory based on a series of these types of impacts would be impractical and exceedingly difficult. Tool interaction with the target surface is such that remote sensing of progress would be of limited value.

Table 2. D&D Tool Summary.

Tool	Target Object(s)/Task	Contact Signature
Reciprocating Saw	Sectioning pipes and smaller metal structural components	
Band Saw	Sectioning pipes and smaller metal structural components (limited to components where the ends are free)	
Circular Saw	Sectioning flat plate and large diameter vessels	
Hydraulic Shears	Sectioning pipes and structural components (limited use because it can damage the manipulator delivery system)	
Sheet Metal Nibblers	Sectioning sheet metal cabinets	
Milling Head/Router	Sectioning flat plate and large diameter vessels	
Impact Wrench	Bolt removal, large components	
Socket Tool/Nut Runner	Bolt removal, small components	
Drill	Sample collection and creation of drainage holes in pipes and vessels	
Air Chisel	Removal of bonded stacked blocks—concrete, graphite, etc.	
Jack Hammer	Removal of concrete	
Plasma Torch	Sectioning of metal structures	No contact

Plasma torch cutting requires precise control of an air gap standoff. While sensing of this control variable would be possible and relevant to the desired approach even though it is not contact based, the cutting trajectory must also be maintained at a fixed rate to ensure sectioning, and the cut path is predetermined a priori by an operator. This indicates that a model-based known-start-point to known-end-point path is the most practical means of control for the plasma torch, and therefore it is not a good fit for sensor-based techniques focused on contact and behavior-based principles.

In summary tools that generate a contact process or identifiable tool signature with a reasonable rate of repetition are the most likely application for the technique outlined in this work. This would include all tools from the table not in the preceding two paragraphs. Relevant tools rely on contact and management of forces to execute their function and to prevent binding of the tool. Fixed path generation, if necessary, would have to be considered as a higher-level function that would exist on top of the reactive control-based telerobotic tool control.

Returning to Table 2, the third column reveals that it is relatively straightforward to infer an expected process profile of the tool interacting with its task object in most cases and to distinguish between practical and impractical applications. Examination of the profile also points to what kinds of tool processes are amenable to certain types of control techniques. Note that the profile for cutting through objects such as pipe, structural elements, or drilling through objects indicates an initial contact followed by a process force or profile (actual to be determined experimentally), and then followed by a loss of contact. An impact wrench or powered socket tool will see a transition in forces as part of the tool process. This information can be used to establish a control sequence necessary to complete the desired task. This also points to the types and number of events that will need to be identified during task execution.

While a tool process signature can be hypothesized, this must be checked experimentally to validate the technique and to compare the expectations against the actual observed tool process signature. Transition thresholds that signal events must also be established experimentally since it is unknown what level of process noise or variation between task instances may be encountered a priori. Especially during any process that modifies the task object, process noise can be a major overriding concern.

While D&D tooling is the focus of this study and while validation of this work focused on contact sensing and force-torque profiles, the concept of monitoring tool process signatures on sensor measurement rather than trying to precisely maintain a process variable can be generalized to almost any tool process that interacts with its task object as long as a reliable means to measure the process variable can be established. Telerobotic use of power tools in task areas such as telesurgery, sub-sea exploration, and underwater oil rig maintenance are among the many potential expansions of this work. As long as an attempt is made to establish a constant rate of tool process progress, these techniques should also be applicable to non-powered hand tools such as saws, sanders, planes, knives—any tool application where there is a process and not simply an impact or contact that occurs between the tool and its task object. In summary, this approach is an alternate way of viewing manipulator/tool to task object interaction by expanding “contact” into a progressive process. The tool signature process is essentially a superset of “contact”.

One key difficulty is the creation of local sensing systems capable of precise useful measurement that will survive the tool processes. Simple contact such as grasping may be detected and controlled with a wide range of existing sensors. Tool processes, on the other hand, can be quite dynamic and destructive to sensing systems. This issue poses one significant obstacle to the full implementation of these techniques. Global sensing, while safe from the tool process, will have issues with

distance-to-target-based error bubbles. Local sensing designed to eliminate error bubbles may not survive even a single execution of the tool task due to vibration and impacts. This is particularly true of imaging cameras and rangefinders. Other sensors such as contact, inductive, capacitive, or electric fields may have vibration/impact issues but will also be susceptible to the electrical noise generated by the power tools in use. Tool signature monitoring is a more difficult problem than feedback for grasping.

5.3 Behavior Selection Methods and Impact on Technique Development

As previously mentioned, Arkin describes behavior selection to be by the various means of arbitration, fusion, or sequencing [17]. In BBR, arbitration is the switching that controls which behavior is executed at what time under what circumstances. One behavior is selected over another using a wide variety of prioritization schemes. Behavior fusion is the summation of directive vectors supplied by multiple behaviors to determine a cumulative path to goal. Sequencing is the preprogrammed selection of an order of actions to complete a goal. However the context of the use of sequencing is more often in the sense of sequenced assemblies of behaviors that use arbitration or fusion internally rather than sequencing of individual behaviors.

An examination of the actual tool processes in combination with a desire to replace the task instance model approach with a task type approach to the task execution reveals a problem with the use of the behavior-based robotics concept. Tool processes, especially those that are the focus of this activity, rely on a fixed sequence of subtasks for execution, i.e. they are inherently model-based. Behavior-based robotics is a combination of multiple sensor-based reactive functions and the intelligent behavior selection process used to determine which behavior(s) is (are) active at any given time. Downgrading the behavior selection process to an always repeated fixed sequence downgrades the degree of adherence to the spirit of behavior-based robotics. Although sequencing is an

acceptable, if primitive, means of behavior selection, it may be a more correct taxonomy to classify the technique generated in this work as an assembly of hard sequenced reactive functions using concepts found in behavior-based robotics.

The task type assembly itself is essentially an a priori model of the tool process that is executed the same every time. The reactive functions are used to locate the task object in space to anchor the task type model to its real task object instance and to control progress of the tool process itself. It has been quite common to find in implementation that reactive control augmented with available model-based information and planning provides a more suitable approach to task completion commonly known as hybrid deliberative/reactive control [117].

While sequencing has been chosen to execute the tooling functions, a question that should be asked is if there are places or instances where arbitration or fusion would be practical for selection of the next action. If so, sequencing could still be used to switch in and out groups of behaviors rather than individual behaviors. Sequencing itself could even be implemented by arbitration with behavior priorities, but that would be a contrivance more complicated than a sequence script since it would always execute the same way every time.

A change in priority (arbitration) during task execution indicates a change in the task at hand. Most tooling processes are concise and focused to a single task on a local task object. One possible situation requiring a change in task would be an event such as saw blade breakage that would render the task impossible to complete. Rather than have the operator intervene, alternate behaviors could recognize the problem, stop the tool process, and extract from the task. Behavior fusion has a more likely possibility of future use if also tied to sequencing of groups of behaviors. One example could include minimization of twisting moments on a circular saw blade in all three orientation axes while controlling the forward cutting force as the saw cuts through its task object. This could be

implemented by six behaviors with each one controlling one degree of freedom of saw motion all operating simultaneously to produce a six axis vector for motion of the saw.

5.4 Description of Methodology

This approach makes use of the human operator's ability to teleoperate tools into the tooling task vicinity, and then adds tool automation (operator assist functions) to complete the task and returns control back to the operator when the specific tooling operation has been completed. The operator completes gross motion by essentially pointing the business end of the tool towards the desired location of the task. Automation operates in a traded control mode to autonomously control contact forces, tool functions, and to reduce fatigue on the operator by giving them periodic breaks from physical manipulation. The step-wise process is illustrated in Figure 19 and outlined below.

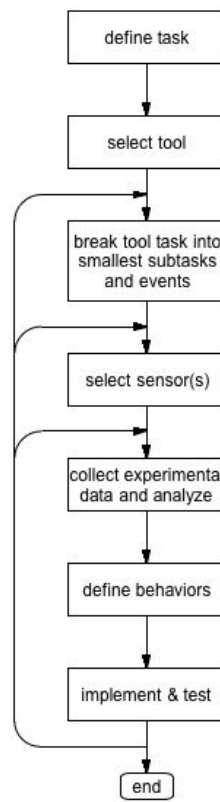


Figure 19. Smart Tool Behavior Development Methodology Block Diagram.

1. Describe the desired task characteristics and related constraints including what is known and what is not known. Consider expected task variability. Consider the task difficulty and the reason for the need to automate the task.
2. Select an appropriate tool based on task parameters. There may be several tool options for any given task.
3. Break the tooling task down into subtasks identifying motivators and/or events for start and end points of each subtask. Focus on minimal subtask complexity.
4. Choose preliminary sensing to identify events and control subtask processes while accommodating task, tool, manipulator delivery system, and operator limitations.
5. Conduct experiments to identify and analyze the characteristics of the specific subtask process to determine a suitable means of controlling that process.
6. Establish the requirements and characteristics for a set of sequenced functions to execute the tool task.
7. Implement and test the functions, first individually and then as a complete set to verify functionality. Iterate as necessary to previous steps to improve performance.

First, a specific task is identified along with the limiting factors involved in executing that task such as access to and clearances around the target object and material composition and structural characteristics of the target object. Characteristics of the operation that might make the task easier or harder to execute should also be identified at this time. Though there are often various options as to what tool may be chosen to implement a given tool task, task characteristics may point to a best option.

Tool operations are not random or arbitrary in terms of what happens when; they are composed of a specific sequence of operations that are subtasks of the overall tool process. Once the task has been defined and a specific tool has been selected, the tool task is examined to segment it into subtasks that are as simple as possible. These should include specific motions needed to approach and retract from the target task and how that approach should be executed. How first contact between the tool and the target task is made and what its purpose is in the tooling operation should also be identified at this point. Standoff from the task object is common and should be defined if that is necessary for tool operation and whether the distance is critical or convenient to operation.

The core of the task is the actual tooling operation on the target component, such as cutting a pipe, unbolting a bolt, drilling a specific material, or cutting a section of a tank. Rates of operation, forces encountered, and position or orientation operational constraints should be outlined. Questions such as the following must be answered. Is this a position-based task, a force-based task, a combination of the two, or something else? How is task completion defined?

For each of the various subtasks, the need for sensing must be established. Sensors must be selected to determine the required events. Sensor suitability is determined not only by the ability to measure the appropriate event or process but also by survivability given the tool characteristics (impacts, vibrations, forces, torques, the presence of fluids or other process debris) and target task interference (clearances around the task object that preclude local sensor mounting or that occlude the task target from sensing). Environmental concerns such as available light levels or chemical or radiological hazards that may constrain sensor choices must be identified. If a particular subtask function is not event critical or is impractical to measure, a model-based time/distance parameter should be investigated to determine suitability and whether its use would assist or hinder robust task execution.

Sensor selection should also include awareness of the manipulator system's capabilities and limitations with regard to sensor-based controls. Can the sensor system be integrated into the manipulator controller, or does it need to reside outside of the manipulator controller? For D&D type systems in particular, smart tooling that applies sensors to an external controller not directly part of the manipulator will be the norm due to cost constraints on manipulator systems and the specific sensing requirements for a particular tool and task. This affects the useful task bandwidth of the information that the sensor can deliver to impact control outcomes.

The next phase of implementation is the collection and analysis of experimental data in order to design reactive functions that map to the corresponding tool subtasks. It is necessary to establish this information experimentally because tooling data of this type does not yet exist in published literature. The motivation for this effort is to determine how the tool processes work, to identify events that would signal subtask start, stop, and progress, and to identify any relevant information that should be tracked during execution of a specific tool process. Required information would include what contact information can practically be collected as far as locating and identifying a desired target in space and what the tool process itself looks like to the available sensor suite. This information feeds function implementation with contact thresholds or tool process characteristic signatures.

In order to complete these tests, the prototype smart tool must be assembled into a package containing the tool, selected sensors, and any necessary fixturing to support manipulator grasping. Trajectories are then programmed as predecessors to the subtask reactive functions so that representative data may be collected. For example, a timed-fixed rate trajectory to cut a horizontal pipe will generate a specific force profile as the pipe is cut. The subtask may then be broken down into measurable segments or events that can be controlled or identified as points of progress.

The complexity of the required sensing and associated control will be dependent on the complexity of the tool process that is being controlled. More complicated tool processes

will require more complicated sensors and controls. Initial sensor selection is determined by an estimation of what needs to be measured. In experimentation and analysis, it may become apparent that additional or different sensing is required from what was initially selected. If a tool process cannot be reasonably measured, estimates or alternatives based on models of the subtask will have to be created.

The end result of these development steps is a set of function requirements needed to implement a set of sequenced reactive functions to execute the desired task with a given tool, using selected sensors, and within the constraints of the available manipulator system and operator skill sets. Reactive functions are then implemented according to requirements, tested individually, and then combined successively into the overall collection of behaviors to complete the tool task.

Reactive functions are specifically matched to the subtasks of the task decomposition and are generally designed to make one simple motion in response to a sensor value or until some sensor measurement is reached. A motion in a certain direction until contact on a target object would be one example. Another example would be a downward motion to cut a horizontal pipe while monitoring forces encountered by the saw blade as it passes through the pipe to determine progress and final success of the cut. These are specifically closed loop in nature; there is direct sensor feedback from contact with objects in the tool task space.

Open loop actions have value to provide functionality where sensor information is not available or impractical to acquire (such as when sensors would be regularly damaged by the tool process) or where the desired action is not critical and there is no hazard to the open loop motion. An example would be to follow a move to contact behavior with a predetermined standoff motion based on the kinematics of the manipulator rather than to use stand off sensors. While interpretation and definition varies somewhat in the behavior-based community, open loop behaviors, also known as “ballistic” behaviors, are included in the accepted tool kit of functions. One interpretation considers that they are

essentially a timed model-based behavior where the robot executes a pre-programmed motion for a predetermined amount of time. These can be applied to tool-based telerobotics in limited circumstances though they are not reactive functions.

In summary, this section describes a new methodology for telerobotic tool control using appropriate selected behavior-based concepts to enhance operation in unstructured environments. Once the task is identified and the tool is selected, the tool task is broken down into a series of sequenced tool subtasks that are decomposed to the simplest level practical. Sensors are then selected to measure the interaction of the subtask with its target object. Experiments are conducted to collect real world data as to how each subtask interacts with its target in terms of contact information and tool processes. An analysis of the experimental data is used to define function characteristics and possibly to modify tool and sensor implementation. Finally the set of reactive functions is implemented and tested first individually and then as a progressive sequenced collection to verify the complete tool task as functional and robust for its given task and operating constraints. It is believed that this methodology offers a simple, yet comprehensive, way of integrating tooling operations in more efficient ways to the classes of teleoperators used in unstructured and uncertain task environments.

5.5 Implementation Guidelines

The outlined telerobotics concept is functionally illustrated in Figure 20. The operator teleoperates tool delivery to the task by using the manipulator to maneuver the tool point of contact oriented towards the task but without actual contact. Depending on the task there may well be certain approach issues to consider. For example a saw blade must be positioned such that the blade's cutting surface is oriented correctly towards and above the surface that it will be cutting.

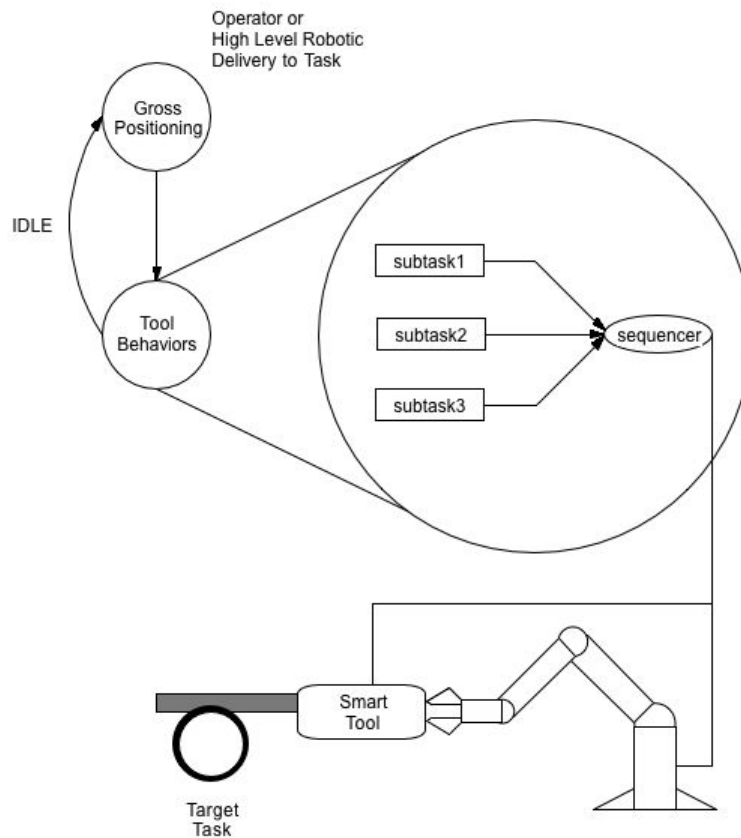


Figure 20. Concept Block Diagram.

Once the gross positioning and pointing have been completed, control is “traded” to the behaviors by the operator. The collection of functions then execute their task automatically and return control of the system to a safe mode for the operator or high level controller to take control and move on to the next location for task execution. A task instance model is never generated, and the operator determines where to execute the tool task.

The task instance model is replaced by a task type model that is encoded in the sequence and function of the functions, both reactive and ballistic. Sequencing is managed by

calling the functions sequentially in a structured program that is essentially a script. Functions are designed such that they terminate with a sensor event or control signal if closed loop or a time limit if open loop. It also becomes easy to edit or add to the script by inserting additional functions into the sequence. Each function may be tested individually by using it alone in the script program. The format is then simply as follows and as illustrated by Figure 21:

```
task ( )
{
    subtask( );
    subtask( );
    subtask( );
}
```

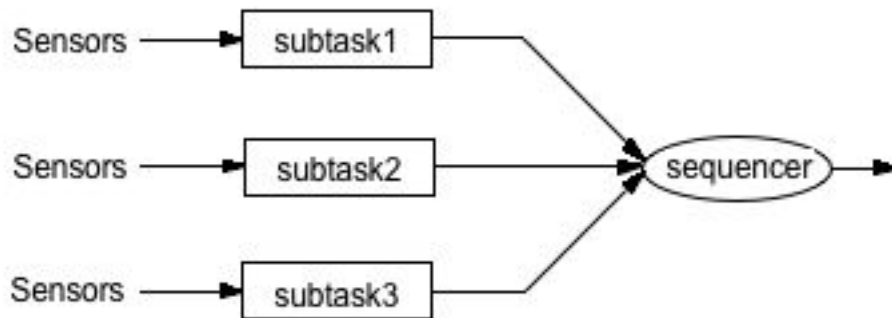


Figure 21. Behavior Selection Sequencing.

The functions themselves are concise subtasks that do one thing based on a reactive “sense-act” model with no planning involved during execution. Given a specific sensor input, the output is predefined and preprogrammed. A function may be a control loop that reads sensors and provides a scaled output, or it could be a generic move based on time

and/or initiated or terminated by sensor input. The functions in a task sequence may have divergent approaches to achieve their ends; they are not necessarily homogeneous in implementation approach.

5.6 Managing Human or Robot to Telerobotic Interaction

The base mode for this work is teleoperation of the tool to complete the task with telerobotic assistance afforded via traded control. The secondary mode of operation is robotic tool delivery to task with assistance via traded control once the target region is reached. Except for the details of how the tradeoff occurs, automated task execution is managed in the same way for both operator and robotics via high-level supervisory controller.

In telerobotic assistance, the human operator positions the teleoperated tool according to best effort, points the tool tip at the target task, and manually triggers the execution of the telerobotic task. When the task concludes, it automatically passes control back to the operator in a safe IDLE mode. The operator then takes control manually of teleoperation to move to the next task. This process happens whether task execution succeeds or fails. If task execution succeeds, the operator simply moves on to another gross positioning of a task of the same type. If task execution fails, the operator can reposition the end-effector and try again or choose to move to the next task regardless.

Autonomous robot switching to the local sensor-based task automation (telerobotics for the human operator) is a simple transition based on completion of the preplanned trajectory. When the trajectory is done, control is passed to the sequencer without any operator interaction or direction. When the sequence of tool tasks is completed, control is passed back to the robotic trajectory generator.

Chapter 6

Functional Implementation

6.1 Introduction

This chapter discusses concept implementation and elucidates the process with two realistic D&D tooling tasks—cutting a horizontal pipe with a reciprocating saw and removing a bolt with a powered socket tool. The assembly of reactive functions is developed according to the process outlined in the chapter on methodology. Although this chapter includes experimental testing to establish final function definition, the following chapter addresses experimental testing of the system of functions for performance evaluation, validation, and discussion of results.

6.2 Cutting a Horizontal Pipe With a Reciprocating Saw

6.2.1 Task Definition

The first task selected is to cut a horizontal metal process pipe approximately two inches in diameter, although the technique will actually accommodate a range of pipe sizes automatically. A representative pipe task is shown in Figure 22. The mockup and hardware located behind the mockup are somewhat representative of the level of clutter that may be seen in the real world, except that the task light levels will typically be much lower with much more shadow and dark background, reducing available image contrast. An example of an actual remote viewing video image used by an operator to during dismantlement of process piping via remote manipulator is shown in Figure 23.



Figure 22. Horizontal Pipe Task.

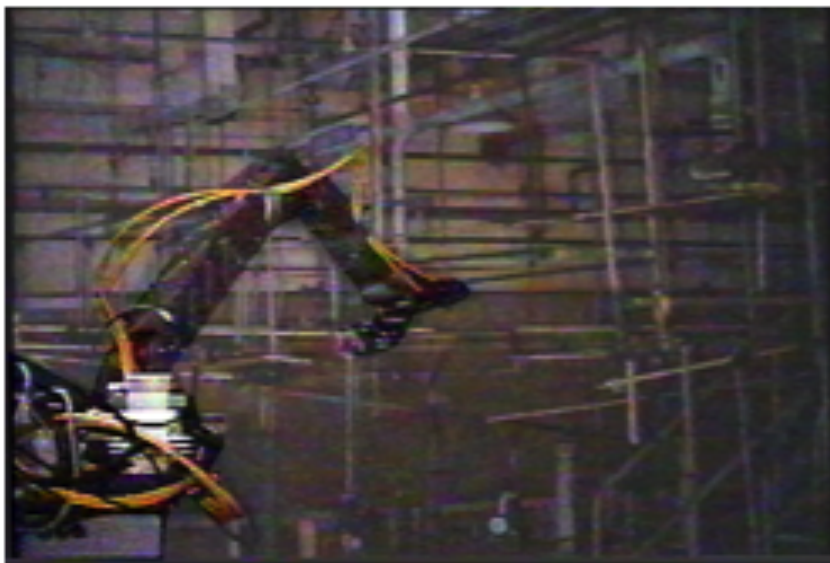


Figure 23. Real World Piping Arrays and Viewing Limitations.

(Courtesy of Oak Ridge National Laboratory)

Process piping occurs in standard sizes and materials. Piping sizes are based on commercial standards and include various standardized diameters. The wall thickness is defined by “schedule” such as schedule 40, and most process piping is either schedule 40 or 80. Standard 2-inch schedule 40 black iron pipe as used in the mockup available for this work has an outside diameter of 60.3mm and a wall thickness of 5.5mm, yielding an inside diameter of 49.9mm. An end view of the pipe is shown in Figure 24.



Figure 24. Pipe End Section.

6.2.2 Tool Selection and Description

Cutting process piping remotely is a difficult task. Small piping may be cut using a hydraulic shear. Larger piping requires the use of a saw; however saws are problematic with free hand positional teleoperation. Binding and maintenance of proper force levels are common issues. Band saws have been used to some extent, but they create problems when the two sides of the cut pipe capture the blade so that the saw cannot be removed from the task. Reciprocating saws have generally not been successful in the field but would be a serious asset to remote dismantlement and are a candidate for remote execution if suitable telerobotic controls can be implemented to assist the operator. The reciprocating saw is selected for this task in an attempt to provide new capability for remote systems that currently have difficulty deploying that particular saw type.

The particular hand held reciprocating saw to be used for this study is shown in Figure 25. The saw is designed to be held by both hands when used by a human operator. A 120 volts (V) alternating current (AC) 1050W universal motor is sandwiched between a rear grip and a front section covered with rubber to facilitate firm gripping of the tool by hand. Universal motors slow substantially under load and will stall if sufficient force is applied to them. As the saw slows it may excite the manipulator causing it to oscillate uncontrollably. Force and/or cutting progression through the work piece must be controlled such that the saw blade oscillating frequency stays high enough to be significantly beyond the bandwidth of the manipulator.



Figure 25. Hand Held Reciprocating Saw.

The length of the tool is 451mm from the tool foot (work piece contact point) to the end of the handle or 572mm from the tip of the blade to the base of the handle with the blade at full extension. The tool is about 76mm wide at its widest part. The mass of the tool is 3360g. The center of gravity of the tool is 191mm back from the tool foot. The motor module (the best location for grasp fixturing due to shape) is located from 191mm inches to 302mm from the tool foot.

The blade is 152.4mm (6 inches) long by 19mm (3/4 inches) wide by about 1.6mm thick with 12 teeth per inch. Blade oscillation travel is 25.4mm (1 inch) at 2280 oscillations per minute while under no load (38Hz for blade motion). This translates to 912 tooth cuts per second on the work piece. The material to be cut determines the blade material and number and configuration of the teeth per unit of blade length. Saw specifications are summarized in Table 3.

Table 3. Reciprocating Saw Specifications Summary.

Characteristic	Specification
Tool body length	451mm
Tool length w/ blade	572mm
Tool width	76mm
Blade dimensions	152.4mm long by 19mm high by 1.6mm thick (6 inches by .75 inch by 1/16 inch, 12 teeth/inch)
Mass	3560g
CG	191mm back from tool foot
Location for fixturing	191mm to 302mm back from tool foot
Power	120VAC, 1050W, universal motor
No load blade speed	2280 cycles/minute or 38 Hz, 912 teeth/second

The reciprocating saw smart tool is shown in Figures 26 and 27 assembled with grasping block and force/torque sensor. The force/torque sensor measures for load on the saw foot for contact and load on the blade for cutting progress. Sensor signals and power are routed back to the control computer through a bundled cable. As completed, the mass of the saw smart tool with all fixturing is 14.38kg.



Figure 26. Reciprocating Saw Smart Tool.

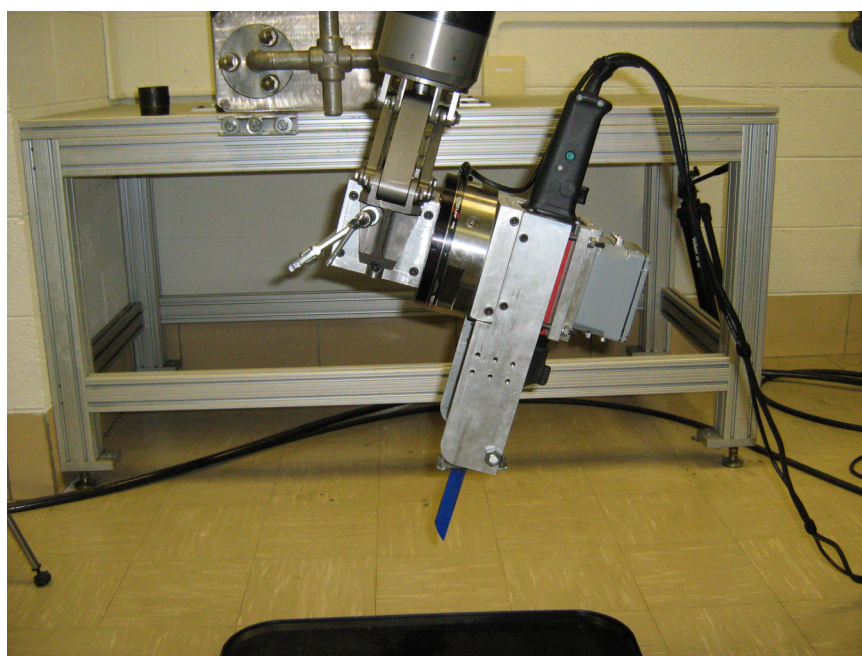


Figure 27. Reciprocating Saw Mounted in Gripper.

6.2.3 Subtask Definition

Given selection of the task and the tool, the subtasks necessary to complete the overall task must be defined by examining the process. These then become the functions or subcomponents of the functions depending on best implementation method. A reasonable assumption is made that an operator would be able to deliver the tool to reasonably close proximity to the task within an error bubble of a few centimeters and can point the tool at the task with the saw blade generally above the pipe to be cut. The goal is to have automation manage contact and cutting progress.

All actions are triggered by the sequencer as a starting event. Available sensor events are identified for each task/subtask.

The first task is to find the pipe.

Approach to contact roughly horizontally. (event = contact)

Back off to create standoff to prevent binding. (event = no contact)

Approach to contact to find the pipe roughly vertically. (event = contact)

Back off to permit starting saw blade without binding. (event = no contact)

The next task is to level the saw so that the cut is as square as practical. (event = level)

(It was later determined that practicality dictated that the saw be leveled at the start of the process.)

The next task is to cut the pipe.

Start the saw blade free of the pipe.

Move to contact the pipe and note when contact is made. (event = contact)

Cut through the pipe. (monitor or control forces/torques)

Note when the cut is completed. (event = no contact)

Turn off the saw blade.

The final task is to clear the pipe to return control back to the operator.

Move clear of the pipe.

Return control to the operator.

6.2.4 Sensor Selection

Next a sensor or sensors must be selected that can provide sufficient input for concept validation and subtask completion.

Though it is subject to placement accuracy and precision of the manipulator, Cartesian “global positioning” of the tool in its task space is available from kinematic equations. Behavior-based mobile platforms do not normally have access to global positioning information; however, it is available here. Due to the kinematics of the Schilling manipulator, the wrist roll joint position resolver can be used as a saw level indicator.

The business end of the tool moves and therefore is not amenable to direct placement of local sensing at the point of contact as would be possible with finger contact sensors for grasping. A six degree-of-freedom (DOF) force-torque sensor is available as mounted in the generic tool fixture and is used for measurement of contact forces and moments. While other sensors may be possible, sensor availability and robustness against damage due to the tooling process drove sensor selection to the force-torque sensor as an example to validate the concept.

Referring to Figure 28, contact in the forward direction of the tool is afforded by force pushback in the $-F_x$ direction and torque in the $-R_y$ direction (rotation about y since the tool is offset from the sensor face plate) of the force/torque sensor. Experimental testing showed that the $-F_x$ axis was sufficient to indicate contact. In addition force on the saw blade is indicated by sensor signals in the $+R_y$ direction of the force/torque sensor.

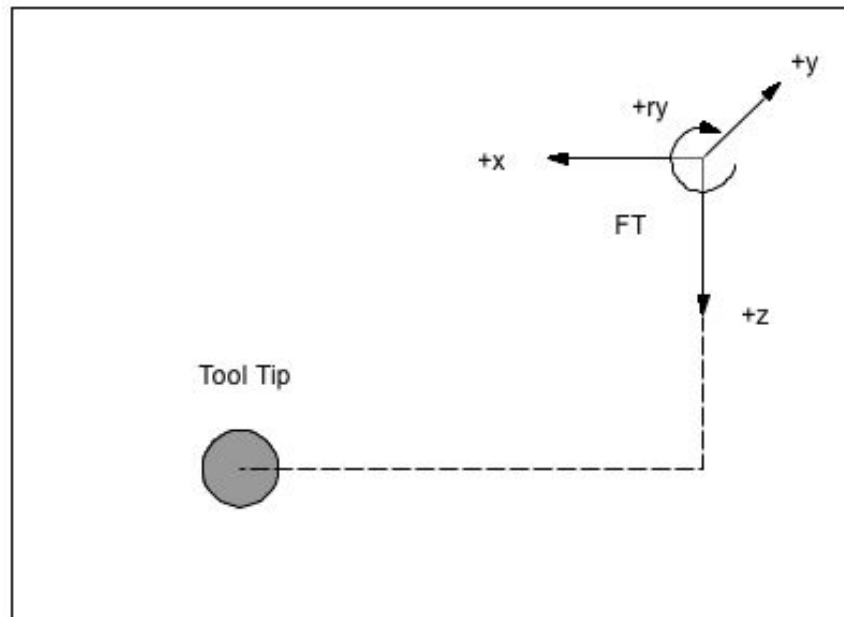


Figure 28. Smart Tool Force-Torque Sensor Axes.

6.2.5 Saw Experimentation, Function Definition, and Implementation

Function prototypes are then generated that use preliminary thresholds to determine reasonable bounds or collect data for further development. Experiments are then conducted to establish the parameters for the function prototypes as needed.

The first set of subtasks locates the pipe in space after the operator or higher level model-based robotic system has managed gross position and pointing at the task.

The prototype reactive function `bApproachH` is designed such that it moves toward the pipe in Cartesian space according to the orientation pose of the end effector (x - y - z) as established by the operator. The function looks for contact against the foot of the saw via

the force-torque sensor. Since the operator should have positioned the tool in reasonably close proximity, the function should time out and generate an error message if it goes more than a certain distance without making contact since this condition should never occur. A potential positioning error bubble of greater than 25mm should be expected. Contact should occur in all instances on the foot of the saw. A reasonable contact threshold should be established.

For this function the manipulator is divided into two planar manipulators to recover decoupled end-effector orientation—the global vertical x-z plane and the global horizontal x-y plane. End-effector yaw is a composite of shoulder azimuth (joint0) and wrist yaw (joint4) in the x-y plane. Wrist pitch is a composite of shoulder pitch (joint1), elbow pitch (joint2), and wrist pitch (joint3) in the x-z plane. The workspace axes are defined such that +x is straight ahead from the robot towards the process piping mockup, +y is to the left, and +z is up.

The increments in the Cartesian motion axes are modified with the composite potential field created by the manipulator joint angles per the following equations:

$$x_{desired} = x_{start} + j(inc)(\cos(joint0 + joint4)) \quad (6.1)$$

$$y_{desired} = y_{start} + j(inc)(\sin(joint0 + joint4)) \quad (6.2)$$

$$z_{desired} = z_{start} + j(inc)(\sin(joint1 + joint2 + joint3 - .174)) \quad (6.3)$$

where:

j = loop increment fixed to the time out limit,

inc = fixed delta for each Cartesian axis to move,

and the joint values are as previously described. Note that joint5, wrist roll, and joint6, gripper, are not part of this function.

The 0.174 radians in equation 6.3 is a cumulative offset to position resolver errors that was identified experimentally by setting the pitch joints to zero and measuring the actual angle of each link with a digital level and the final end-effector orientation. While this error may be partially due to compliance in the arm joint actuators, the joint resolvers are not installed with great accuracy as the manipulator used in this work is designed for joint-by-joint level teleoperation where such calibration is not of concern. Joint zero reference positions were also checked with the manipulator holding the tool at full extension; the additional error was only 0.1°.

Approach reads the force/torque sensor to look for contact based on a threshold value and will terminate on either contact or after a time limit is reached. While all axes are read, the dominant axis is the x-axis of the force/torque sensor that aligns with the longitudinal axis of the tool where contact is made. On completion control is passed to the next function in the sequence.

Once contact is made the saw should back off from the pipe to clear contact to prevent binding of the saw foot on the pipe and to permit the force-torque sensor to be used to find the pipe vertically. Contact should be minimized, and a reasonable distance should be defined. The prototype function is called bBackH.

The prototype function bApproachV is designed such that, given that the tool is already aligned and in close enough proximity to the pipe so that the blade will make contact, a downward vertical motion (-z) is used to locate the pipe vertically using the force-torque sensor. Force cannot be excessive, or the blade will be damaged. A reasonable contact threshold should be established. bApproachV is a variant of bApproachH.

The saw blade will bind if it is started while in contact with the pipe with any appreciable force. Therefore, a standoff should be created to eliminate contact with the pipe so that the cutting operation may begin. Contact should be minimized, and a reasonable distance should be defined but is not critical. This functional is labeled bBackV.

Before cutting, the saw should be made as level as practical to provide for a perpendicular cut on a horizontal pipe. Given the kinematics of the manipulator, the wrist roll joint is accurately used as the angle sensor for this task. bWristR levels the wrist roll joint.

The next major task set is to cut the pipe. This requires turning on the saw, monitoring the cutting process as the saw moves in the Cartesian $-z$ direction, and turning off the saw when done. The function is labeled bCutS and the details of the cutting process are established by examining forces and torques during cutting.

The final major task is to clear the pipe cut task so that control may be returned to the operator or higher level system. This requires a motion roughly the opposite of the original horizontal approach motion bApproachH. There is no significant need for sensing since the saw should return roughly to the starting point at the beginning of the automated telerobotic task, and it is known to be clear since that is where the operator initially positioned the tool. The prototype function is labeled bRetractS.

The equations of motion for bRetractS are as follows:

$$x_{desired} = x_{start} - j(inc)(\cos(joint0 + joint4)) \quad (6.4)$$

$$y_{desired} = y_{start} - j(inc)(\sin(joint0 + joint4)) \quad (6.5)$$

$$z_{desired} = z_{start} - j(inc)(\sin(joint1 + joint2 + joint3 - .174)) \quad (6.6)$$

6.2.6 Testing to Establish Saw Thresholds and Control Approaches.

Table 4 summarizes the results of developmental testing to determine thresholds for the various functions. Relevant implementation notes follow the table.

Table 4. Reciprocating Saw Event Tabulation.

Function Name	Action	Event	Variable(s)	Threshold
bWristR	Start via sequencer	Function call	N/A	N/A
bWristR	Level saw blade	Terminate at joint value	Wrist roll position	= -1.604185
bApproachH	Start via sequencer	Function call	N/A	N/A
bApproachH	Move to pipe horizontally	Terminate on threshold	Force-torque sensor fx-axis	> 30N
bBackH	Start via sequencer	Function call	N/A	N/A
bBackH	Back off horizontally	Terminate on force + coast	Force-torque sensor fx-axis	> 20N
bApproachV	Start via sequencer	Function call	N/A	N/A
bApproachV	Move to pipe vertically	Terminate on threshold	Force-torque sensor ry-axis	> .5N-m
bBackV	Start via sequencer	Function call	N/A	N/A
bBackV	Back off horizontally	Terminate on torque + coast	Force-torque sensor ry-axis	< 0.0
bCut128S	Start via sequencer	Function call	N/A	N/A
bCut128S	Motion to cut pipe	Closed loop	Force-torque sensor ry-axis	10N-m, P+F
bCut128S	Log contact	Store contact	Force-torque sensor abs(ry)	> 1N-m
bCut128S	Log rise of first peak	Store trigger & set variables	Force-torque sensor abs(ry)	> 10N-m
bCut128S	Stop cutting when done	Terminate on torque + coast	Force-torque sensor abs(ry)	< 1N-m
bRetractS	Start via sequencer	Function call	N/A	N/A
bRetractS	Extract saw	Count limit	Time via counts	Time = 8s

bWristR

The level position was measured experimentally under joint level control with the wrist in a horizontal position establishing a target value for the function action of -1.604185 radians. This is different from the expected value of -1.570796 radians. The difference is due to vendor placement tolerances of the position sensor and reinforces the need to validate sensor and system performance experimentally. bWristR uses a calculated quintic trajectory equation starting from the initial arbitrary teleoperated position to the desired indicated “level” position using the manipulator joint controller to close the loop on position.

bApproachH (find the pipe horizontally in space)

Force, torques, and manipulator Cartesian positions are collected in a data file that also records start/terminate times for the function. A typical plot of contact forces and torques is shown in Figure 29. As previously mentioned, the most practical axes for event monitoring would be the F_x force axis or the R_y torque axis. Since F_x indicates the larger value that would be less subject to noise, it is selected for the variable to use for the threshold.

Threshold value determination is somewhat subjective. In this case a firm contact to the pipe was desired to avoid contact noise and uncertainty. After multiple trials, 30N was selected such that as soon as the magnitude of F_x is greater than 30N, the function terminates on the next loop and passes control on to the next function.

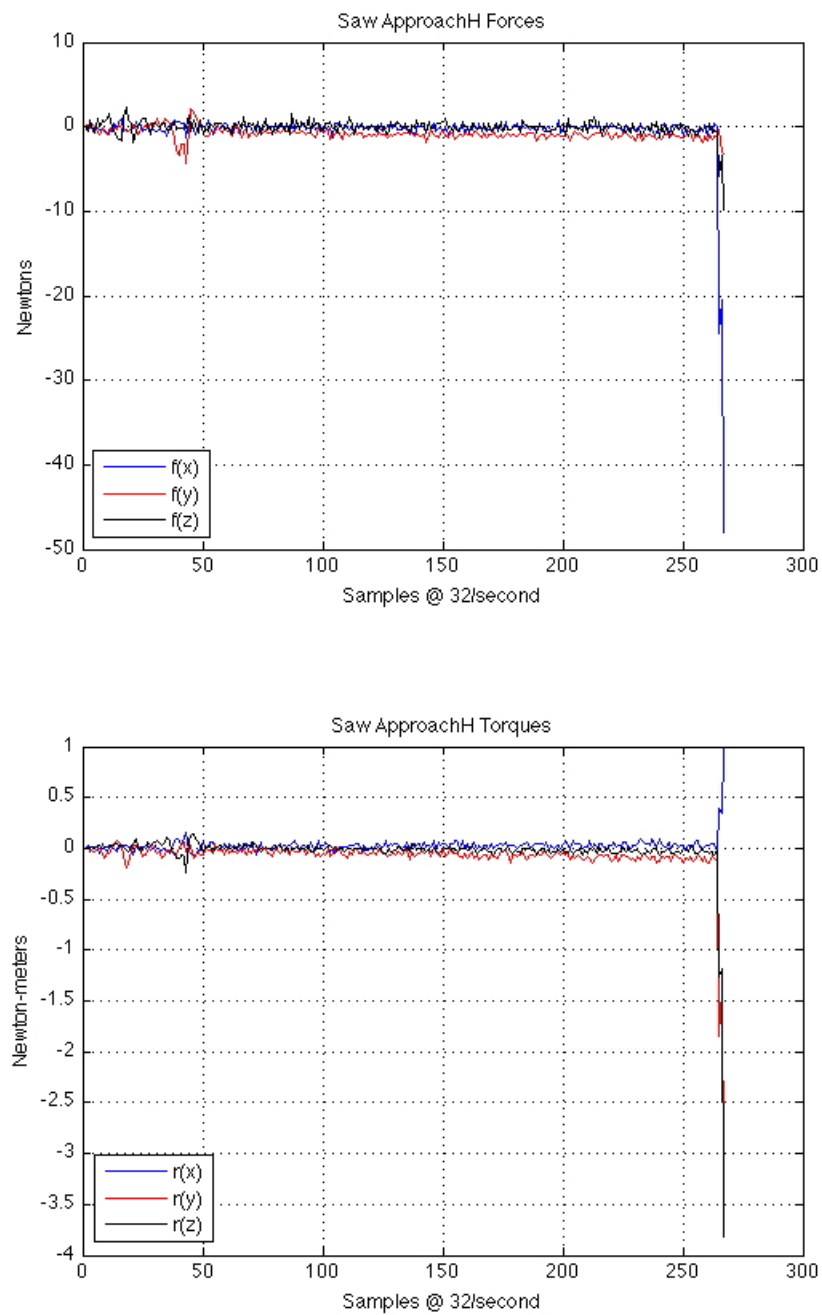


Figure 29. Sample *bApproachH* Plot of Forces and Torques.

bBackH

Once contact is established, the desire is to move back along the approach vector away from the pipe to a noncontact standoff distance so that the pipe may be located vertically in the task space without interference or distortion from existing contact. Standoff also facilitates cutting by removing a potential for the foot of the saw to bind on the pipe during the cutting process and corrupting force-torque sensor values. The equations of motion are the negative of the approach equations:

$$x_{desired} = x_{start} - j(inc)(\cos(joint0 + joint4)) \quad (6.7)$$

$$y_{desired} = y_{start} - j(inc)(\sin(joint0 + joint4)) \quad (6.8)$$

$$z_{desired} = z_{start} - j(inc)(\sin(joint1 + joint2 + joint3 - .174)) \quad (6.9)$$

The goal is to break contact and move to an approximate standoff clear of the pipe. This is accomplished by monitoring the Fx force-torque axis to a threshold value. However, the force-torque sensor is initialized while in contact with the pipe, giving the sensor a starting preload (offset). To achieve an approximate standoff from the pipe, motion is given a momentum “coast” such that it continues to move a small distance after reaching the threshold. Since it was found that the final Fx value could vary substantially between approximately 25N to more than 60N, 20N was selected as the threshold value ($F_x > 20$). On threshold trigger, the simulated momentum coast provides for an additional free space standoff of less than 13mm, depending on how far the force continues above the 20N threshold. Actual distance is not significant; only that contact is cleared. One data set for bBackH is shown in Figure 30. There is significant distortion of the forces and torques as the manipulator moves to clear contact.

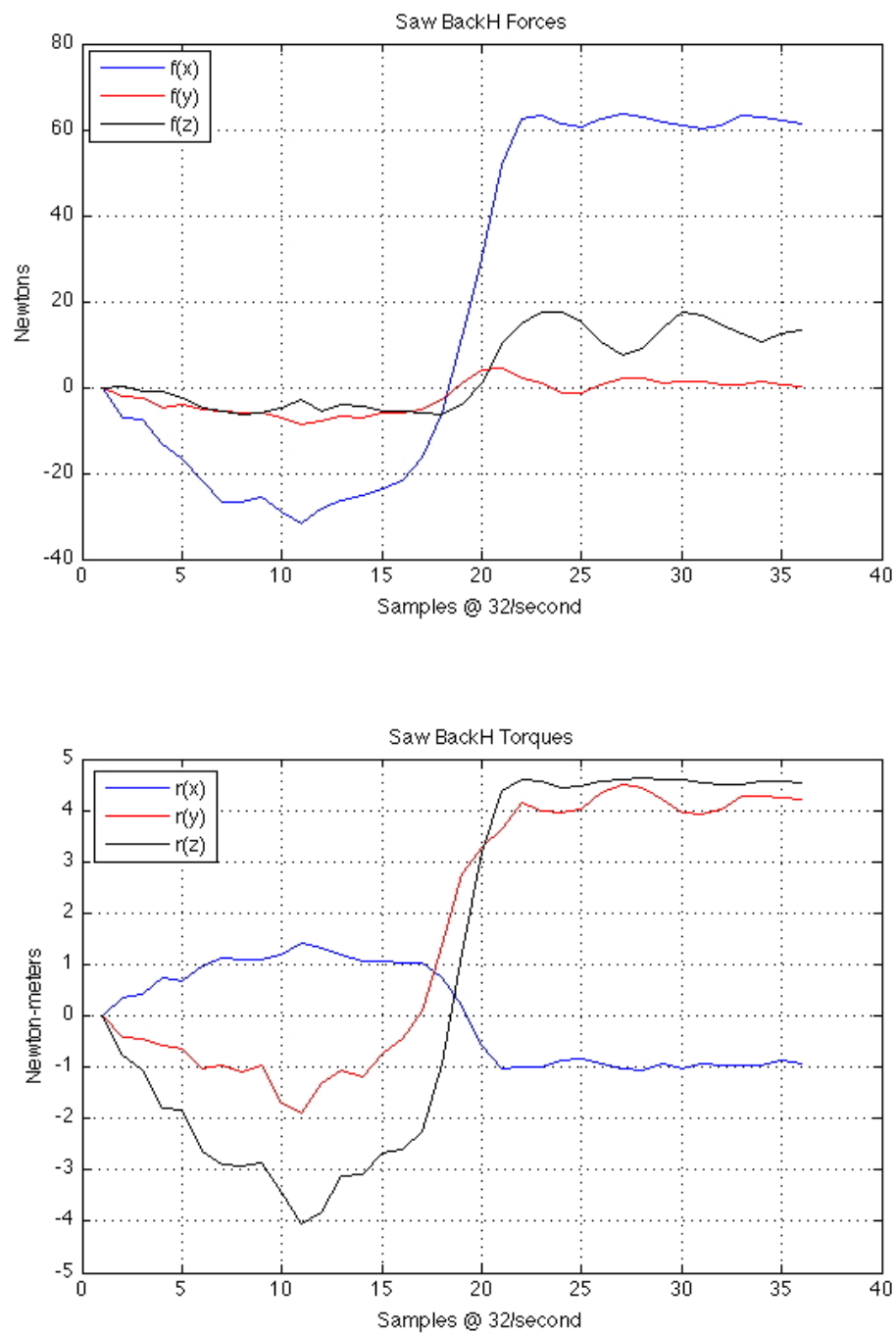


Figure 30. Sample *bBackH* Plot of Forces and Torques.

bApproachV

After bApproachH has located the pipe horizontally, bApproachV locates the pipe vertically. Given the amount of standoff provided by bApproachH, the saw blade is guaranteed to act as a finger to contact the pipe when the tool is moved down in the manipulator base frame z-axis. The behavior terminates upon contact threshold. From multiple tests, it was determined that the Ry force-torque sensor axis was most appropriate and that a threshold of .5N-m ($R_y > .5$) would succeed in all cases. Force-torque data for one instance of bApproachV is shown in Figure 31.

Since a low threshold value was used, the loop increment motion rate was decreased to 0.1mm. The equation of motion for the single Cartesian axis move is simply:

$$z_{desired} = z_{start} - j(inc / 4) \quad (6.10)$$

bBackV

bBackV moves back along the vertical approach vector away from the pipe to a non-contact standoff distance so that the saw blade will not bind on startup. Since contact was established by Ry in bApproachV, Ry is used as the control in bBackV. As in bBackH, the force-torque sensor is initialized with a contact preload that must be reflected in the threshold value. Also as in bBackH, a momentum coast is used after the threshold has been reached to create a standoff from the pipe of less than 4mm. Inspection of multiple runs revealed that $R_y < 0.0$ would reliably terminate the behavior. Sample bBackV data is shown in Figure 32.

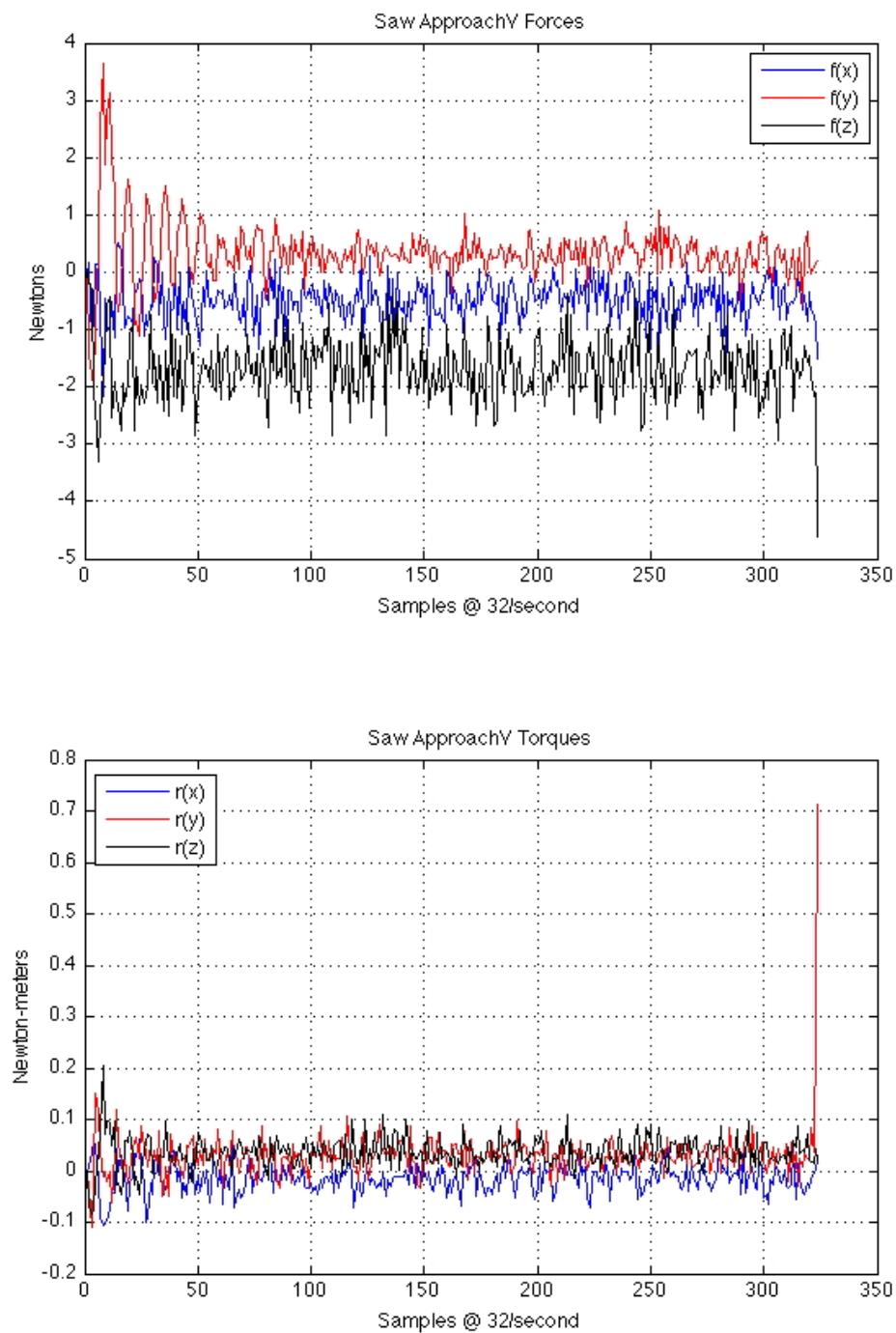


Figure 31. Sample *bApproachV* Plot of Forces and Torques.

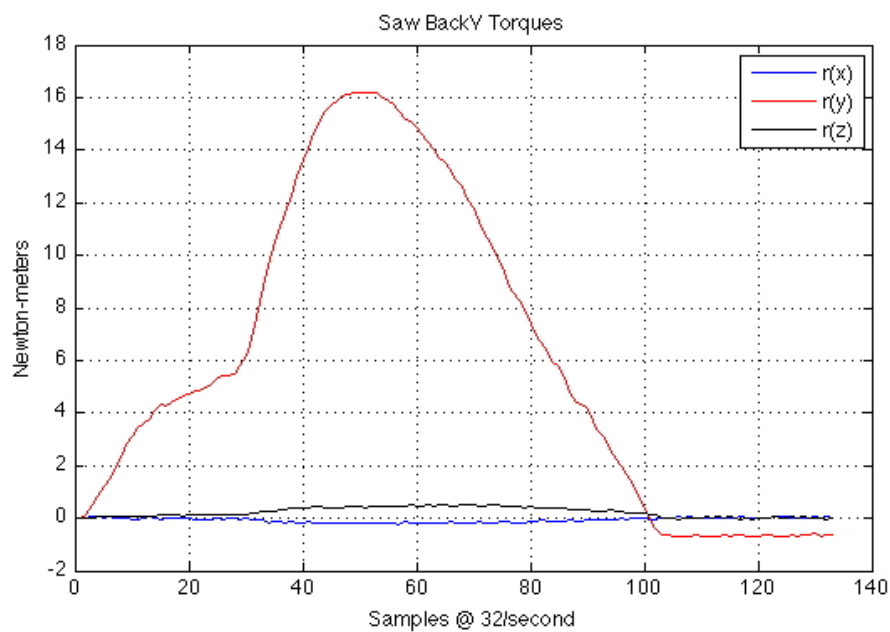
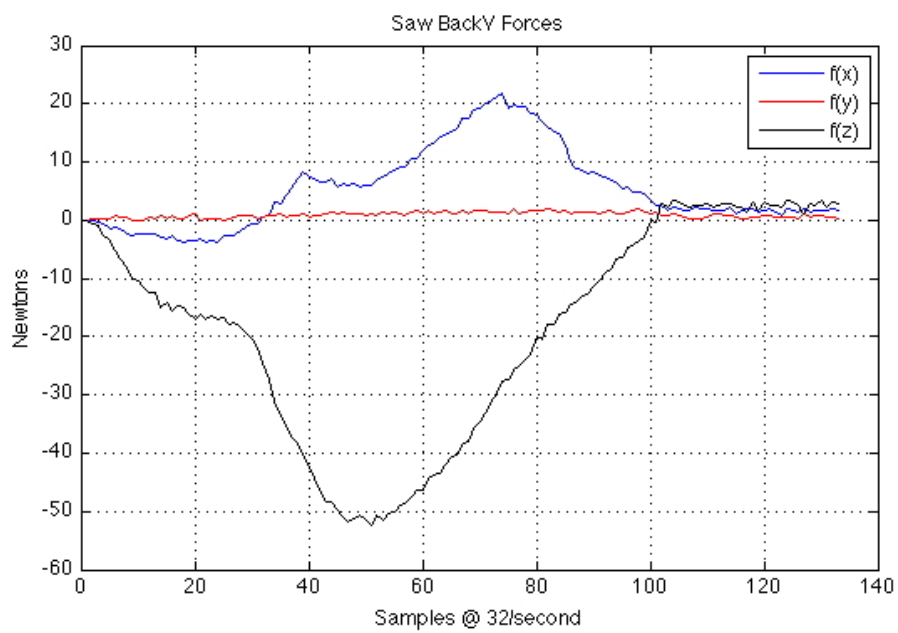


Figure 32. Sample bBackV Plot of Forces and Torques.

There is significant distortion in all axes of force and torque as the manipulator moves vertically away from the pipe. This is due to compliance in the manipulator elbow joint aggravated by the weight of the tool package. However, the value of R_y settles to the negative value of the initial R_y axis preload, permitting the aforementioned $R_y < 0.0$ threshold.

The motion increment for `bBackV` is the same as for `bApproachV` and the single axis equation of motion is:

$$z_{desired} = z_{start} + j(inc/4) \quad (6.11)$$

In summary, `bBackV` executes a Cartesian move in the manipulator base frame +z direction. An event generated when $R_y < 0.0$ terminates the function after a momentum coast on the order of 4mm.

`bCut128S`

`bCut128S` is the core reactive function that actually cuts the pipe. The prototype of this function used a time-based position trajectory to experimentally define a tool process signature of the cutting process based on cutting forces. It collects force-torque data at 128 Hz to ensure that sampling occurs at greater than twice the saw reciprocating frequency. The equation of motion for testing purposes is as follows:

$$z_{desired} = z_{start} + j(inc/32) \quad (6.12)$$

The forces and torques from a sample time/position-based cut are shown in Figure 33. It is immediately obvious that the sensor signals are unusable as is for control or monitoring. Since the primary cutting value should be offered by the R_y axis of the force-

torque sensor, a filter is applied to that axis according to the following equations in an attempt to recover useful data:

$$ryFilt_n = \left(\frac{ry_n}{128} \right) + \left(\frac{127}{128} \right) ryFilt_{n-1} \quad (6.13)$$

$$ryFilt_{n-1} = ryFilt_n \quad (6.14)$$

Two examples of resulting data are shown in Figures 34 and 35. Although the magnitudes can vary widely and there is significant variation in the details of the waveform, there is a distinct signature to the pipe cutting process that can be used to determine progress through the pipe and to determine when the cut is done. This information is used to regulate the bCut128S reactive function.

bCut128S uses filtered measured Ry axis force-torque sensor readings (ryFilt) to control motion in the manipulator's base frame z-axis to cut the pipe. The selected position + force (P + F) controller is bounded such that the rate of z motion varies from approximately 6mm/second – 19mm/second centered about a 10N-m controller set point. The P + F control is not designed to tightly control the force of the saw blade on the pipe since that would mask the tool process signature and since it is not practical given the control architecture bandwidth. Rather, it is designed to protect the saw blade and to provide faster motion when moving in free space in order to shorten the task. The lower bound is maintained to avoid damage to the saw blade due to excessive force; the upper bound provides higher velocity motion in free space and prevents premature trigger of terminating thresholds during contact.

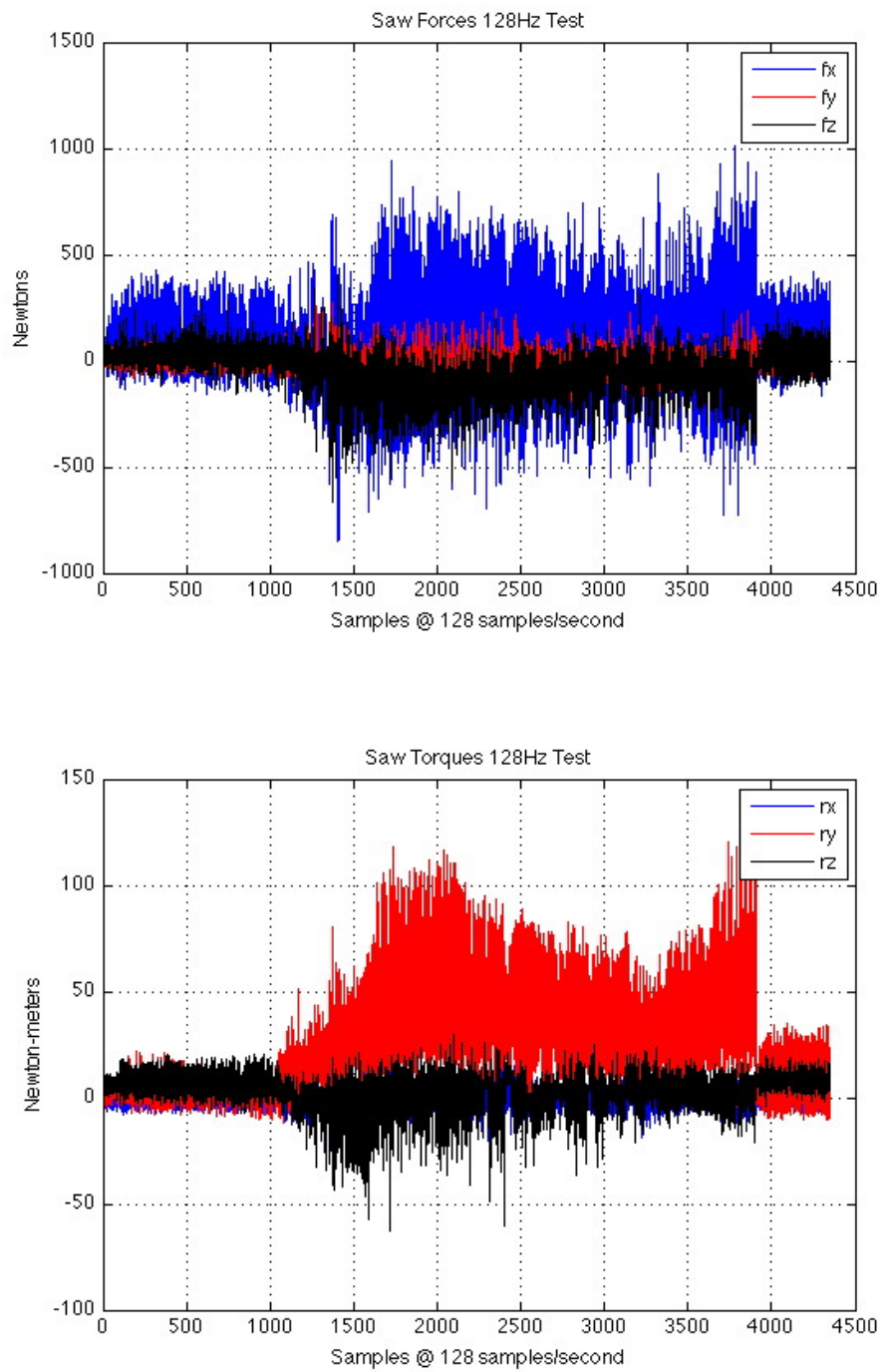


Figure 33. Unfiltered Cut Forces and Torques.

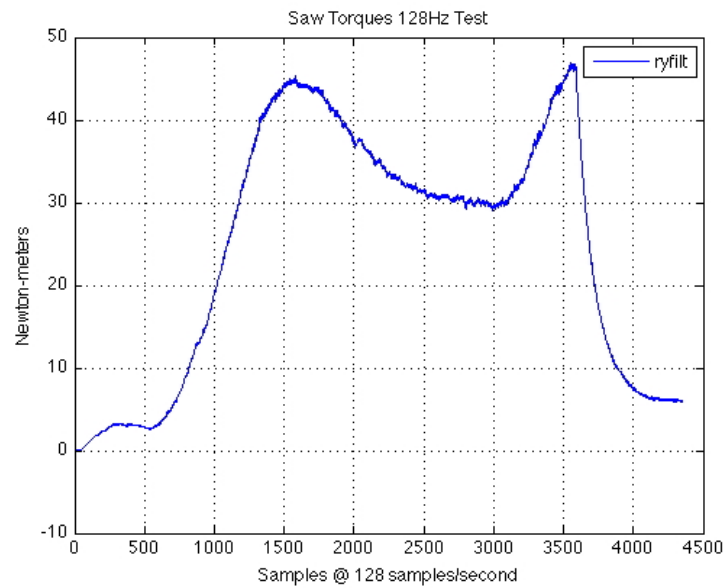


Figure 34. Example 1 Filtered Ry.

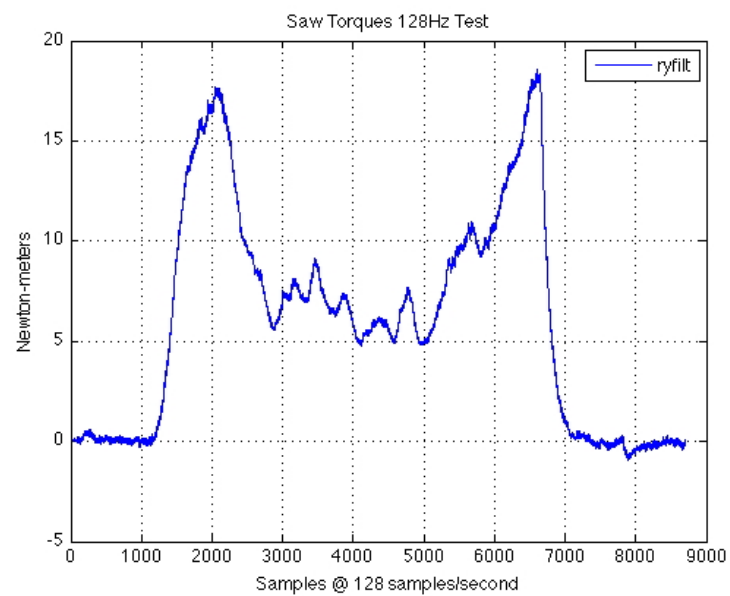


Figure 35. Example 2 Filtered Ry.

Along with the primary cutting action of the function, multiple events are used to monitor and terminate the process. When the magnitude of the absolute value of the Ry-axis of the force-torque sensor (ryFiltAbs) reaches 1N-m, this is logged as first contact with the pipe and is stored for data analysis. This is a data analysis event and not a control event. ryFiltAbs is used to ensure that any spurious negative values, which are rare but did occasionally occur in testing, would not excessively lower the value of the filtered signal.

When the value of ryFiltAbs reaches 10N-m, the pipe cut signature is rising to its first force peak, signaling the major portion of the cut. If ryFiltAbs drops below 10N-m after this event, a simulated momentum/coast of 1 second is initiated to carry through any oscillations generating low values of the controlled variable that may occur during cutting and while the P+F controller is accelerating to maximum velocity to increase the cutting force. Whenever ryFiltAbs rises above 10N-m, the momentum variable is set back to maximum.

When the value of ryFiltAbs drops below 1N-m and when the 1 second momentum/coast has expired to verify that the cut actually is done and that the low value is not due to oscillation during cutting, the behavior terminates and logs end time.

bRetractS

The motion executed by bRetractS is an incremental Cartesian motion in the manipulator base frame x, y, and z-axes in the negative direction of the approach vector established by the end-effector pose. Since the saw blade has vertically cleared the pipe as part of the cutting operation, no z-axis motion is necessary. bRetractS is specifically a ballistic function, meaning that it has no local task space sensor feedback. It executes a quintic trajectory at a specific rate for a fixed time and then terminates by returning control to the operator.

6.3 Removing a Bolt With a Powered Socket Tool

6.3.1 Task Definition

The second task selected is to remove a bolt from a process assembly. The key concern and motivation for automating this task is to limit the forces applied so that the tool, manipulator, and task components are not damaged. The mockup available for this dissertation, shown in Figure 36, is based on remote maintenance guidelines and uses captured cone head bolts that have a 30° taper on extended heads. The bolt on the process mockup is 23.8mm (standard 15/16-inch) in size; the tapers on the bolt head and the socket permit a misalignment of about 12.7mm inch.

The cone head bolt has a capture mechanism such that the bolt is loosely contained when removed; it can drop about 10° when the unbolted bolt is extracted to its maximum travel of 50.8mm (2 inches), but it will not fall out. The bolt must be extracted at least 15.9mm (5/8-inch) to be considered loosened.

For the process mockup, the bolts are on a 101.6mm (4-inch) diameter bolt circle with three bolts separated by 120°. A 31.7mm (1¼-inch) outside diameter stainless steel pipe comes out from the flange perpendicularly and turns right 90°, coming within 12.7mm (1/2-inch) of two of the three flange bolts (see previous Figure 18), restricting access to these bolts and occluding view of the bolts, depending on the ability of the manipulator to be positioned for disassembly.

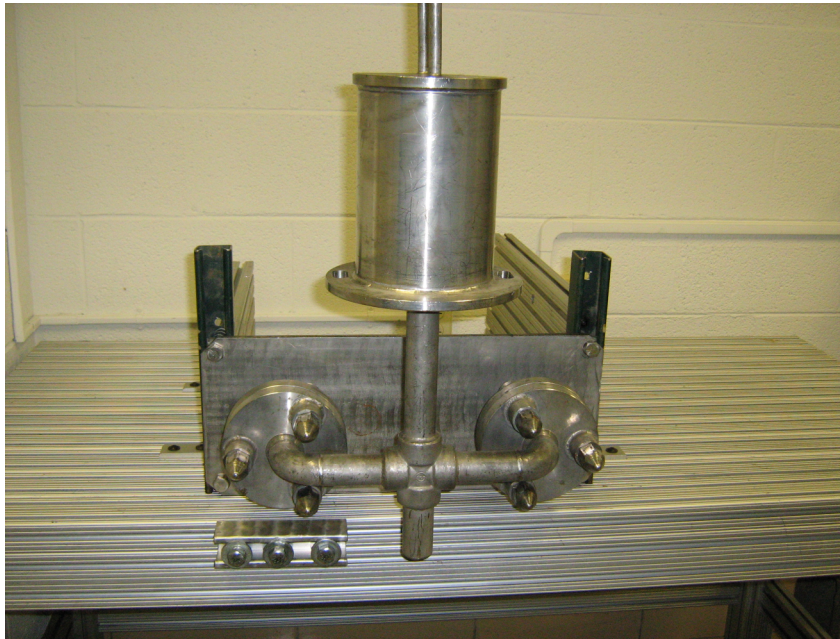


Figure 36. Disassembly Mockup.

6.3.2 Tool Selection and Description

Tools for this task may involve electric or pneumatic impact wrenches, motorized socket tools based on drills, and even hand tools though remote hand tool use is fatiguing and not time efficient. Given that the purpose of this work is to demonstrate concept validity for smart tooling, a motorized socket tool with an appropriately sized socket is selected. For this work, a standard 3/8-inch electric drill fitted with a standard 1/2-inch socket drive and modified to provide remote actuation is shown in Figure 37 prior to fixturing for remote use. Specifications for the socket tool are collected into Table 5.

The socket smart tool is shown in Figures 38 and 39, assembled with grasping block and force/torque sensor. The force/torque sensor measures contact loads and operating torques. Sensor signals and power are also routed back to the control computer through a

bundled cable. Much of the cabling and interface is common with the saw tool. As completed, the mass of the smart socket tool is 11.48kg.

While the saw uses a simple on/off relay controlled by the smart tool electronics interface at the computer, the socket tool requires additional control at the tool itself to change direction. This was the preferred solution over bringing a much larger bundle of wires back to the electronics interface. (Cabling handling is always a significant and problematic issue with remote tooling.) At the design phase it was not known that changing direction would not be a significant issue for capturing the socket, but the capability facilitated tightening as well as loosening bolts.

Contact in the forward direction of the tool is afforded by force in the $-F_x$ direction and torque in the $-R_y$ direction (negative moment about the Cartesian y-axis since the tool is offset from the sensor face plate) of the force/torque sensor. Experimental testing showed that force in the $-F_x$ direction was sufficient to indicate contact.



Figure 37. Electric Drill for Socket Tool.

Table 5. Socket Tool (Drill) Specifications Summary.

Characteristic	Specification
Tool body length	222mm
Tool length w/ socket	323mm
Tool width	67mm
Socket dimensions including drive	15/16-inch: 30mm outside diameter by 67mm long 3/4-inch: 29mm outside diameter by 67mm long
Mass	1444g
CG	121mm back from tip of drill chuck
Location for fixturing	89mm to 191mm back from tip of drill chuck
Power	120VAC, 264W, universal motor, 7.5n-m
No load speed	1200 rpm maximum



Figure 38. Smart Socket Tool.

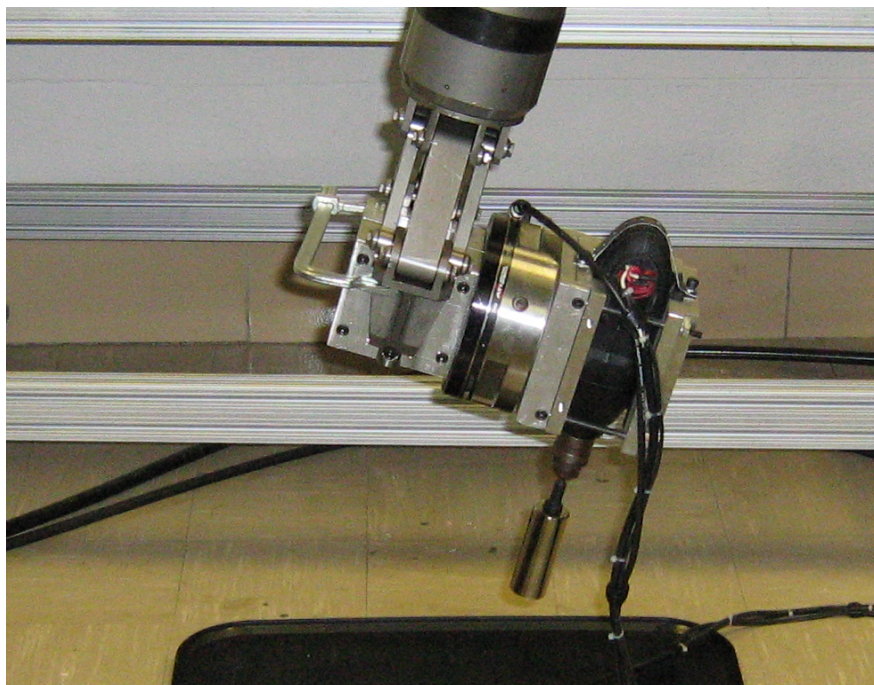


Figure 39. Smart Socket Tool Mounted in Gripper.

6.3.3 Subtask Definition

As before with the reciprocating saw tool, an assumption is made that the operator can deliver the tool tip to within reasonable proximity of the target bolt head while also pointing the tool tip towards the intended target. The task problem (motivation for automation) is to limit forces on the tool to prevent damage.

As with the reciprocating saw tool, the subtasks with notable events are defined and outlined for experimental development.

The first task is to find the bolt head in space.

Approach to contact according to the pose of the end effector. (event = contact)

The next task is to undo the bolt.

Turn on the motor.

Monitor motion to determine if the bolt is adequately undone. (event = relative motion)

Turn off the motor.

The final task is to clear the task to return control back to the operator.

Move clear of the bolt/process assembly.

Return control to the operator.

6.3.4 Sensor Selection

Available sensing will be considered to be the same as for the reciprocating saw. Manipulator joint sensing provides a type of Cartesian global position system. The 6DOF force-torque sensor provides contact and force management information.

6.3.5 Socket Experimentation, Function Definition, and Implementation

The prototypes functions requiring experimental development are listed below.

The first task is to locate the bolt head in space. This can be done by moving forward along a vector defined by the pose of the end effector. This would involve motion in all Cartesian position axes (x-y-z). Motion should stop upon reaching a certain threshold preload of the socket on the bolt head. The function should time out and generate an error message if it goes more than a certain distance without making contact. If acquisition fails, the operator should be given another chance to reposition for a retry. The prototype function is a derivative of the saw approach function and is labeled bApproachB. The equations of motion are the same as for the saw function bApproachH. Contact threshold is the only notable difference between the two functions.

The next task is actual removal of the bolt. To do this the motor must be turned on. Forces and torques are monitored to determine task progress. Once complete, the motor is turned off. The prototype function is labeled as bUnboltB. Time-based operation is used to look for a characteristic signature.

The final major task is to clear the socket tool task so that control may be returned to the operator or higher level system. The best approach is to return roughly to the starting position of the entire task along the lines of the original approach vector. The exception is that a captured bolt will extend the required motion to clear the task. The prototype function is labeled bRetractB and is a minor variation on bRetractS.

Approach and retract functions run at 32Hz; the unbolt function operates at 128Hz.

6.3.6 Testing to Establish Socket Thresholds and Control Approaches.

Table 6 summarizes the results of developmental testing to determine thresholds for the various functions. Relevant implementation notes follow the table in the same manner as for the saw.

Table 6. Socket Tool Event Tabulation.

Function Name	Action	Event	Variable(s)	Threshold
bApproachB	Start via sequencer	Function call	N/A	N/A
bApproachB	Move to bolt horizontally	Terminate on threshold	Force-torque sensor Fx-axis	< - 40N
bUnboltB	Start via sequencer	Function call	N/A	N/A
bUnboltB	“Push back” on bolt	Terminate timed 2 second motion bursts on threshold	fabs(fxstop – fxstart) (both start and stop come from fxFilt)	>100N
bRetractB	Start via sequencer	Function call	N/A	N/A
bRetractB	Extract socket	Count limit	Time via counts	Time = 8s

bApproachB (find the bolt head horizontally in space given approximate alignment)

Force, torques, and manipulator Cartesian positions are collected in a data file that also records start/terminate times for the function. As previously mentioned for saw data collection, the most practical axes for event monitoring would be the x force axis (Fx) or the y torque axis (Ry). Since Fx indicates the larger value that would be less subject to noise, it is selected for the variable to use for the threshold. See Figure 42 in section 7.4 for a plot of bApproachB.

Similar to the reciprocating saw threshold, value determination is somewhat subjective. A firm contact to the bolt was desired to avoid contact noise and uncertainty and to ensure that the unbolting operation would be successful due to a firmly seated socket; however, excessive force that might cause binding during bolt removal needed to be avoided. After multiple trials, - 40N was selected such that as soon as the magnitude of Fx is less than - 40N, the function terminates on the next loop and passes control on to the next function.

bUnboltB

Force, torques, and manipulator Cartesian positions are collected in a data file that also records start/terminate times for the function. For unbolting, the most practical axes for event monitoring would be the x force axis (Fx) or the y torque axis (Ry), since the unbolting operation creates a “push back” force as it is backed out. Fx is chosen.

Threshold value determination required heavy filtering of Fx as with the saw tool. The same filter was used as expressed in equations 6.13 and 6.14. The terminating threshold was set to 1000N so that the loop would run on till manually ended. Start/stop forces were accommodated by the equation:

$$f_{abs}(f_{xstop} - f_{xstart}) > 100 \quad (6.15)$$

where:

f_{abs} is the absolute value of the function,

f_{xstop} is the final filtered pushback force at the end of the tool burst, and

f_{xstart} is the beginning filtered pushback force before the start of the tool burst.

After multiple trials, 100N was selected such that as soon as the magnitude of equation 6.14 is greater than 100N, the function terminates and passes control on to the next function. Actually any significant push back of the bolt as it was unscrewed was a good measure of success for the task. Ranges from 20N to 120N proved successful in indicating success. See Figure 43 in section 7.4 for a plot of $b_{UnboltB}$ unfiltered and filtered values.

$b_{RetractB}$

Mentioning the last function first, motion executed by $b_{RetractB}$ is an incremental Cartesian motion in the manipulator base frame x , y , and z -axes in the negative direction of the approach vector established by the end-effector pose. $b_{RetractB}$ is specifically a ballistic function, meaning that it has no local task space sensor feedback. It executes a quintic trajectory at a specific rate for a fixed time and then terminates by returning control to the operator. In testing it was found that retracting in all three Cartesian position axes often caused the socket to snag on the unbolted but captured bolt. This was addressed by eliminating the z axis motion in the retract function.

6.4 A Note on Expansion to Other Tools

Sensing requirement and reactive function development complexity is directly proportional to the complexity of the tool process. More complex tooling operations require more sensing and control. Note that the socket tool only required three functions

to meet its automation needs; however, the reciprocating saw required seven functions to meet its automation needs. Note also that many of these functions assemble repeatedly in minor variations, indicating that they may serve as primitives with which to build new tool controllers.

An impact wrench would use the same sequence of functions as the socket driver; however, it should be expected that the process “noise” thresholds and possibly the push back profile would be different. Drilling would use the approach and retract of the socket tool in conjunction with a process cut similar to the reciprocating saw.

A milling head cutter would be similar to a reciprocating saw in that it would require a horizontal and vertical approach. It would be different in that the cut motion is in a different plane and that cutting a uniform metal plate would not have the same signature that a cutting a hollow pipe would have, but the cut process could be managed in the same manner. Retract would most likely best be completed by raising the milling head out of the cut and then back as with the saw. Most of the functions in the sequence could be identical to those of the reciprocating saw with different threshold values. A band saw would use a simplified version of the reciprocating saw sequence and would not have the same difficulties with process noise.

The circular saw may be the most complicated D&D-type tool to control due to its need to prevent binding of the rigid blade in multiple axes while the cut progresses through the task object, as described in section 5.3. This tool sequencer would have all of the functions of the reciprocating saw but would also have to use command fusion to maintain orientation and position of the five axes that were not aligned with the direction of the cut in the task object.

While D&D power tooling has been the focus of this effort, this collection of function primitives could be expanded and applied to any power tooling and even cutting and

friction-based hand tools. It has merit wherever a process signature is created between the tool and its task object.

Chapter 7

Experimental Results

7.1 Discussion of Overall Telerobotic Reciprocating Saw Results

Data collection for validation of the telerobotic reciprocating saw task was accomplished by running 15 instances of the task on a horizontal pipe. Tool placement to the task target area was via a sequence of robotic moves to a targeted end point in task space. Positioning repeatability of the manipulator delivery system is known to be on the order of $\pm 6\text{mm}$ in the Cartesian x , y , and z axes. Pipe placement in the process rack was intentionally not precisely aligned for each incremental test with variations in vertical (Cartesian z) and depth/distance away (Cartesian x) task axes on the order of $\pm 6\text{mm}$. There was minimal attempt to fight the inherent variability in the task mockup or the tool placement as that was an opportunity to test the ability of telerobotic task execution to adapt to manipulator and task placement uncertainty.

For the 15 trials, successful completion of the cutting task was 100% with no faults. Experimental data is presented in Table 6 for the 15 trials. The functions are presented in each column with the maximum number of loops possible and the loop rate noted. None of the functions hit their maximum value indicating that all reactive functions terminated on sensor events and did not time out. The function `bWristR` is not included in the table since it executes a fixed 2-second closed loop trajectory to level the saw so that it is perpendicular to the horizontal pipe. (The wrist roll position sensor is used as the level sensor.) The function `bRetractS` is not included in the table since it is a time-limited (8s) ballistic function designed to extract the tool from the task area along a vector established by the end-effector pose so that the operator will not be concerned about trapping the saw blade in nearby piping or structures. However the fixed execution times of these functions are figured into the final telerobotic execution times noted in the last column. Total task execution times are computed from time stamps collected from the high-level

controller system clock initiating at the start of bWristR and terminating with the final time stamp on bRetractS.

For each function column, the highest count (longest execution time in green) and lowest count (shortest execution time in red) are noted along with an average function execution time at the bottom of the column. “Times” are noted in counts for most entries except for minimum, maximum, and average where execution times in seconds appear in parentheses.

The saw blade was inspected periodically looking for worn or broken teeth or any other damage to the blade. It was changed on test 11 as a precaution since several teeth had broken or acquired hardened debris. Prior experimentation had shown that the teeth would eventually wear to the point that cutting forces would increase significantly.

Table 7 provides additional detail to the internal workings of the bCut128S function. Except for minimum, maximum, and average execution times, the data is presented in counts from start with 128 counts per second in the control loop. “First Contact” indicates when the force/torque sensor reaches contact from the starting stand off of the saw blade from the pipe. “Cut Threshold Reached” indicates when the control point of 10N is reached on the cutting forces. “Cut Completed” is measured at the completion of the threshold rule conditions at the close of the bCut128S function and includes the time required for move to contact.

Note that First Contact and Cut Threshold Reached are paired; time to contact links to time to threshold reached. However, these two do not drive total cut completion time since the highest and lowest actual cut completion times do not follow from the highest and lowest values of the contact and threshold values. The last column “Total Actual Cut Time” is the completion time minus the time to contact.

Table 7. Reciprocating Saw Data.

	ApproachH	BackH	ApproachV	BackV	Cut128S	Total Time
Max Loop Count	320	64	640	320	12000	
Loop Rate	32 Hz	32 Hz	32 Hz	32 Hz	128 Hz	
Test #						
1	266	35	323	132 (4.13s)	7411	90s
2	245	35	294	124	7189 (56.16s)	89s
3	234 (7.31s)	36	244 (7.63s)	130	7376	87s
4	264	35	368	119	7244	90s
5	261	35	267	112	7588	90s
6	261	35	289	112	7504	90s
7	261	35	297	107	7539	88s
8	260	35	288	104	7762	93s
9	264	35	361	107	7397	93s
10	249	34 (1.06s)	244	100	7793	90s
11	258	36	344	95 (2.97s)	7670	92s
12	265	36	370	107	7616	94s
13	271	36	437 (13.66s)	108	7512	93s
14	266	37 (1.16s)	334	107	7934	94s
15	277 (8.66s)	35	309	104	8032 (62.75s)	91s
Average	260.13 (8.13s)	35.33 (1.10s)	317.93 (9.94s)	111.20 (3.48s)	7571.13 (59.15s)	90.93s

Table 8. *bCut128S Internal Performance Data.*

Test Run	First Contact Counts	Cut Threshold Reached Counts	Cut Completed Counts	Total Actual Cut Time
1	247	966	7411	55.97s
2	362	1019	7189 (56.16s)	53.33s
3	544 (4.25s)	1230 (9.61s)	7376	53.37s
4	224	870	7244	54.84s
5	265	948	7588	57.21s
6	238	835	7504	56.76s
7	268	880	7539	56.80s
8	354	993	7762	57.87s
9	262	822	7397	55.74s
10	223	867	7793	59.14s
11	200	1083	7670	58.36s
12	304	1093	7616	57.12s
13	179 (1.40s)	772 (6.03s)	7512	57.29s
14	356	1097	7934	59.20s
15	279	1056	8032 (62.75s)	60.57s
Average	287 (2.24s)	968.73 (7.57s)	7571.13 (59.15s)	56.91s

It was noted after testing that the pipe has a welded seam along its length as part of its manufacturing process. As the pipe was moved to facilitate additional cutting, it was typically rotated to facilitate motion in the pipe clamps. This randomly moved the location of the weld in the cut profile probably impacting cut time somewhat due to a change in hardness of the metal being cut.

7.2 Examination of Specific Saw Tool Representative Test Cases

The shortest and longest duration cut data files are examined for variations. The z-axis graph in each figure plots z motion vertically against counts horizontally. Counts translates to time with 128 counts/second. The vertical axis is expressed in inches according to what the manipulator controller generates. The second graph for each figure is moment in N-m about the force-torque sensor y-axis. Test 2, shown in Figure 40, had the shortest execution time. It took about 2000 counts (15.62s) to reach the first peak while cutting the upper section of the pipe with about 4600 counts (35.94s) between the two peaks. There is minor oscillation of force in the main body of the cut that is commonly seen. Note that the peak forces are about 22N-m and 25N-m, respectively. Test 15, shown in Figure 41, has significantly higher forces and much more oscillation during the cutting process. While minimal oscillation is indicated in the z-axis motion of Test 2, there is obvious distortion in the z-axis motion of Test 15. The oscillation appeared to significantly delay the rise to first peak that is the indicator of successful cutting through the top of the pipe. Actual peak-to-peak time is shorter despite the oscillations at 4100 counts (32.03s). Despite the variations, both end cleanly and in similar fashion. The forces for Test 15 range from approximately 64% higher for peak 2, to 68% higher for peak 1, and 92% higher for the mid section of the pipe.

Post-test examination noted that there was a slight but noticeable pitch angle to the saw blade in the saw. This angle was corrected as much as possible (the vendor mounting method does not adequately fix the blade angle), and another post-trial test was completed. Overall completion time dropped from 60.57s to 55.28s with no other changes. Saw blade condition is critical to time-to-complete performance.

Referring back to Table 2 and the column containing the expected tool process signature for the reciprocating saw, the actual signatures of Figures 40 and 41 resemble but are not exactly like the proposed profile and show variation even from the experimental data taken to determine thresholds. However, the control technique still worked at 100%. Even with the variations, initial contact, closing loss of contact, and transitions through thicker and thinner walled sections of the pipe may be discerned. The intended process signature proved valid for control.

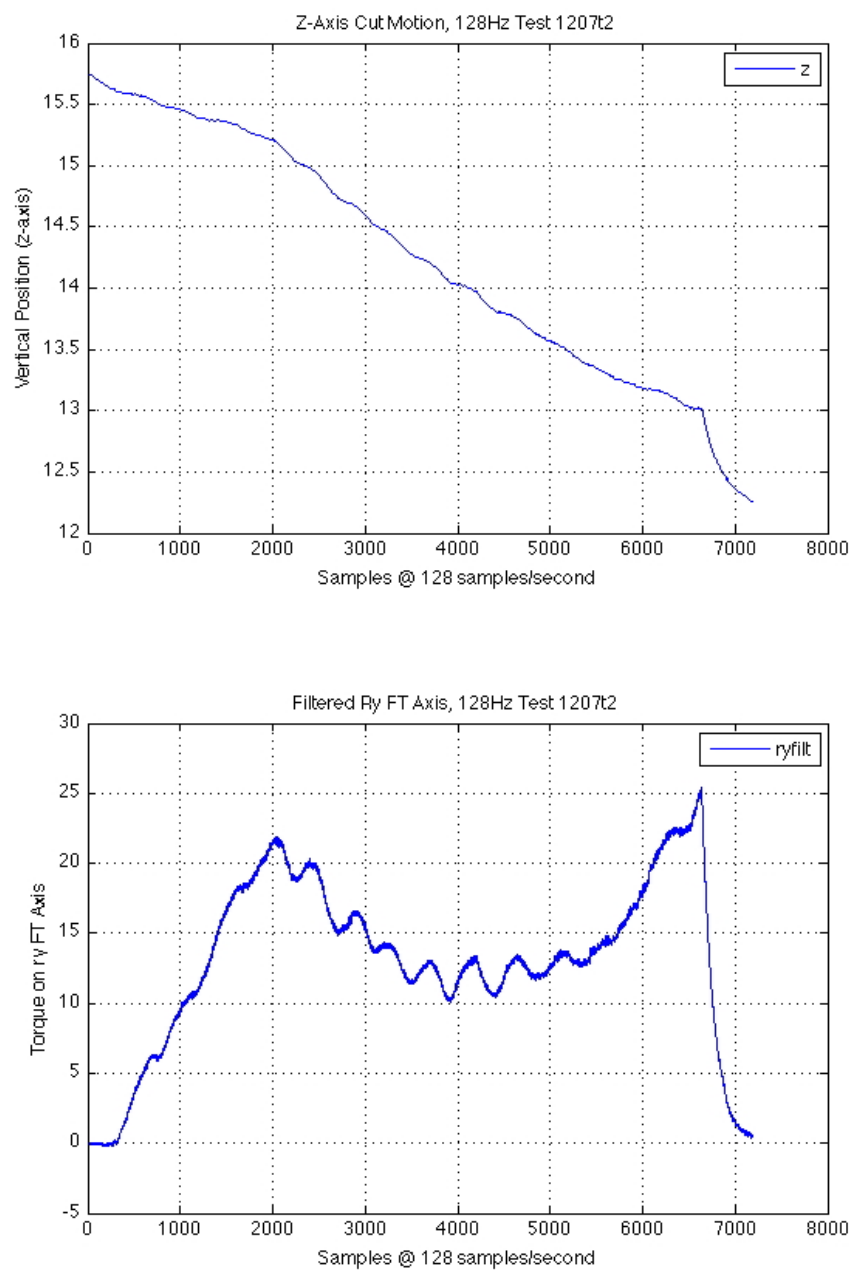


Figure 40. Cut Data From Shortest Duration Cut.

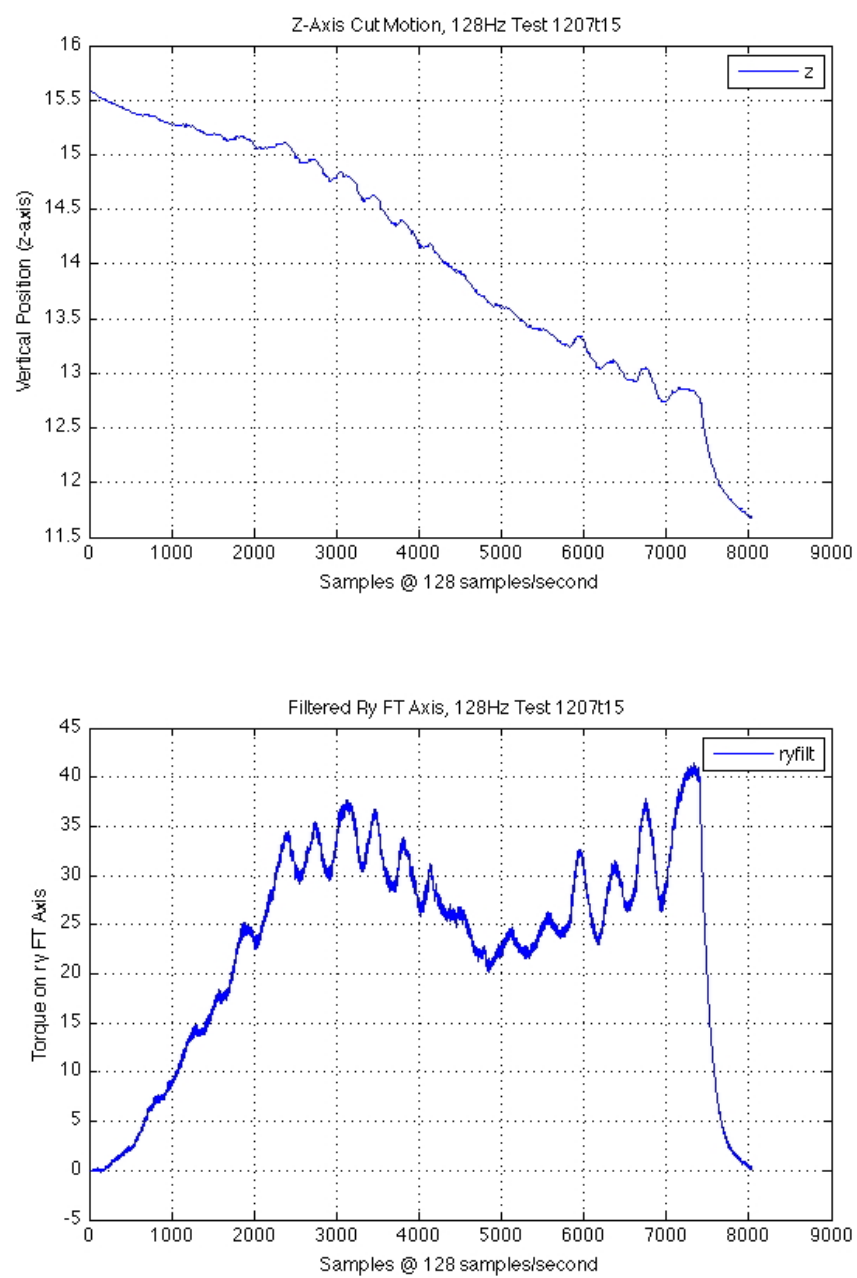


Figure 41. Cut Data From Longest Duration Cut.

7.3 Discussion of Overall Telerobotic Socket Tool Results

Data collection for validation of the telerobotic socket tool task was accomplished by running 20 instances of the task on a process mockup cone head bolt mounted horizontally. As with the saw tool task, tool placement to the task target area was via a sequence of robotic moves to a targeted end point in task space. Also as mentioned for the saw tool task, positioning repeatability of the manipulator delivery system is known to be on the order of $\pm 6\text{mm}$ in the Cartesian x, y, and z axes. Since the process module containing the bolt was rigidly mounted, its position in space was consistent and repeatable for all tests.

For the 20 trial runs, there were 16 successful completions and four failures. Success was defined as the bolt being loose enough in its captured bolt fixture to slide out to full extension by hand without twisting it. There were three types of failures including:

- Minor capture of the bolt by its last thread such that it was easily removed by hand with less than a 90° twist. The terminating threshold was triggered and operation completed. This should be considered a soft failure since after bolt removal a remote system could probably shake the component loose without further tool action. Quantity of failures = 2.
- Major capture of the bolt such that it was too tight to rotate by hand. The threshold condition was met and operation terminated normally. This is a hard failure. Quantity of failures = 1.
- Threshold value never reached despite moving bolt. Terminated by operator. This is a hard failure. Quantity of failures = 1.

Table 9 outlines the composite performance of bApproachB for a representative subset of eight of the 20 tests. If the threshold is never reached, the behavior would run for 320 counts or 10s while moving a distance of 127mm. All bApproachB functions triggered

successfully well before the counter limit. Thresholds of 40N and 50N for the Fx axis were tried in the series of tests with no noticeable difference in performance of the bUnboltB function.

The bUnboltB function runs in 2-second bursts and then checks for bolt pushback in the Fx axis to verify that motion has occurred. Although designed such that it could operate for multiple bursts, in actual operation, in all cases except the run-on failure requiring operator intervention, the function successfully terminated after one burst of the socket tool even given a wide range of examined thresholds. Therefore, an examination of counts and run time is not relevant for bUnboltB. The best measure of performance is the rate of success (16) /failure (4) out of the full number of tests (20) previously mentioned.

7.4 Examination of Specific Representative Socket Tool Test Cases

Example test cases are presented to more specifically illustrate individual function performance. Figure 42 shows the Fx axis event trigger upon reaching preload of 40N. Although other axes increase in force and torque, Fx is the axis that represents the preload on the bolt to prepare for removal. Figure 43 shows the actual unbolt process for all six axes of force and torque, with the main axis of interest being the Fx axis. Due to the process noise on this signal, filtering (fxfilt) is used to monitor the Fx axis of the force-torque sensor to determine push back into the manipulator system, indicating that the bolt has moved out during the unbolt operation.

Table 9. bApproachB Socket Tool Composite Results.

	Loop Counts	Motion	Time
Theoretical limit	320	127mm	10.0s
Actual low	26	10.32mm	.81s
Actual high	74	29.37mm	2.31s
Average	56.9	22.58mm	1.78s

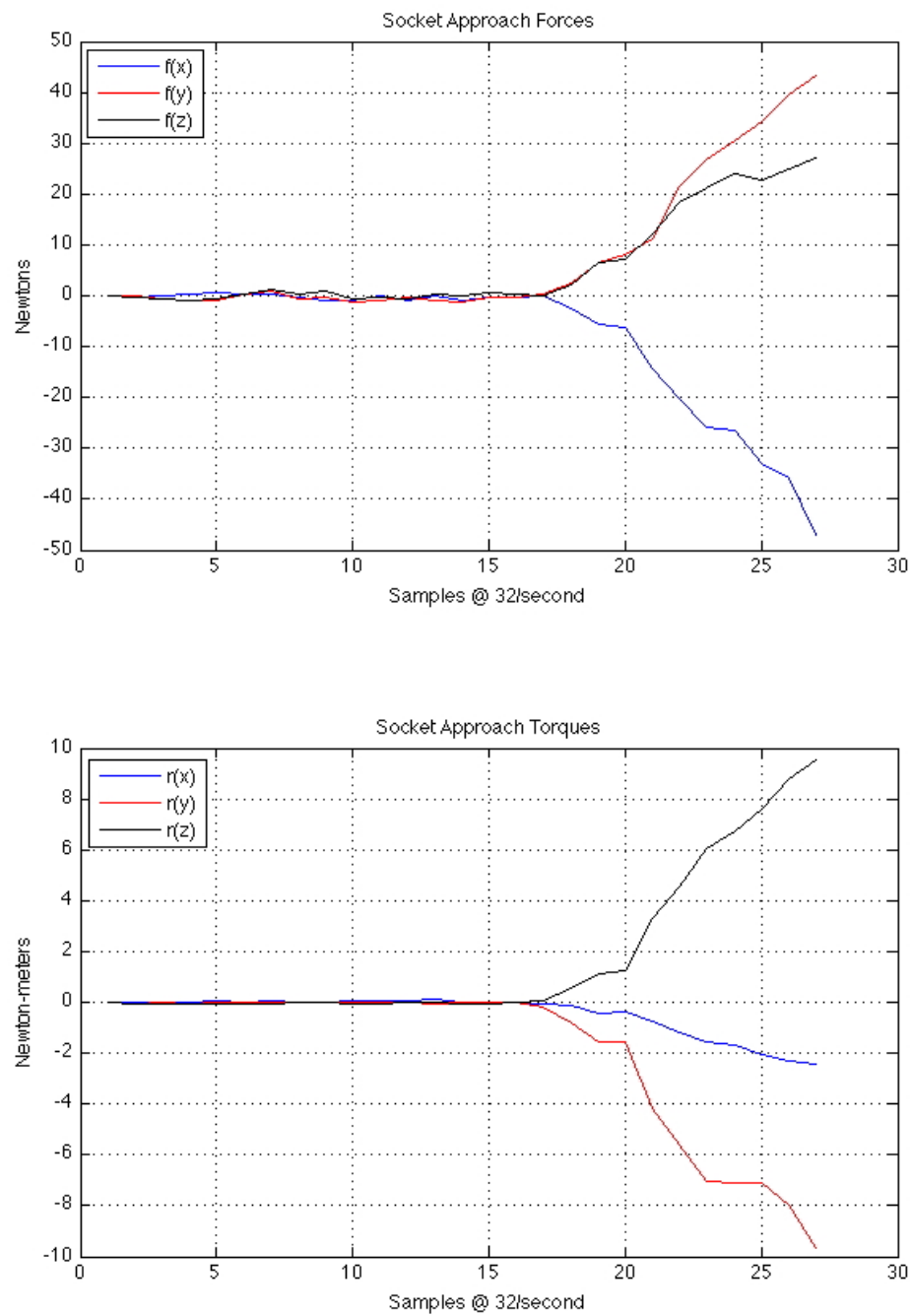


Figure 42. Sample Approach Forces and Torques, F_x Used for Event Monitoring.

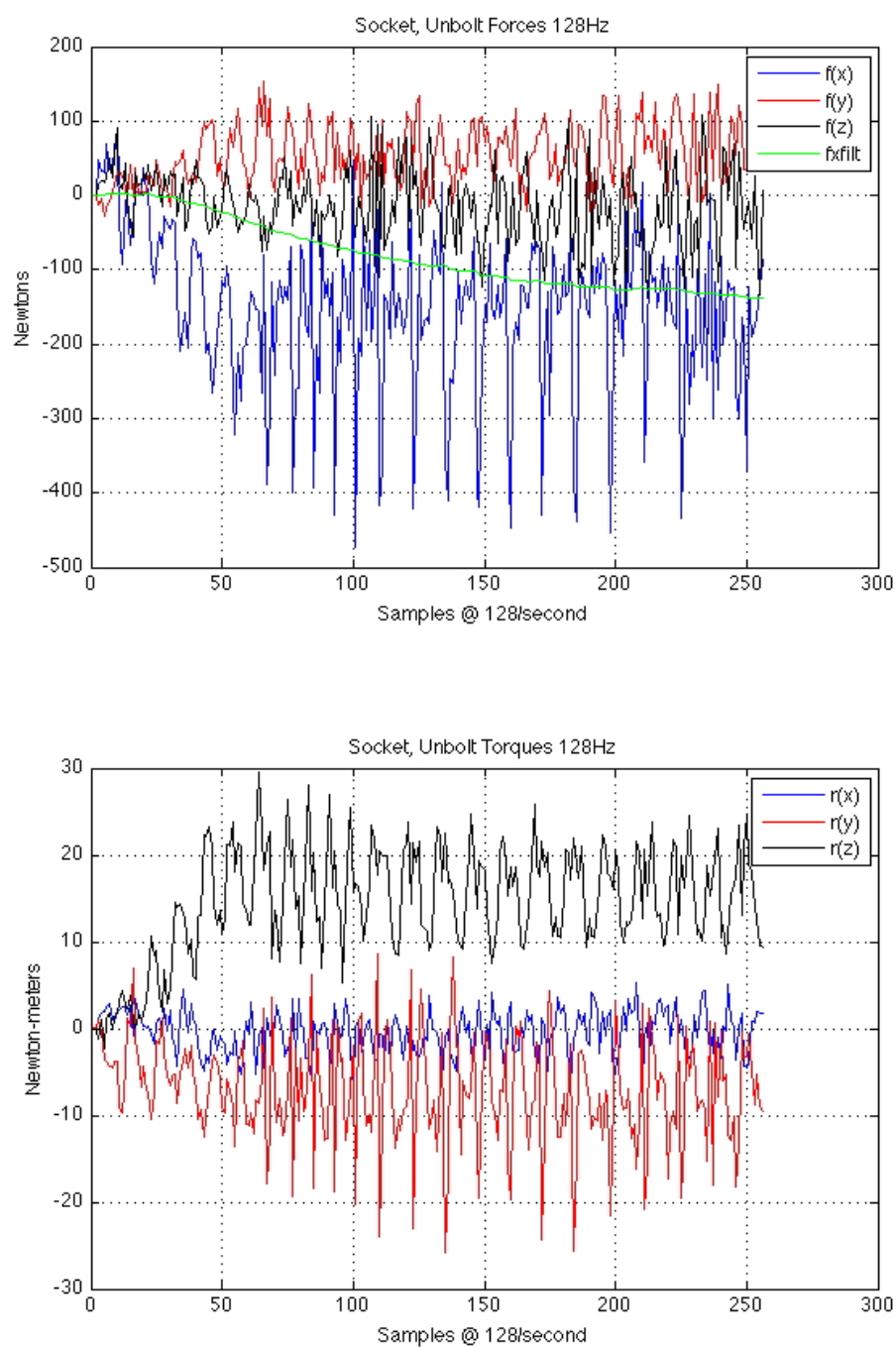


Figure 43. Sample Unbolt Forces and Torques, $fxfilt$ Used for Event Monitoring.

Chapter 8

Summary and Future Work

This work has examined the possibilities of integrating behavior-based concepts into teleoperation and robotics to provide efficient real world-usable telerobotic tooling assists. The concept was implemented and demonstrated for two tools common to remote D&D activities. In summary the basic approach works and has some merit but also has some limitations specifically related to the usefulness of behavior concepts.

8.1 Summary

As described in the previous chapter, both of the representative D&D tasks were implemented successfully. Performance of the saw task was 100% successful across the sample set. For the socket tool sample set, successful runs were completed 80% of the time with the given implementation.

Referring back to Table 2 D&D Tool Summary, column three Contact Signatures are verified to approximate expectations. In both cases the raw signals contain so much process noise that they are unrecognizable and unusable as is. However, heavy filtering is possible to discern usable profiles that resemble those found in Table 2. In neither case is the replication exact, but it is sufficient to work reliably.

Closed loop dynamic control using these signals would be difficult, but the reactive function-based approach achieved consistent successful results. The task decomposition technique derived from behavior-based concepts provided manageable subtasks that facilitated overall task completion.

One aspect of smart tooling is that it is expensive to implement due to multiple expensive sets of sensors. While a force-torque sensor was used for each tool implementation,

limiting the event and process axes to one or two axes indicates clearly that successful operation is possible with a reduced sensing set such as one or two load cells per tool that would reduced the cost of smart tool sensing roughly by an order of magnitude.

Another advantage to the approach presented in this work is that complicated kinematic or dynamics-based solutions are not necessary. Transformation of the force-torque sensor to manipulator kinematics was not even necessary. Each signal used was used independently of kinematic reference.

Most importantly the behavior-derived technique functioned as desired to calibrate an a priori task model to a point of execution on the task mockup target point. Task instance modeling was eliminated. The task type was “calibrated” to the location of the task object in space permitting reliable telerobotic task execution.

While successful, limitations were also found that made complete adherence to a behavior-based approach inappropriate for telerobotic use of power tooling. Tool tasks are inherently sequential in nature. Sequential behavior selection is considered the most primitive form and least desirable means of switching; however, it is most appropriate to telerobotic tooling. Also there are times when open loop robotic motions are the most efficient and practical means of task execution. These “ballistic” behaviors are accepted but discouraged in behavior-based approaches. These practicalities of implementation for telerobotic tooling reduce the “purity” of the behavior-based approach to more of an approach based on concepts derived from behavior-based techniques.

8.2 Review of Contributions

The fundamental contributions of this dissertation are summarized here:

1. The exploration and evaluation of behavior-based robotics for concepts to create a new methodology for integrating telerobotic tool control with positional teleoperation in the execution of complex tool-centric remote tasks such as those associated with remote nuclear operations. Successful experimental results with selected power tools and a full-scale telerobotics test bed have revealed the attractive combination of simple implementation and efficient/effective tooling operations.

This methodology provides a workable clear path to implementation relevant to the existing architectures of typical teleoperator systems while addressing tasks that are currently difficult to automate due to complexity and limited registration to actual task hardware. Once the first couple of tool tasks were programmed, it was quite obvious that this technique has created a set of primitives that may be assembled in different ways or with slight modification to quickly produce new automated tooling tasks. This work represents the first known application of these techniques to power tooling tasks.

2. The creation of a new tooling task modeling process that is general in nature and applicable to a wide range of power tools used in typical remote operations. This task type modeling can replace task instance modeling to reduce and simplify the application of the new behavior-based methods to complex telerobotic tooling applications. It was demonstrated that the task type model could be reliably encoded in a sequence of simple behavior-like reactive functions, thereby alleviating the need for extensive a priori generation of a task instance model for each task execution. This reduces the modeling time needed for individual task automation making telerobotics more time competitive even with proficient operators.

3. The generation of specific characteristic tooling data for reciprocating saw cutting and removal of bolts with a powered socket tool. These results have general value in that they are relevant to extensions of this work and in the pursuit of other tool control strategies. In particular, the force profile generated for pipe cutting produces a well-defined characteristic signature that should be broadly useful even outside of the telerobotics community. Progressive variation in the tool signature profiles over repeated test instances indicate that tool wear, maintenance prediction, and fault detection can probably be deduced from further study of the process signature.

8.3 Future Work

There are several possibilities to consider for future work building on the research presented in this dissertation.

One topic of particular interest is to investigate how these techniques can be used to track and compensate for tool wear, to indicate component end of life, and to identify operational faults. Tool signatures were found to vary according to wear in the primary contact medium executing the tool task (such as a saw blade in a pipe). Higher and more rounded force levels in the saw process signature indicate a worn blade with dull or broken teeth. This should make it possible to determine at what point a tool piece should be changed out facilitating maintenance scheduling.

The basic framework is now in place to pursue dynamic motion of the manipulator base or the task object during task execution. This will require the development of new position sensing capabilities that can tolerate the vibrations, forces, and moments imposed by tooling operations. However this would afford the possibility of cutting operations even when the manipulator and task object are shaking and

vibrating in response to the cutting operation. This is a common task problem in D&D activities.

Success was shown to be possible using sensor data collected from sensors mounted in a common tool fixturing point. This indicates that it should be possible to move to a multi-fingered end-effector with a wrist mounted force-torque sensor and achieve similar success by also addressing tool position and orientation when grasping the tool. This is important because common sensing could be provided without the cost of bolting tools into a smart tool fixture.

Another area worthy of further investigation would be to consider how to apply these techniques to tool process that are essentially impact-based such as the jack hammer, air chisel, and sheet metal nibbler. Rapid motion of the tool coupled with a wide range of ways that the target object may react to the tool impact will make this a difficult study probably requiring extensive analytical and experimental development. Due to the rate of impacts and the forces encountered in the process, data acquisition and process control sample rates would have to be far higher than is typically used in manipulator control.

As with other early implementations using behavior-based concepts, the implementation process tends to be tedious, incremental, and leans heavily on experimental development. While this was intentional for this work in order to start from first principles, various learning techniques under development in the behavior-based community should be considered to provide automated assistance in the reactive function development process.

Chapter 9

Conclusions

This dissertation has described a methodology for combining concepts from behavior-based systems with telerobotic tool control in a way that is compatible with existing manipulator architectures used by remote systems typical to the D&D and remote operations environments. The concept was implemented and demonstrated for two tools useful to D&D type operations—a reciprocating saw and a powered socket tool. The experimental results demonstrated that the approach works to facilitate traded control telerobotic tooling execution by enabling difficult tasks and by limiting tool damage.

The original concept was intended as a means of adding telerobotic assists for human operators (1) to permit task tooling operation where it is currently difficult or impossible or (2) to relieve fatigue where the tool operation is tedious. For this purpose it appears to work either exceptionally well (reciprocating saw) or adequately (socket tool). The reciprocating saw task was impossible with freehand teleoperation on the test bed but readily achievable via the reactive function assists. The socket tool concept works well enough to use in conjunction with teleoperation since the operator has the capability to retarget and retry if the initial targeting fails.

The reactive functions formed a set of simple move primitives that can be readily assembled into new tooling tasks with relatively little difficulty. Knowledge of the task, task execution sequence, and tool characteristics are needed. A majority of the functions needed for the socket tool were directly derived from the saw tool. Having done the reciprocating saw and socket tool, a band saw, circular saw, and drill would be relatively easy to complete. The approach should expand readily to other D&D type tools as well as tools in other task sets such as small scale medical tooling for minimally invasive surgery.

There is no a priori modeling of a specific task instance. A task type model is embedded into the sequence of reactive functions and the actions of the functions themselves. Contact sensing is used to establish the location of the task object on which the tooling task executes. The point of interest for the task is established by the operator or a higher level robotic program. There is no abstract representation of the specific task instance stored anywhere in the system.

Initial investigation showed that telerobotic use of power tooling did not completely conform to the tenets of the behavior-based approach. Tooling tasks are almost entirely sequential and deterministic or can be made that way with minimal planning. This decreases the behaviorism content of the concept since behavior arbitration essentially goes away in favor of the sequential execution of reactive behaviors. It may be best to consider this as a BBR-inspired or derived technique rather than a pure behavior-based robotics technique.

The most advantageous component to this work that would facilitate complete robotic task execution is what has been learned about task decomposition to make what appears to be an exceedingly difficult task relatively easy by breaking it down into a set of simple moves and looking for target object contact and tool task signatures. The discovery of the tool process signatures and how they may be used to manage the tool process was an unexpected benefit of this work. It opens the door to the difficult to address needs of tool fault identification and recovery, predictive tool maintenance, and more extensive dynamics-based control techniques.

In summary the purpose of this work was to explore the use of behavior-based robotics concepts to determine techniques relevant to the use of telerobotic assists in D&D type tool tasks with a purpose of minimizing the task instance modeling in favor of a priori task type models while using sensor information to register the task type model to the task instance. An approach was implemented and tested for two tools with variation to the usual behavior selection process by using fixed sequencing of the reactive functions.

The task type model was embedded into the sequencing of the functions and the functions themselves. There is no abstract representation used to build a specific task instance. Both tool implementations worked well. In the case of the reciprocating saw, the implementation was an enabling technology. The role of the tools and tasks as drivers to the telerobotic implementation was better understood in the need for thorough task decomposition. This work has been successful enough that it can be implemented and used near term on real world systems.

List of References

- [1] M. W. Noakes, "CP5 Reactor Remote Dismantlement Activities: Lessons Learned in the Integration of New Technology in an Operations Environment," presented at the Advanced Robotics Beyond 2000, 29th International Symposium on Robotics, Birmingham, UK, 1998.
- [2] W. R. Hamel, "Sensor-Based Planning and Control in Telerobotics (Chapter 10)," in *Control in Robotics and Automation: Sensor-based Integration*, B. K. Ghosh, *et al.*, Eds., ed: Academic Press, 1999, pp. 285 - 309.
- [3] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*: The MIT Press, 1992.
- [4] T. W. Burgess, *et al.*, "Design Guidelines for Remotely Maintained Equipment," Oak Ridge National Laboratory ORNL/TM-10864, November 1988.
- [5] R. C. Goertz. (November 1952, November) Fundamentals of General-Purpose Remote Manipulators. *Nucleonics*. 36 - 42.
- [6] R. C. Goertz, "Manipulator Systems Development at ANL," in *12th Conference on Remote Systems Technology*, 1964, pp. 117 - 136.
- [7] R. C. Goertz, "Some Work on Manipulator Systems at ANL Past, Present, and a Look at the Future," in *1964 Seminars on Remotely Operated Special Equipment*, Germantown, Maryland, 1964, pp. 27 - 69.
- [8] R. C. Goertz and F. Bevilacqua. (November 1952) A Force-reflecting Positional Servomechanism. *Nucleonics*. 43 - 45.
- [9] C. R. Flateau, "Development of Servo Manipulators for High Energy Accelerator Requirements," in *13th Conference on Remote Systems Technology*, 1965, pp. 29 - 35.
- [10] C. R. Flateau, "Compact Servo Master-slave Manipulator with Optimized Communication Links," in *17th Conference on Remote Systems Technology*, 1969, pp. 154 - 164.
- [11] J. W. Clark, "Mobotry: The New Art of Remote Handling," *IRE Transactions on Vehicular Communications*, vol. 10, pp. 12 - 24, 1961.
- [12] M. W. Noakes, *et al.*, "Telerobotic Planning and Control for DOE D&D Operations," in *2002 IEEE International Conference on Robotics and Automation*, 2002, pp. 3485 - 3492.
- [13] H. C. Humphreys, *et al.*, "Large Scale Multi-fingered End-effector Manipulation," presented at the 2nd International Joint Topical Meeting on Emergency Preparedness & Response and Robotics & Remote Systems, Albuquerque NM, 2008.
- [14] T. W. Burgess and M. W. Noakes, "US Robotics and Remote Systems in the Nuclear Field," presented at the 1st PREFIT (Preparing Remote Handling Engineers for ITER) Workshop, Culham Science Center, Culham Laboratory, Abingdon UK, 2007.
- [15] S. Hyati, *et al.*, "A Testbed for a Unified Teleoperated-autonomous Dual-arm Robotic System," in *1990 IEEE International Conference on Robotics and Automation*, 1990, pp. 1090 - 1095.

- [16] S. E. Everett and R. V. Dubey, "Human-machine Cooperative Telerobotics Using Uncertain Sensor or Model Data," in *1998 IEEE International Conference on Robotics & Automation*, 1998, pp. 1615 - 1622.
- [17] R. C. Arkin and L. E. Parker, "personal communication on a concise definition of behavior-based systems relayed via Lynne E. Parker," M. W. Noakes, Ed., ed, 2006.
- [18] J. H. Connell, "A Behavior-based Arm Controller," *IEEE Transactions on Robotics and Automation*, vol. 5, pp. 784 - 791, 1989.
- [19] E. C. Tolman, "Prediction of Vicarious Trial and Error by Means of the Schematic Sowbug," *Psychological Review*, vol. 46, pp. 318-336, 1939.
- [20] E. Yoichiro and R. C. Arkin, "Implementing Tolman's Schematic Sowbug: Behavior-based Robotics in the 1930's," in *2001 IEEE International Conference on Robotics and Automation*, Seoul Korea, 2001, pp. 477 - 484.
- [21] N. Wiener. (1948, November 1948) Cybernetics. *Scientific American*. 14 - 19.
- [22] W. G. Walter. (May 1950) An Imitation of Life. *Scientific American*. 42-45.
- [23] W. G. Walter. (August 1951) A Machine That Learns. *Scientific American*. 60-63.
- [24] W. G. Walter, *The Living Brain*. New York NY: Norton & Company, Inc, 1953.
- [25] R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, pp. 14 - 23, 1986.
- [26] V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*. Cambridge MA: The MIT Press, 1984.
- [27] R. A. Brooks, *Cambrian Intelligence: The Early History of the New AI*. The MIT Press, 1999.
- [28] R. C. Arkin, "Reactive Control as a Substrate for Telerobotic Systems," in *IEEE 1991 National Telesystems Conference*, Atlanta GA, 1991, pp. 309 - 314.
- [29] F. G. Pin, "A Fuzzy Behaviorist Approach to Sensor-based Reasoning and Robot Navigation," in *Control in Robotics and Automation: Sensor-based Integration*, B. K. Ghosh, *et al.*, Eds., ed: Academic Press, 1999.
- [30] B. D. Tribelhorn, Z., "Evaluating the Roomba: A Low-cost, Ubiquitous Platform for Robotics Research and Education," in *2007 IEEE International Conference on Robotics and Automation*, Roma, Italy, 2007, pp. 1393 - 1399.
- [31] R. C. Arkin and K. S. Ali, "Integration of Reactive and Telerobotic Control in Multi-agent Robotic Systems, From Animals to Animats," in *Third International Conference on the Simulation of Adaptive Behavior*, Brighton UK, 1994, pp. 473 - 478.
- [32] P. Garcia, *et al.*, "Trauma Pod: A Semi-automated Telerobotic Surgical System," *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 5, pp. 136 - 146, 16 February 2009.
- [33] M. W. Noakes, *et al.*, "Development of a Remote Trauma Care Assist Robot," in *IROS 2009: The 2009 IEEE/RSJ International Conference on intelligent Robots and Systems*, St. Louis, MO, 2009, pp. 2580 - 2585.
- [34] R. C. Arzbaeher, "Servomechanisms With Force Feedback," PhD, University of Illinois, 1960.

- [35] J. Vertut, "Advances in Remote Manipulation," in *International Conference on World Nuclear Energy - A Status Report*, Washington, D. C., 1976, pp. 503-505.
- [36] M. W. Noakes, "Experiments in Teach-Playback Robotics with the Advanced Servomanipulator," Oak Ridge National Laboratory 1987.
- [37] J. Vertut and P. Coiffet, *Teleoperators and Robotics, Evolution and Development, Robot Technology* vol. 3A: Hermes Publishing, 1985.
- [38] G. K. Corbett, *et al.*, "Modifying A Telerobotic System to Include Robotic Operation by Means of Dynamic Modeling," presented at the IEEE International Conference On Robotics and Automation, Scottsdale, Arizona, 1989.
- [39] T. W. Burgess, "Discussion on Limitations of Teleoperation," ed, 2009.
- [40] O. Khatib, "Real-time Control of Manipulators in Operational Space," presented at the 1984 ASQC 28th Annual Stanford Conference, Stanford, CA, 1984.
- [41] P. Garrac, "Discussions on Dissimilar Kinematic Cartesian Controls," M. W. Noakes, Ed., ed. Oak Ridge, Tennessee, 2010.
- [42] J. V. Draper and M. W. Noakes, "Fundamental Issues in the Application of Teleoperation, Robotics, and Supervisory Control," presented at the American Nuclear Society Sixth Topical Meeting on Robotics and Remote Systems, Monterey, California, 1995.
- [43] T. W. Burgess, "Cost of Commercial Servomanipulator Systems," M. W. Noakes, Ed., ed. Oak Ridge TN, 2010.
- [44] P. Garrec, "Discussions on Dissimilar Kinematic Cartesian Controls," M. W. Noakes, Ed., ed. Oak Ridge, Tennessee, 2010.
- [45] P. Garrec, *et al.*, "Evaluation Tests of the Telerobotic System MT200-TAO in AREVA-NC/LA Hague Hot Cells," in *European Nuclear Conference (ENC 2007)*, Brussels, Belgium, 2007, pp. 1-7.
- [46] F. Geffard, *et al.*, "On the Use of a Base Force/Torque Sensor in Teleoperation," in *2000 IEEE International Conference on Robotics and Automation*, San Francisco, California, 2000, pp. 2677-2683.
- [47] T. F. Chan, *et al.*, "Generalized Bilateral Control for Dissimilar Kinematic Teleoperators and Application to D&D Type Tasks," presented at the American Nuclear Society Sixth Topical Meeting on Robotics and Remote Systems, Monterey, CA, 1995.
- [48] S. E. Everett, "Human-machine Cooperative Telerobotics Using Uncertain Sensor and Model Data," PhD, Mechanical Engineering, University of Tennessee at Knoxville, 1998.
- [49] T. B. Sheridan, "Space Teleoperation Through Time Delay: Review and Prognosis," *IEEE Transactions on Robotics and Automation*, vol. 9, pp. 592 - 606, October 1993 1993.
- [50] M. Montemerlo, "NASA's Automation and Robotics Technology Development Program," in *1986 IEEE International Conference on Robotics and Automation*, 1986, pp. 977 - 986.
- [51] C. R. Weisbin and M. Montemerlo, "NASA's Telerobotics Research Program," in *1992 IEEE International Conference on Robotics and Automation*, Nice FR, 1992, pp. 2653 - 2666.

- [52] G. Hirzinger, *et al.*, "Sensor-based Space Robotics_ROTEx and Its Telerobotic Features," *IEEE Transactions on Robotics and Automation*, vol. 9, pp. 649 - 663, October 1993.
- [53] G. Hirzinger, *et al.*, "Teleoperating Space Robots. Impact for the Design of Industrial Robots," in *ISIE '97: IEEE International Symposium on Industrial Electronics*, 1997, pp. SS250 - SS256.
- [54] P. Backes, *et al.*, "A Local-remote Telerobot System for Time-delayed Traded and Shared Control," presented at the Fifth International Conference on Advanced Robotics, 1991, Pisa, Italy, 1991.
- [55] S. Lee. (1993, June 1993) Intelligent Sensing and Control for Advanced Teleoperation. *Control Systems Magazine, IEEE*. 19-28.
- [56] R. O. Ambrose, *et al.* (2000, July - August) Robonaut: NASA's Space Humanoid. *IEEE Intelligent Systems and Their Applications*. 57 - 63.
- [57] D. E. Whitney, "Development and Control of an Automated Robotic Weld Bead Grinding System," *Journal of Dynamical Systems, Measurement, and Control*, vol. 112, pp. 166 - 176, June 1990.
- [58] (2003, October 18). *DOE Robotics Crosscutting Program*.
- [59] N. Xi and T. J. Tarn, "Sensor-Based Planning and Control for Robotic Systems: An Event-based Approach," in *Control in Robotics and Automation: Sensor-based Integration*, B. K. Ghosh, *et al.*, Eds., ed: Academic Press, 1999, pp. 3 - 55.
- [60] N. Xi, "Event-based Motion Planning and Control for Robotic Systems," PhD PhD, Washington University, 1993.
- [61] M. W. Noakes and W. E. Dixon, "Application of a Selective Equipment Removal System to D&D Tasks," presented at the American Nuclear Society Sixth Topical Meeting on Robotics and Remote Systems, Monterey, California, 1995.
- [62] W. R. Hamel and R. L. Kress, "Elements of Telerobotics Necessary for Waste Cleanup Automation," in *2001 IEEE International Conference on Robotics & Automation*, 2001, pp. 393 - 400.
- [63] W. R. Hamel, *et al.*, "A Real-time Controller for Human-machine Cooperative Telerobotics," in *2002 IEEE International Conference on Robotics and Automation*, 2002, pp. 2880 - 2885.
- [64] S. Kim and W. R. Hamel, "Design of Supervisory Control Scheme for Fault Tolerant Control of Telerobotics System in Operational Space," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, pp. 2803 - 2807.
- [65] S. Kim and W. R. Hamel, "Force Assistance Function for Human Machine Cooperative Telerobotics Using Fuzzy Logic," in *2002 IEEE International Conference on Robotics and Automation*, 2002, pp. 2165 - 2170.
- [66] S. Kim and W. R. Hamel, "Operational Fault Detection for Telerobotic Systems Using Discrete Wavelet Transform (DWT) and AC Analysis of Position Data," in *2004 IEEE International Conference on Robotics and Automation*, 2004, pp. 5001 - 5006.

- [67] A. Nycz and W. R. Hamel, "Robot Task Space Analyzer System Calibration," in *1st Joint Emergency Preparedness and Response and Robotic and Remote Systems*, Salt Lake City, Utah, 2006.
- [68] G. Zhang, "An Adaptive Tool-based Telerobotic Control System," PhD, Mechanical Engineering, University of Tennessee at Knoxville, December 2004.
- [69] W. R. Hamel, *et al.*, "Large Scale Multi-fingered End-effector Teleoperation," in *IEEE/RSJ 2009 International Conference on Intelligent Robots and Systems*, 2009, pp. 3304 - 3310.
- [70] T. Kesavadas and D. J. Cannon, "Virtual Tools With Attributes for Robotic Based Intermediate Manufacturing Processes," in *1996 IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, 1996, pp. 1845 - 1850.
- [71] M. Hwan Yun, *et al.*, "An Instrumented Glove for Grasp Specification in Virtual-Reality-Based Point-and-Direct Telerobotics," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 27, pp. 835-846, 1997.
- [72] C. Wang, *et al.*, "Human-Machine Collaboration in Robotics: Integrating Virtual Tools with a Collision Avoidance Concept using Conglomerates of Spheres," *Journal of Intelligent and Robotic Systems*, pp. 1-31, 1997.
- [73] D. J. Cannon and G. Thomas, "Virtual Tools for Supervisory and Collaborative Control of Robots," *Presence*, vol. 6, pp. 1-28, 1997.
- [74] D. Aarno, *et al.*, "Adaptive Virtual Fixtures for Machine-Assisted Teleoperation Tasks," in *2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005, pp. 1151 - 1156.
- [75] W. Yu, *et al.*, "Telemanipulation Assistance Based on Motion Intention Recognition," in *2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005, pp. 1133 - 1138.
- [76] O. Bebek and M. C. Cavusoglu, "Whisker Sensor Design for Three Dimensional Position Measurement in Robotic Assisted Beating Heart Surgery," in *2007 IEEE International Conference on Robotics and Automation*, Roma Italy, 2007, pp. 225 - 231.
- [77] S. Saha, "Appropriate Degrees of Freedom of Force Sensing in Robot-assisted Minimally Invasive Surgery," Master of Science MS, Johns Hopkins University, Baltimore, MD, 2005.
- [78] P. Dario, *et al.*, "Smart Surgical Tools and Augmenting Devices," *IEEE Transactions on Robotics and Automation*, vol. 19, pp. 782-792, October 2003.
- [79] J. S. Albus, "A Theory of Intelligent Systems," in *5th IEEE International Symposium on Intelligent Control*, 1990, pp. 866 - 875.
- [80] E. Angelopoulou, *et al.*, "World Model Representation for Mobile Robots," in *1992 Symposium on Intelligent Vehicles*, 1992, pp. 293 - 297.
- [81] R. E. Barry, *et al.*, "Rapid World Modeling From a Mobile Platform," in *1997 IEEE International Conference on Robotics and Automation*, 1997, pp. 72 - 77.
- [82] InnovMetric. (2010, October 18). *PolyWorks V11, The Universal 3D Metrology Software Platform*.

- [83] R. C. Arkin, "Motor Schema-based Mobile Robot Navigation," *International Journal of Robotics Research*, vol. 8, pp. 92 - 112, August 1989.
- [84] A. Stoytchev and R. C. Arkin, "Combining Deliberation, Reactivity, and Motivation in the Context of a Behavior-based Robot Architecture," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 2001, pp. 290 - 295.
- [85] R. C. Arkin and D. C. Mackenzie, "Planning to Behave: A Hybrid Deliberative/Reactive Control Architecture for Mobile Manipulation," in *1994 International Symposium on Robotics and Manufacturing*, Maui HI, 1994, pp. 5 - 12.
- [86] R. C. Arkin, "Towards the Unification of Navigational Planning and Reactive Control," in *working notes of the AAI Spring Symposium on Robot Navigation*, Stanford University, 1989.
- [87] D. C. Bentivegna, *et al.*, "Design and Implementation of a Teleautonomous Hummer," presented at the Mobile Robots XII, Pittsburgh PA, 1997.
- [88] K. S. Ali and R. C. Arkin, "Multiagent Teleautonomous Behavioral Control," *Machine Intelligence and Robotic Control*, vol. 1, pp. 3 - 10, 2000.
- [89] R. Arkin, "Reactive Robotic Systems," College of Computing, Georgia Institute of Technology 1995.
- [90] J. M. Cameron, *et al.*, "Reactive Control for Mobile Manipulation," in *1993 IEEE International Conference on Robotics and Automation*, 1993, pp. 228 - 235.
- [91] D. C. MacKenzie and R. C. Arkin, "Behavior-based Mobile Manipulation for Drum Sampling," in *1996 IEEE International Conference on Robotics and Automation*, Minneapolis, MN, 1996, pp. 2389 - 2395.
- [92] M. R. Stein and R. P. Paul, "Operator Interaction, for Time-delayed Teleoperation, With a Behavior-based Controller," in *1994 IEEE International Conference on Robotics and Automation*, 1994, pp. 231 - 236.
- [93] Y. S. Park, "Structured Beam Projection for Semi-automatic Teleoperation," in *SPIE Optomechatronic Systems, Photonics East 2000: Intelligent Systems and Advanced Manufacturing*, Boston MA, 2000, pp. 192 - 201.
- [94] Y. S. Park and *e. al.*, "Perceptual Basis for Reactive Teleoperation," in *SPIE Optomechatronic Systems II, Photonics Boston: Intelligent Systems and Advanced Manufacturing 2001*, Newton MA, 2001, pp. 115 - 122.
- [95] Y. S. Park, *et al.*, "Agent-based Dual Arm Motion Planning for Semi-automatic Teleoperation," presented at the ANS 9th International Topical Meeting on Robotics and Remote Systems, Seattle WA, 2001.
- [96] Y. S. Park, *et al.*, "Enhanced Teleoperation for D&D," in *2004 IEEE International Conference on Robotics and Automation*, New Orleans LA, 2004, pp. 3702 - 3707.
- [97] H. Kang, *et al.*, "Visually and Haptically Augmented Teleoperation in D&D Tasks using Virtual Fixtures," in *10th Robotics & Remote Systems Meeting*, Gainesville FL, 2004, pp. 466 - 471.
- [98] G. C. Pettinaro, "Behavior-based Peg in Hole," *Robotica*, vol. 17, pp. 189 - 201, March - April 1999.

- [99] Z. Wasik and A. Saffiotti, "A Fuzzy Behavior-based Control System for Manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002, pp. 1596 - 1601.
- [100] Z. Wasik and A. Saffiotti, "A Hierarchical Behavior-Based Approach to Manipulation Tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, Taipei, TW, 2003, pp. 2780 - 2785.
- [101] A. Stoytchev, "Behavior-grounded Representation of Tool Affordances," in *2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 3060 - 3065.
- [102] F. G. Pin and Y. Watanabe, "Navigation of Mobile Robots Using a Fuzzy Behaviorist Approach and Custom-designed Fuzzy Inference Boards," *Robotica*, vol. 12, pp. 491 - 503, 1994.
- [103] F. G. Pin and Y. Watanabe, "Automatic Generation of Fuzzy Rules Using the Fuzzy Behaviorist Approach: the Case of Sensor-based Robot Navigation," *Intelligent Automation and Soft Computing*, vol. 1, pp. 161 - 178, 1995.
- [104] M. W. Noakes, "Solution to Remote Cutting "Tool Chatter" Via Tool-oriented, Sensor-based Control," Oak Ridge National Laboratory/DOE EM50 September 2003.
- [105] D. C. Haley and M. W. Noakes, "Solution to Remote Cutting "Tool Chatter" via Tool-oriented Sensor-based Control," ed, 2003.
- [106] S. A. Tobias, *Machine-tool Vibration*. New York NY: Blackie & Son Ltd., 1965.
- [107] R. N. Arnold, "Mechanism of Tool Vibration in Cutting of Steel," *Proceedings of Institution of Mechanical Engineers*, vol. 154, p. 261, 1946.
- [108] R. S. Hahn, "Metal Cutting Chatter and Its Elimination," *Transactions of the ASME*, vol. 35, p. 1073, 1953.
- [109] L. Le-Ngoc and H. McCallion, "Self-induced Vibration of Bandsaw Blades During Cutting," *Proceedings of Institution of Mechanical Engineers*, vol. 213, pp. 371 - 380, 1999.
- [110] C. Andersson, *et al.*, "Bandsawing. Part I: Cutting Force Model Including Effects of Position Errors, Tool Dynamics, and Wear," *International Journal of Machine Tools & Manufacture*, vol. 41, pp. 227 - 236, 2001.
- [111] C. Andersson, *et al.*, "Bandsawing. Part II: Detecting Positional Errors, Tool Dynamics and Wear by Cutting Force Measurement," *International Journal of Machine Tools & Manufacture*, vol. 41, pp. 237 - 253, 2001.
- [112] S. Y. Liang, *et al.*, "Machining Process Monitoring and Control: The State-of-the-Art," presented at the Proceedings of IMECE2002, ASME International Mechanical Engineering Congress & Exposition, New Orleans LA, 2002.
- [113] M. W. Noakes, "Saw Blade Chatter Detection and Correction--Developing the Smart Tooling Concept for Remote Manipulation," presented at the American Nuclear Society Tenth International Topical Meeting on Robotics and Remote Systems for Hazardous Environments, Gainesville, Florida, 2004.
- [114] M. Smith, "RealTime Performance of the Linux 2.2 Kernel," University of Tennessee, Knoxville, Tennessee, 1999.

- [115] R. Zhou, *et al.*, "Using the WAM as a Master Controller," presented at the 1st Joint Emergency Preparedness and Response/Robotic and Remote Systems Topical Meeting, Salt Lake City UT, 2006.
- [116] A. Nycz and W. R. Hamel, "Whole Arm Manipulator as a Universal Master Controller," in *2nd International Joint Topical Meeting on Emergency Preparedness & Response and Robotics & Remote Systems*, Albuquerque, New Mexico, 2008.
- [117] R. Arkin, *Behavior-based Robotics*. Cambridge, Massachusetts: MIT Press, 1998.

Appendices

Appendix A

Software

The software for this dissertation was written using GNU open source tools for the Linux environment. Except for some system level interactions that use C++, all files are written in C without the use of object-oriented techniques. Functions are meant to be stand-alone as much as possible to facilitate individual testing and regrouping for other tools with minimal function modification; however there are groups of functions that are quite similar as outlined below. Vestigial code for the interface of analog sensors not used is left intact to facilitate future expansion and as documentation to those conducting follow-on work in our lab.

bApproachB, bApproachH, and bApproachV, though using slightly different trajectory generation, are similar. bBackH and bBackV are likewise similar to each other and derived from the Approach functions. moveHome is included as representative of a family of robotic moves used in this work to go to preprogrammed targets. bWristR is a similar robotic move function to moveHome and its category of motions. bCut128S and bUnboltB are each unique to their tooling operation. The included files are listed below in order of presentation in this appendix. No attempt has been made to include all of the many MATLAB files used for analysis in this work, but all of those techniques are straightforward engineering exercises. References to comediFT.h refer to a support file available from ATI. References to newChild.h and child2() indicate software borrowed from Andrzej Nycz's UTK Robotics Laboratory system software and are also therefore not included here.

robot.h.

read_writeIO.h
read_writeIO2.c

runbSaw.c
runbSocketS1.c

bApproachB.c
bApproachH.c
bApproachV.c
bBackH.c
bBackV.c
bRetractB.c
bRetractS.c

bWristR.c
bCut128S.c
bUnboltB.c

functGoIdle.c
functMoveHome.c


```

/*****
*
*   robot.h
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*****/

#include <time.h>
#include <math.h>
#include <stdio.h>
#include <unistd.h>

// Calculate for each DOF; numbers in inches, used in trajectory calcs.

double qZero[6];    // Start point of robotic move
double qFinal[6];   // Finish point of robotic move
double qNow[6];     // Current calculated point in robotic trajectory

float    FT[6];     // Force torque sensor values

// Stored Cartesian position, from Approach --> Retract

double cStored[6] = {0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000};

// Stored instantaneous joint positions

double qJoints[7] = {0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000};

// Following joint positions are ordered as follows:
// shoulder azimuth, shoulder pitch, elbow, wrist pitch, wrist yaw, wrist roll

// Right T2 home joint positions.

double qHome[7] = {0.0000, 0.8800, -2.5436, 0.3889, 0.0000, -1.4653, 0.0000};

// Socket robotic move target T2 joint positions.

double qSocket[7] = {-0.0859, 0.8183, -2.1817, 1.0144, -1.3175, -1.3866,
0.0000};

// Saw robotic move target T2 joint positions.

double qSaw1[7] = {-0.705671, 0.484300, -2.542479, 1.476005, 0.381025,
-1.802548, 0.0000};
double qSaw2[7] = {-0.675950, 0.994828, -2.060330, 0.996445, 0.547078,
-1.602363, 0.0000};
double qSaw3[7] = {-0.539330, 0.989076, -2.371632, 1.358752, 0.513427,
-1.587311, 0.0000};
double qPipe[7] = {-0.286702, 0.453717, -2.124469, 1.469198, 0.235584,
-1.650875, 0.0000};

```

```

// Position increment instead of time but run at sample time.

int posInc;

// Function prototypes

int read_writeIO(void);

int functMoveHome(void);
int functMoveSocket(void);
int functMoveSocket1(void);

int storeCartesian(void);
int functApproach(void);
int functApproachD(void);
int functRetract(void);
int functRetractD(void);
int functGoIdle(void);

int functMoveSaw1(void);
int functMoveSaw2(void);
int functMoveSaw3(void);
int functMovePipe(void);

int bWristR(void);
int bApproachB(void);
int bApproachH(void);
int bApproachV(void);
int bBack(void);
int bBackH(void);
int bBackV(void);
int bCut128S(void);
int bRetractB(void);
int bRetractS(void);
int bUnboltB(void);

```

```

/*****
*
* Filename = read_writeI02.h
* Support for digital and analog I/O
*
* Obligatory GNU Comedi acknowledgment
*
* Derived from Comedilib, tut1.c
* Copyright (c) 1999,2000 David A. Schleeef <ds@schleeef.org>
*
* This file may be freely modified, distributed, and combined with
* other software, as long as proper attribution is given in the
* source code.
*
* NOTES
*
* subdev 0 = analog input port
* subdev 2 = digital I/O port, note that there are many ports including
* several digital ports and it's easy to get confused as to what does what.
*
* Much of this is now vestigial code but required to read I/O anyway.
*
*****/

#include <stdio.h>
#include <comedi.h>
#include <comedilib.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <getopt.h>
#include <ctype.h>
#include <signal.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include </usr/src/linux-2.6.23/include/linux/rtc.h>

#include "examples.h"

// Reciprocating saw sensor inputs = left and right, horizontal and
// vertical, slide and touch inputs.

double sawRHslidePos;
double sawRHtouchPos;
double sawRVslidePos;
double sawRVtouchPos;
double sawLHslidePos;
double sawLHtouchPos;
double sawLVslidePos;
double sawLVtouchPos;

// Power supply monitoring

double checkPlusTen;
double checkMinusTen;
double checkFive;

int bits[8];

```

```
// Digital outputs for smart tool on comedi0

int toolOn;          // toolOn = 1 is on; use as either on/off or PWM.
int toolDir;  // toolDir = 0 is forward as default; reverse is 1.

// Digital inputs for smart tool on comedi1

int toolOnIN;        // toolOn input
int toolDirIN;       // toolDir input
```

```

/*****
*
* Filename = read_writeI02.c
*
* This file is the function to read and write analog and digital I0 from
* the National Instruments 6034E for the HLC. It does not do the
* force/torque sensor.
*
* Obligatory GNU comedi acknowledgment
*
* Derived from Comedilib, tut1.c
* Copyright (c) 1999,2000 David A. Schleeef <ds@schleeef.org>
*
* This file may be freely modified, distributed, and combined with
* other software, as long as proper attribution is given in the
* source code.
*
* NOTES
*
* subdev 0 = analog input port
* subdev 2 = digital I/O port, note that there are many ports including
* several digital ports and it's easy to get confused as to what does what.
*
*****/

#include "read_writeI02.h"

int read_writeI0(void)
{
    int subdev      = 0;    // varies depending on analog/digital port
    int chan        = 0;    // varies under this application
    int range       = 0;    // 0 = +/-10, still have to use for digital
    int aref = AREF_GROUND; // AREF_GROUND for SE; AREF_DIFF for DE

    int n_chans0;
    int maxdata0;

    double voltage[16];

    comedi_t *device0;
    comedi_t *device1;

    lsampl_t data0;
    lsampl_t bits0 = 0;

    int ret;

    lsampl_t data1;

    // comedi0 smart tooling I/O

    device0 = comedi_open("/dev/comedi0");

    n_chans0 = comedi_get_n_channels(device0, subdev);

    for(chan = 0; chan < n_chans0; ++chan){
        maxdata0 = comedi_get_maxdata(device0, subdev, chan);
        comedi_data_read(device0, subdev, chan, range, aref, &data0);
    }
}

```

```

        voltage[chan] = comedi_to_phys(data0, comedi_get_range(device0, subdev,
            chan, range), maxdata0);
    }

// Smart tool position sensors, +/- 10VDC to mm,
// NOTE: Calibration on voltage only

    sawRHslidePos = 1.016 * voltage[0] - .119;
    sawRHtouchPos = 1.016 * voltage[1] - .119;
    sawRVslidePos = 1.016 * voltage[2] - .119;
    sawRVtouchPos = 1.016 * voltage[3] - .119;
    sawLHslidePos = 1.016 * voltage[4] - .119;
    sawLHtouchPos = 1.016 * voltage[5] - .119;
    sawLVslidePos = 1.016 * voltage[6] - .119;
    sawLVtouchPos = 1.016 * voltage[7] - .119;

// Power supply checks for diagnostics and scaling
// --> Calibrated DC voltages

    checkPlusTen = 1.016 * voltage[13] - .119;
    checkMinusTen = 1.016 * voltage[14] - .119;
    checkFive = 1.016 * voltage[15] - .119;

// Digital input

    for(chan = 4; chan < 8; ++chan){
        comedi_data_read(device0, 2, chan, range, aref, &bits0);
        bits[chan] = bits0;
    }

// Reads inputs and assign to outputs.

// toolOnIN = bits[4]; // change for manual vs. auto input
// toolDirIN = bits[5]; // change for manual vs. auto input

    bits[0] = ! toolOnIN; // ! fixes inverted logic.
    bits[1] = ! toolDirIN; // ! fixes inverted logic.

// Digital output

    for(chan = 0; chan < 4; ++chan)
    {
        comedi_data_write(device0, 2, chan, range, aref, bits[chan]);
    }

    comedi_close(device0);

    return 0;
}

```

```

/*****
*
*   runbSaw.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*****/

#include "robot.h"

int main(void)
{
    functMoveHome(); // Joint motion move to home for consistent starting position.
    printf("move home ok\n\n");
    functMoveSaw1(); // Joint level move to Saw way point.
    printf("move saw 1 ok\n\n");
    functMoveSaw2(); // Joint level move to Saw way point.
    printf("move saw 2 ok\n\n");
    functMoveSaw3(); // Joint level move to Saw way point.
    printf("move saw 3 ok\n\n");

    // BBR Start //////////////////////////////////////
    bWristR(); // Level wrist roll to horizontal before cutting.
    printf("wrist roll ok\n\n");
    bApproachH(); // Cartesian approach to target along EE to contact
    printf("approachH ok\n\n");
    bBackH(); // Cartesian motion along the EE vector to stand off
    printf("backH ok\n\n");
    bApproachV(); // Cartesian approach to target along EE to contact
    printf("approachV ok\n\n");
    bBackV(); // Cartesian motion along the EE vector to stand off
    printf("backV ok\n\n");
    bCut128S(); // Cartesian -Z for time/distance
    printf("cut ok\n\n");
    bRetractS(); // Retract along line to clear area.

```

```
    printf("retract ok\n\n");  
    // BBR conclude ///////////////////////////////////////  
    functMoveSaw2(); // Joint level move to Saw way point.  
    printf("move saw 2 ok\n\n");  
    functMoveSaw1(); // Joint level move to Saw way point.  
    printf("move saw 1 ok\n\n");  
    functMoveHome(); // Joint motion move to home.  
    printf("move home ok\n\n");  
    functGoIdle(); // Set control state via shared memory to Idle.  
    printf("idle ok\n\n");  
    return 0;  
}
```



```

/*****
*
*   runbSocketS1.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*****/

#include "robot.h"

int main(void)
{
    functMoveHome(); // Joint motion move to home for consistent starting position.
    printf("move home ok\n\n");
    sleep(1);
    functMoveSocket(); // Joint level move to Socket task start point.
    printf("move socket ok\n\n");
    sleep(1);
    functMoveSocket1(); // Joint level move to conehead socket task start point.
    printf("move cone1 ok\n\n");
    sleep(1);

    // BBR Start //////////////////////////////////////
    bApproachB(); // Cartesian approach to target along EE to contact
    printf("approach ok\n\n");
    sleep(1);
    bUnboltB();
    printf("unbolt ok\n\n");
    bRetractB(); // Retract along line to clear area.
    printf("retract ok\n\n");
    sleep(1);

    // BBR End //////////////////////////////////////
    functMoveHome(); // Joint motion move to home for consistent starting position.
    printf("move home ok\n\n");
    sleep(1);

```

```
    functGoIdle(); // Set control state via shared memory to Idle.  
    printf("idle ok\n\n");  
    sleep(1);  
    return 0;  
}
```

```

/*****
*
*   bApproachB.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*
*
*****/

/*****
*
*   Obligatory Acknowledgements for libraries used in this file.
*
*   ATIDAQ F/T C Library
*   v1.0.1
*   Copyright (c) 2001 ATI Industrial Automation
*
*   The MIT License
*
*   Permission is hereby granted, free of charge, to any person obtaining a
*   copy of this software and associated documentation files (the "Software")
*   to deal in the Software without restriction, including without limitation
*   the rights to use, copy, modify, merge, publish, distribute, sublicense,
*   and/or sell copies of the Software, and to permit persons to whom the
*   Software is furnished to do so, subject to the following conditions:
*
*   The above copyright notice and this permission notice shall be included
*   in all copies or substantial portions of the Software.
*
*   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
*   OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
*   MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
*   IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
*   CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
*   TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
*   SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*
*****/

/*****
*
*   Comedilib
*   Copyright (c) 1999,2000 David A. Schleef <ds@schleef.org>
*
*   This file may be freely modified, distributed, and combined with
*   other software, as long as proper attribution is given in the
*   source code.
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

```

```

#include "comediFT.h"

static int biasFlag = 1;    // for sampleBias switching to initialize F/T
int read_writeIO(void);    // reads comedi0 analog/digital IO

int bApproachB(void)
{
    // System level communications

    int QUIT = 0;

    int shmidR,shmidRW, semid; // IPC identifiers
    key_t key_memRW,key_memR, key_sem; // keys for shared mem and semaphores.
    struct sembuf sb; // semaphore control structure

    /*******

    void safe_quit(void)
    {
        QUIT=1;
    }

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

    int grabSem(int semNum, struct sembuf *sb, int semid)
    //semNum should be zero for this program so far.
    {
        sb->sem_op=-1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)
            // make sure you're using the semaphore when it is necessary.
            {
                perror("semaphore access problem");
                QUIT=1;
            }
        return 1;
    }

    /*******

    int retSem(int semNum, struct sembuf *sb, int semid)
    //semNum should be zero for this program so far.
    {
        sb->sem_op=1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)
            {
                perror("semaphore return problem ");
                QUIT=1;
            }
        return 1;
    }

    /*******

    // Calculate for each DOF; numbers in inches, used in trajectory calcs.

```

```

double qZero[6];    // Start point of robotic move
                    // (where you are now)

double qNow[6];     // Current calculated point in
                    // robotic trajectory

double qNowV[6];    // Incremental velocity for wrist
                    // orientations--warning not functional

double qNowOld[6];  // Used for incremental velocity calcs

// Stored instantaneous joint positions

double qJoints[7];

// Following joint positions are ordered as follows:
// shoulder azimuth, shoulder pitch, elbow, wrist pitch, wrist yaw,
// wrist roll

double Data[6];     // current manipulator position

int senseContact = 0;

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
//    ts.tv_nsec = 31250000; // 32 hz, not calibrated
    ts.tv_nsec = 24400000; // calibrated for actual runtime 32 hz

// time-stamping variables

    time_t time(time_t *tp);

    time_t now;

// file for data capture

    FILE *fp;

    if ((fp = fopen("approachB_data", "wb"))==NULL)
    {
        printf("Cannot open file.\n");
        exit(1);
    }

// Setup shared memory

    child2( );

// Loop Variables

    int i = 0;
    int j = 0;

    double inc = .015625; // .5 in/sec @ 32 hz
    float contactThreshold = -40.00;

    // Set constraints and scaling.

```

```

// Note that positions use 1; orientations use 0.

for (i=0;i<6;i++) //initialize memory
{
    parmRW->armCtrl.axesConstr[i]=1.0;
    parmRW->armCtrl.axesScal[i]=1.0;
    parmRW->armCtrl.armMode=IDLE;
    parmRW->armCtrl.cartesCtrl[i]=parmR->armRightCar[i];
    if(i>2)
    {
        parmRW->armCtrl.cartesCtrl[i]=0.0;
    }
}

// Read the starting Cartesian position (where you are now) from
// shared memory.

for (i = 0; i < 6; i++)
{
    qZero[i] = parmR->armRightCar[i];
}

// Read the starting joint angles (where you are now) from shared memory.
// This is for end-effector orientation calculations.

for (i = 0; i < 6; i++)
{
    qJoints[i] = parmR->armRight[i];
}

// timestamp

now = time(NULL);

fprintf(fp, "\n%s\n", ctime(&now));

// Set up Force/Torque Sensor

char *calfilepath;    // name of calibration file
unsigned short index; // index of calibration in file
Calibration *cal;     // struct containing calibration information
short sts;            // return value from functions

// ATI F/T sensor variables

float SampleBias[7]; // measures preloads on sensor before task

float SampleReading[7]; // raw sensor values as read from comedil

float SampleTT[6]={0,0,0,0,0,0}; //sensor axis transform
// Translate along/about {x translate, y translate, z translate,
// x rotate, y rotate, z rotate}

float FT[6]={0,0,0,0,0,0}; // array to hold the resultant
// force/torque vector.

// comedil variables

```

```

int subdev = 0;                // analog port (comedi1 not used for anything
                                // other than F/T sensor)
int range = 0;                // 0 = +/-10VDC
int aref = AREF_DIFF;        // Differential Input

int n_chans0;
int maxdata0;
comedi_t *device0;
int chan=0;
lsampl_t data0;

device0 = comedi_open("/dev/comedi1");

n_chans0 = comedi_get_n_channels(device0, subdev);

for(chan = 0; chan < n_chans0; ++chan)
{
    maxdata0 = comedi_get_maxdata(device0, subdev, chan);
    comedi_data_read(device0, subdev, chan, range, aref, &data0);
    SampleReading[chan] = comedi_to_phys(data0, comedi_get_range(device0, subdev,
        chan, range), maxdata0);
}

// Set up ATI functions

calfilepath="FT5240.cal";
index = 1;

// create Calibration

cal=createCalibration(calfilepath,index);
if (cal==NULL) {
    printf("\nSpecified calibration could not be loaded.\n");
    scanf(".");
    return 0;
}

// NOTE: BELOW FT SETUP KEPT IN EVENT OF FUTURE USE!

// Set force units.
// This step is optional; by default, the units are inherited from
// the calibration file.

sts=SetForceUnits(cal,"N");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid force units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set torque units.
// This step is optional; by default, the units are inherited from the
// calibration file.

sts=SetTorqueUnits(cal,"N-m");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid torque units"); return 0;
}

```

```

    default: printf("Unknown error"); return 0;
}

// Set tool transform.
// This line is only required if you want to move or rotate the
// sensor's coordinate system.

sts=SetToolTransform(cal,SampleTT,"mm","degrees");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid distance units"); return 0;
    case 3: printf("Invalid angle units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Trajectory begins here.////////////////////////////////////////////////////

for (j = 0; j < 320; j++) // 320 points = 32hz X 10 seconds
{
    // Check forces/torques for contact; terminate if contact above threshold

    for(chan = 0; chan < n_chans0; ++chan)
    {
        maxdata0 = comedi_get_maxdata(device0, subdev, chan);

        comedi_data_read_delayed(device0, subdev, chan, range, aref,
                                &data0, 10000);

        SampleReading[chan] = comedi_to_phys(data0, \
            comedi_get_range(device0, subdev, chan, range), maxdata0);
    }

    // Bias the sensor once only.

    if(biasFlag==1)
    {
        for (i = 0; i < 6; i++)
        {
            SampleBias[i] = SampleReading[i];
        }

        Bias(cal, SampleBias);

        biasFlag = 0;
    }

    // convert a loaded measurement into forces and torques

    ConvertToFT(cal,SampleReading,FT);

    // read current Titan position and write to data file

    for (i = 0; i < 6; i++)

```



```

{
    Data[i] = parmR->armRightCar[i];
}

fprintf(fp, "\n %d %d %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f\n", j, toolOnIN, FT[0], FT[1], FT[2], FT[3], FT[4],
        FT[5], Data[0], Data[1], Data[2], Data[3], Data[4], Data[5]);

// Check forces/torques for contact; terminate if contact above threshold
if (FT[0] < contactThreshold)
{
    senseContact = 1;

    printf("FT trip values\n");

    printf("FT:\n");
    printf("%f %f %f %f %f %f\n\n", \
        FT[0], FT[1], FT[2], FT[3], FT[4], FT[5]);

// timestamp

    now = time(NULL);

    fprintf(fp, "\n%s\n", ctime(&now));

    break;
}

// Calculate incremental positions once through each loop.
qNow[0] = qZero[0] + j * inc * cos(qJoints[0] + qJoints[4]); // X
qNow[1] = qZero[1] + j * inc * sin(qJoints[0] + qJoints[4]); // Y

qNow[2] = qZero[2] + j * inc * \
    sin(qJoints[1] + qJoints[2] + qJoints[3] -.0174); // Z
// (note cumulative joint error offset)

// Don't move the wrist joints

qNow[3] = qZero[3]; // rX stays the same
qNow[4] = qZero[4]; // rY stays the same
qNow[5] = qZero[5]; // rZ stays the same

// Write joint positions back to shared memory.

for (i = 0; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

// Calculate delta position once through each loop (for wrist).
// WARNING: HELD TO ZERO CHANGE.

for (i = 0; i < 6; i++)

```

```

{
    qNowV[i] = qNow[i] - qNowOld[i];
}

// Write joint positions back to shared memory.
// Position uses qNow; orientation uses qNowV.
// 0, 1, 2 are qNow for positions, 3, 4, 5 are qNowV for
// velocities.

for (i = 0; i < 3; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

for (i = 3; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = 0; //qNowV[i];
}

// Flag and write to Cartesian

grabSem(0,&sb,semid);
parmRW->armCtrl.updFlag=1;
parmRW->armCtrl.armMode=CART;

// Xfer current new positions to old positions

for (i = 0; i < 6; i++)
{
    qNowOld[i] = qNow[i];
}

// Return semaphore
retSem(0,&sb, semid);

// Delay to control loop rate
nanosleep(&ts, NULL);

// Loop until j = 320 or trigger
}

// timestamp
now = time(NULL);
fprintf(fp, "\n%s\n", ctime(&now));

// exit mode...clean up and get out
grabSem(0,&sb,semid);
parmRW->armCtrl.armMode=IDLE;
retSem(0,&sb, semid);

// free memory allocated to Calibration structure
destroyCalibration(cal);

```

```
        comed_i_close(device0);  
return 0;  
}
```

```

/*****
*
*   bApproachH.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*
*
*****/

/*****
*
*   Obligatory Acknowledgements for libraries used in this file.
*
*   ATIDAQ F/T C Library
*   v1.0.1
*   Copyright (c) 2001 ATI Industrial Automation
*
*   The MIT License
*
*   Permission is hereby granted, free of charge, to any person obtaining a
*   copy of this software and associated documentation files (the "Software")
*   to deal in the Software without restriction, including without limitation
*   the rights to use, copy, modify, merge, publish, distribute, sublicense,
*   and/or sell copies of the Software, and to permit persons to whom the
*   Software is furnished to do so, subject to the following conditions:
*
*   The above copyright notice and this permission notice shall be included
*   in all copies or substantial portions of the Software.
*
*   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
*   OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
*   MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
*   IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
*   CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
*   TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
*   SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*
*****/

/*****
*   Comedilib
*   Copyright (c) 1999,2000 David A. Schleef <ds@schleef.org>
*
*   This file may be freely modified, distributed, and combined with
*   other software, as long as proper attribution is given in the
*   source code.
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

#include "comediFT.h"

```

```

static int biasFlag = 1;    // for sampleBias switching to initialize F/T
int read_writeIO(void);    // reads comedI0 analog/digital IO

int bApproachH(void)
{
    // System level communications

    int QUIT = 0;

    int shmidR,shmidRW, semid; // IPC idenfifiers
    key_t key_memRW,key_memR, key_sem; // keys for shared mem and semphores.
    struct sembuf sb; // semaphore control structure

    /**/

    void safe_quit(void)
    {
        QUIT=1;
    }

    /**/

    /**/

    int grabSem(int semNum, struct sembuf *sb, int semid)
        //semNum should be zero for this program so far.
    {
        sb->sem_op=-1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)
            // make sure you're using the semaphore when it is necessary.
        {
            perror("semaphore access problem");
            QUIT=1;
        }
        return 1;
    }

    /**/

    int retSem(int semNum, struct sembuf *sb, int semid)
        //semNum should be zero for this program so far.
    {
        sb->sem_op=1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)
        {
            perror("semaphore return problem ");
            QUIT=1;
        }
        return 1;
    }

    /**/

```

```

// Calculate for each DOF; numbers in inches, used in trajectory calcs.

double  qZero[6];    // Start point of robotic move
                    // (where you are now)

double  qNow[6];      // Current calculated point in
                    // robotic trajectory

double  qNowV[6];     // Incremental velocity for wrist
                    // orientations--warning not functional

double  qNowOld[6];   // Used for incremental velocity calcs

// Stored instantaneous joint positions

double  qJoints[7];

// Following joint positions are ordered as follows:
// shoulder azimuth, shoulder pitch, elbow, wrist pitch, wrist yaw,
// wrist roll

double Data[6];       // current manipulator position

int senseContact = 0;

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
//    ts.tv_nsec = 31250000; // 32 hz, not calibrated
    ts.tv_nsec = 24400000; // calibrated for actual runtime 32 hz

// time-stamping variables

    time_t time(time_t *tp);

    time_t now;

// file for data capture

    FILE *fp;

    if ((fp = fopen("approachH_data", "wb"))==NULL)
    {
        printf("Cannot open file.\n");
        exit(1);
    }

// Setup shared memory

    child2( );

// Loop Variables

    int i = 0;
    int j = 0;

    double inc = .015625; // .5 in/sec @ 32 hz
    float contactThreshold = 30.00;

```

```

// Set constraints and scaling.
// Note that positions use 1; orientations use 0.

for (i=0;i<6;i++) //initialize memory
{
    parmRW->armCtrl.axesConstr[i]=1.0;
    parmRW->armCtrl.axesScal[i]=1.0;
    parmRW->armCtrl.armMode=IDLE;
    parmRW->armCtrl.cartesCtrl[i]=parmR->armRightCar[i];
    if(i>2)
    {
        parmRW->armCtrl.cartesCtrl[i]=0.0;
    }
}

// Read the starting Cartesian position (where you are now) from
// shared memory.

for (i = 0; i < 6; i++)
{
    qZero[i] = parmR->armRightCar[i];
}

// Read the starting joint angles (where you are now) from shared memory.
// This is for end-effector orientation calculations.

for (i = 0; i < 6; i++)
{
    qJoints[i] = parmR->armRight[i];
}

// timestamp

now = time(NULL);

fprintf(fp, "\n%s\n", ctime(&now));

// Set up Force/Torque Sensor

char *calfilepath;    // name of calibration file
unsigned short index; // index of calibration in file
Calibration *cal;     // struct containing calibration information
// unsigned short i;   // loop variable used to print results
short sts;            // return value from functions

// ATI F/T sensor variables

float SampleBias[7]; // measures preloads on sensor before starting task

float SampleReading[7]; // raw sensor values as read from comed11

float SampleTT[6]={0,0,0,0,0,0}; //sensor axis transform
// Translate along/about {x translate, y translate, z translate,
// x rotate, y rotate, z rotate}

float FT[6]={0,0,0,0,0,0}; // array to hold the resultant
// force/torque vector.

```

```

// comedil variables

int subdev = 0;           // analog port (comedil not used for anything
                          // other than F/T sensor)
int range = 0;           // 0 = +/-10VDC
int aref = AREF_DIFF;    // Differential Input

int n_chans0;
int maxdata0;
comedil_t *device0;
int chan=0;
lsampl_t data0;

device0 = comedil_open("/dev/comedil");

n_chans0 = comedil_get_n_channels(device0, subdev);

for(chan = 0; chan < n_chans0; ++chan)
{
    maxdata0 = comedil_get_maxdata(device0, subdev, chan);
    comedil_data_read(device0, subdev, chan, range, aref, &data0);
    SampleReading[chan] = comedil_to_phys(data0, comedil_get_range(device0, subdev,
        chan, range), maxdata0);
}

// Set up ATI functions

calfilepath="FT5240.cal";
index = 1;

// create Calibration

cal=createCalibration(calfilepath,index);
if (cal==NULL) {
    printf("\nSpecified calibration could not be loaded.\n");
    scanf(".");
    return 0;
}

// NOTE: BELOW FT SETUP KEPT IN EVENT OF FUTURE USE!

// Set force units.
// This step is optional; by default, the units are inherited from
// the calibration file.

sts=SetForceUnits(cal,"N");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid force units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set torque units.
// This step is optional; by default, the units are inherited from the
// calibration file.

sts=SetTorqueUnits(cal,"N-m");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;

```



```

    case 2: printf("Invalid torque units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set tool transform.
// This line is only required if you want to move or rotate the
// sensor's coordinate system.

sts=SetToolTransform(cal,SampleTT,"mm","degrees");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid distance units"); return 0;
    case 3: printf("Invalid angle units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Trajectory begins here.////////////////////////////////////////////////////

for (j = 0; j < 320; j++) // 320 points = 32hz X 10 seconds
{
    // Check forces/torques for contact; terminate if contact above threshold

    for(chan = 0; chan < n_chans0; ++chan)
    {
        maxdata0 = comedi_get_maxdata(device0, subdev, chan);

        comedi_data_read_delayed(device0, subdev, chan, range, aref,
                                &data0, 10000);

        SampleReading[chan] = comedi_to_phys(data0, \
            comedi_get_range(device0, subdev, chan, range), maxdata0);
    }

    // Bias the sensor once only.

    if(biasFlag==1)
    {
        for (i = 0; i < 6; i++)
        {
            SampleBias[i] = SampleReading[i];
        }

        Bias(cal, SampleBias);

        biasFlag = 0;
    }

    // convert a loaded measurement into forces and torques

    ConvertToFT(cal,SampleReading,FT);

```

```

// read current Titan position and write to data file
for (i = 0; i < 6; i++)
{
    Data[i] = parmR->armRightCar[i];
}

fprintf(fp, "\n %d %d %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f
%9.6f %9.6f %9.6f\n", j, toolOnIN, FT[0], FT[1], FT[2], FT[3], FT[4],
FT[5], Data[0], Data[1], Data[2], Data[3], Data[4], Data[5]);

// Calculate incremental positions once through each loop.
qNow[0] = qZero[0] + j * inc * cos(qJoints[0] + qJoints[4]); // X
qNow[1] = qZero[1] + j * inc * sin(qJoints[0] + qJoints[4]); // Y
qNow[2] = qZero[2] + j * inc * \
sin(qJoints[1] + qJoints[2] + qJoints[3] -.0174); // Z
// (note cumulative joint error offset)

// Don't move the wrist joints
qNow[3] = qZero[3];           // rX stays the same
qNow[4] = qZero[4];           // rY stays the same
qNow[5] = qZero[5];           // rZ stays the same

// Write joint positions back to shared memory.
for (i = 0; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

// Calculate delta position once through each loop (for wrist).
// WARNING: NOT FUNCTIONAL AT THIS TIME; HELD TO ZERO CHANGE.
for (i = 0; i < 6; i++)
{
    qNowV[i] = qNow[i] - qNowOld[i];
}

// Write joint positions back to shared memory.
for (i = 0; i < 3; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

for (i = 3; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = 0; //qNowV[i];
}

```

```

// Flag and write to Cartesian
grabSem(0,&sb,semid);
parmRW->armCtrl.updFlag=1;
parmRW->armCtrl.armMode=CART;

// Xfer current new positions to old positions
for (i = 0; i < 6; i++)
{
    qNowOld[i] = qNow[i];
}

// Return semaphore
retSem(0,&sb, semid);

// Delay to control loop rate
nanosleep(&ts, NULL);

// Check forces/torques for contact; terminate if contact above threshold

if (((fabs(FT[0])) > contactThreshold) || ((fabs(FT[1])) > contactThreshold) ||
((fabs(FT[2])) > contactThreshold))
{
    senseContact = 1;
    printf("FT trip values\n");

    printf("FT:\n");
    printf("%f    %f    %f    %f    %f    %f\n\n", \
    FT[0], FT[1], FT[2], FT[3], FT[4], FT[5]);

// timestamp

    now = time(NULL);
    fprintf(fp, "\n%s\n",ctime(&now));

    j = 320;
}

// Loop
}

// timestamp

now = time(NULL);
fprintf(fp, "\n%s\n",ctime(&now));

// exit mode...clean up and get out

grabSem(0,&sb,semid);
parmRW->armCtrl.armMode=IDLE;

```

```
    retSem(0,&sb, semid);  
    // free memory allocated to Calibration structure  
    destroyCalibration(cal);  
    comedi_close(device0);  
    return 0;  
}
```

```

/*****
*
*   bApproachV.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*
*
*****/

/*****
*
*   Obligatory Acknowledgements for libraries used in this file.
*
*   ATIDAQ F/T C Library
*   v1.0.1
*   Copyright (c) 2001 ATI Industrial Automation
*
*   The MIT License
*
*   Permission is hereby granted, free of charge, to any person obtaining a
*   copy of this software and associated documentation files (the "Software")
*   to deal in the Software without restriction, including without limitation
*   the rights to use, copy, modify, merge, publish, distribute, sublicense,
*   and/or sell copies of the Software, and to permit persons to whom the
*   Software is furnished to do so, subject to the following conditions:
*
*   The above copyright notice and this permission notice shall be included
*   in all copies or substantial portions of the Software.
*
*   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
*   OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
*   MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
*   IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
*   CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
*   TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
*   SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*
*****/

/*****
*   Comedilib
*   Copyright (c) 1999,2000 David A. Schleef <ds@schleef.org>
*
*   This file may be freely modified, distributed, and combined with
*   other software, as long as proper attribution is given in the
*   source code.
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

#include "comediFT.h"

```

```

static int biasFlag = 1;    // for sampleBias switching initializing the F/T
int read_writeIO(void);    // reads comedi0 analog/digital IO
int bApproachV(void)
{
    // System level communications
    int QUIT = 0;

    int shmidR,shmidRW, semid;    // IPC idenfifiers
    key_t key_memRW,key_memR, key_sem;// keys for shared mem and semphores.
    struct sembuf sb; // semaphore control structure

    /**/

void safe_quit(void)
{
    QUIT=1;
}

    /**/
    /**/

int grabSem(int semNum,  struct sembuf *sb, int semid)
    //semNum should be zero for this program so far.
{
    sb->sem_op=-1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
        // make sure you're using the semaphore when it is necessary.
        {
            perror("semaphore access problem");
            QUIT=1;
        }
    return 1;
}

    /**/

int retSem(int semNum,  struct sembuf *sb, int semid)
    //semNum should be zero for this program so far.
{
    sb->sem_op=1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
        {
            perror("semaphore return problem ");
            QUIT=1;
        }
    return 1;
}

    /**/

    // Calculate for each DOF; numbers in inches, used in trajectory calcs.

```

```

double  qZero[6];    // Start point of robotic move
                  // (where you are now)

double  qNow[6];     // Current calculated point in
                  // robotic trajectory

double  qNowV[6];    // Incremental velocity for wrist
                  // orientations--warning not functional

double  qNowOld[6];  // Used for incremental velocity calcs

// Stored instantaneous joint positions

extern double qJoints[7];

// Following joint positions are ordered as follows:
// shoulder azimuth, shoulder pitch, elbow, wrist pitch, wrist yaw,
// wrist roll

    double Data[6]; // current manipulator position

    int senseContact = 0;

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
//    ts.tv_nsec = 31250000; // 32 hz, not calibrated
    ts.tv_nsec = 24400000; // calibrated for actual runtime 32 hz

// time-stamping variables

    time_t time(time_t *tp);

    time_t now;

// file for data capture

    FILE *fp;

    if ((fp = fopen("approachV_data", "wb"))==NULL)
    {
        printf("Cannot open file.\n");
        exit(1);
    }

// Setup shared memory

    child2( );

// Loop Variables

    int i = 0;
    int j = 0;

    double inc = .015625; // .5 in/sec @ 32 hz
    float contactThreshold = .50;

    // Set constraints and scaling.
    // Note that positions use 1; orientations use 0.

```

```

    for (i=0;i<6;i++) //initialize memory
    {
        parmRW->armCtrl.axesConstr[i]=1.0;
        parmRW->armCtrl.axesScal[i]=1.0;
        parmRW->armCtrl.armMode=IDLE;
        parmRW->armCtrl.cartesCtrl[i]=parmR->armRightCar[i];
        if(i>2)
        {
            parmRW->armCtrl.cartesCtrl[i]=0.0;
        }
    }

// Read the starting Cartesian position (where you are now) from
// shared memory.

    for (i = 0; i < 6; i++)
    {
        qZero[i] = parmR->armRightCar[i];
    }

// Read the starting joint angles (where you are now) from shared memory.
// This is for end-effector orientation calculations.

    for (i = 0; i < 6; i++)
    {
        qJoints[i] = parmR->armRight[i];
    }

// timestamp

    now = time(NULL);

    fprintf(fp, "\n%s\n", ctime(&now));

// Set up Force/Torque Sensor

    char *calfilepath;    // name of calibration file
    unsigned short index; // index of calibration in file (second parameter;
                        // default = 1)
    Calibration *cal;     // struct containing calibration information
// unsigned short i;     // loop variable used to print results
    short sts;           // return value from functions

// ATI F/T sensor variables

    float SampleBias[7]; // measures preloads on sensor before starting task

    float SampleReading[7]; // raw sensor values as read from comed11

    float SampleTT[6]={0,0,0,0,0,0}; //sensor axis transform
// Translate along/about {x translate, y translate, z translate,
// x rotate, y rotate, z rotate}

    float FT[6]={0,0,0,0,0,0}; // array to hold the resultant
                        // force/torque vector.

```



```

// comedil variables

int subdev = 0;           // analog port (comedil not used for anything
                          // other than F/T sensor)
int range = 0;           // 0 = +/-10VDC
int aref = AREF_DIFF;    // Differential Input

int n_chans0;
int maxdata0;
comedit *device0;
int chan=0;
lsampl_t data0;

device0 = comedil_open("/dev/comedil");

n_chans0 = comedil_get_n_channels(device0, subdev);

for(chan = 0; chan < n_chans0; ++chan)
{
    maxdata0 = comedil_get_maxdata(device0, subdev, chan);
    comedil_data_read(device0, subdev, chan, range, aref, &data0);
    SampleReading[chan] = comedil_to_phys(data0, comedil_get_range(device0, subdev,
chan, range), maxdata0);
}

// Set up ATI functions

calfilepath="FT5240.cal";
index = 1;

// create Calibration

cal=createCalibration(calfilepath,index);
if (cal==NULL) {
    printf("\nSpecified calibration could not be loaded.\n");
    scanf(".");
    return 0;
}

// NOTE: BELOW FT SETUP KEPT IN EVENT OF FUTURE USE!

// Set force units.
// This step is optional; by default, the units are inherited from
// the calibration file.

sts=SetForceUnits(cal,"N");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid force units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set torque units.
// This step is optional; by default, the units are inherited from the
// calibration file.

sts=SetTorqueUnits(cal,"N-m");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;

```

```

    case 2: printf("Invalid torque units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set tool transform.
// This line is only required if you want to move or rotate the
// sensor's coordinate system.

sts=SetToolTransform(cal,SampleTT,"mm","degrees");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid distance units"); return 0;
    case 3: printf("Invalid angle units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Trajectory begins here.////////////////////////////////////////////////////

for (j = 0; j < 640; j++) // 640 points = 32hz X 20 seconds
{
    // Check forces/torques for contact; terminate if contact above threshold

    for(chan = 0; chan < n_chans0; ++chan)
    {
        maxdata0 = comedi_get_maxdata(device0, subdev, chan);

        comedi_data_read_delayed(device0, subdev, chan, range, aref,
                                &data0, 10000);

        SampleReading[chan] = comedi_to_phys(data0, \
            comedi_get_range(device0, subdev, chan, range), maxdata0);
    }

    // Bias the sensor once only.

    if(biasFlag==1)
    {
        for (i = 0; i < 6; i++)
        {
            SampleBias[i] = SampleReading[i];
        }

        Bias(cal, SampleBias);

        biasFlag = 0;
    }

    // convert a loaded measurement into forces and torques

    ConvertToFT(cal,SampleReading,FT);

    // read current Titan position and write to data file

```

```

for (i = 0; i < 6; i++)
{
    Data[i] = parmR->armRightCar[i];
}

fprintf(fp, "\n %d %d %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f\n", j, toolOnIN, FT[0], FT[1], FT[2], FT[3], FT[4],
        FT[5], Data[0], Data[1], Data[2], Data[3], Data[4], Data[5]);

// Calculate incremental positions once through each loop.
qNow[0] = qZero[0]; // X
qNow[1] = qZero[1]; // Y
qNow[2] = qZero[2] - j * (inc/4.0); // Z
// (note cumulative joint error offset)

// Don't move the wrist joints
qNow[3] = qZero[3]; // rX stays the same
qNow[4] = qZero[4]; // rY stays the same
qNow[5] = qZero[5]; // rZ stays the same

// Write joint positions back to shared memory.
for (i = 0; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

// Calculate delta position once through each loop (for wrist).
// WARNING: NOT FUNCTIONAL AT THIS TIME; HELD TO ZERO CHANGE.

for (i = 0; i < 6; i++)
{
    qNowV[i] = qNow[i] - qNowOld[i];
}

// Write joint positions back to shared memory.
for (i = 0; i < 3; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

for (i = 3; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = 0; //qNowV[i];
}

// Flag and write to Cartesian
grabSem(0,&sb,semid);
parmRW->armCtrl.updFlag=1;
parmRW->armCtrl.armMode=CART;

```

```

    // Xfer current new positions to old positions
    for (i = 0; i < 6; i++)
    {
        qNowOld[i] = qNow[i];
    }

    // Return semaphore
    retSem(0,&sb, semid);

    // Delay to control loop rate
    // Check forces/torques for contact; terminate if contact above threshold

    if (FT[4] > contactThreshold)
    {
        senseContact = 1;
        printf("FT trip values\n");

        printf("FT:\n");
        printf("%f    %f    %f    %f    %f    %f\n\n", \
            FT[0], FT[1], FT[2], FT[3], FT[4], FT[5]);
    // timestamp

        now = time(NULL);
        fprintf(fp, "\n%s\n", ctime(&now));

        j = 640;
    }

    nanosleep(&ts, NULL);

    // Loop until j = 640
}

// timestamp

now = time(NULL);
fprintf(fp, "\n%s\n", ctime(&now));

// exit mode...clean up and get out

grabSem(0,&sb, semid);
parmRW->armCtrl.armMode=IDLE;
retSem(0,&sb, semid);

// free memory allocated to Calibration structure
destroyCalibration(cal);

```

```
    comed_i_close(device0);  
    return 0;  
}
```

```

/*****
*
*   bBackH.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*****/

/*****
*
*   Obligatory Acknowledgements for libraries used in this file.
*
*   ATIDAQ F/T C Library
*   v1.0.1
*   Copyright (c) 2001 ATI Industrial Automation
*
*   The MIT License
*
*   Permission is hereby granted, free of charge, to any person obtaining a
*   copy of this software and associated documentation files (the "Software")
*   to deal in the Software without restriction, including without limitation
*   the rights to use, copy, modify, merge, publish, distribute, sublicense,
*   and/or sell copies of the Software, and to permit persons to whom the
*   Software is furnished to do so, subject to the following conditions:
*
*   The above copyright notice and this permission notice shall be included
*   in all copies or substantial portions of the Software.
*
*   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
*   OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
*   MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
*   IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
*   CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
*   TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
*   SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*
*****/

/*****
*   Comedilib
*   Copyright (c) 1999,2000 David A. Schlee <ds@schlee.org>
*
*   This file may be freely modified, distributed, and combined with
*   other software, as long as proper attribution is given in the
*   source code.
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

#include "comediFT.h"

```

```

static int biasFlag = 1;    // for sampleBias switching initializing F/T
int read_writeIO(void);    // reads comedi0 analog/digital IO
int bBackH(void)
{
    int QUIT = 0;

    int shmidR,shmidRW, semid; // IPC identifiers
    key_t key_memRW,key_memR, key_sem; // keys for shared mem and semphores.
    struct sembuf sb; // semaphore control structure

    /**/

void safe_quit(void)
{
    QUIT=1;
}

/**/

/**/

int grabSem(int semNum, struct sembuf *sb, int semid)
//semNum should be zero for this program so far.
{
    sb->sem_op=-1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    // make sure you're using the semaphore when it is necessary.
    {
        perror("semaphore access problem");
        QUIT=1;
    }
    return 1;
}

/**/

int retSem(int semNum, struct sembuf *sb, int semid)
//semNum should be zero for this program so far.
{
    sb->sem_op=1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    {
        perror("semaphore return problem ");
        QUIT=1;
    }
    return 1;
}

/**/

// Calculate for each DOF; numbers in inches, used in trajectory calcs.

double    qZero[6];        // Start point of robotic move
                        // (where you are now)

```

```

double  qNow[6];      // Current calculated point in
                      // robotic trajectory

double  qNowV[6];     // Incremental velocity for wrist
                      // orientations

double  qNowOld[6];   // Used for incremental velocity calcs

// Stored instantaneous joint positions

double  qJoints[7];

// Following joint positions are ordered as follows:
// shoulder azimuth, shoulder pitch, elbow, wrist pitch, wrist yaw,
// wrist roll

double  Data[6];      // current manipulator position

// Position increment instead of time but run at sample time.

int  senseContact = 0;

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
//    ts.tv_nsec = 31250000; // set to 32 hz
    ts.tv_nsec = 24400000; // calibrated for actual runtime 32 hz

// time-stamping variables

    time_t time(time_t *tp);

    time_t now;

// file for data capture

    FILE *fp;

    if ((fp = fopen("backH_data", "wb"))==NULL)
    {
        printf("Cannot open file.\n");
        exit(1);
    }

// Setup shared memory

    child2( );

// Set up Force/Torque Sensor

    char *calfilepath;    // name of calibration file
    unsigned short index; // index of calibration in file
    Calibration *cal;     // struct containing calibration information
    short sts;            // return value from functions

```



```

// ATI F/T sensor variables

float SampleBias[7];    // measures preloads on sensor before starting

float SampleReading[7]; // raw sensor values as read from comed1l

float SampleTT[6]={0,0,0,0,0,0};    //sensor axis transform
// Translate along/about {x translate, y translate, z translate,
// x rotate, y rotate, z rotate}

float FT[6]={0,0,0,0,0,0};    // array to hold the resultant
// force/torque vector.

// comed1l variables

int subdev = 0;                // analog port (comed1l not used for anything
// other than F/T sensor)
int range = 0;                // 0 = +/-10VDC
int aref = AREF_DIFF;        // Differential Input

int n_chans0;
int maxdata0;
comed1_t *device0;
int chan=0;
lsampl_t data0;

device0 = comed1_open("/dev/comed1l");

n_chans0 = comed1_get_n_channels(device0, subdev);

for(chan = 0; chan < n_chans0; ++chan)
{
    maxdata0 = comed1_get_maxdata(device0, subdev, chan);
    comed1_data_read(device0, subdev, chan, range, aref, &data0);
    SampleReading[chan] = comed1_to_phys(data0, comed1_get_range(device0, subdev,
        chan, range), maxdata0);
}

// Set up ATI functions

calfilepath="FT5240.cal";
index = 1;

// create Calibration

cal=createCalibration(calfilepath,index);
if (cal==NULL) {
    printf("\nSpecified calibration could not be loaded.\n");
    scanf(".");
    return 0;
}

// NOTE: BELOW FT SETUP KEPT IN EVENT OF FUTURE USE!

// Set force units.
// This step is optional; by default, the units are inherited from
// the calibration file.

sts=SetForceUnits(cal,"N");
switch (sts) {
    case 0: break; // successful completion

```

```

    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid force units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set torque units.
// This step is optional; by default, the units are inherited from the
// calibration file.

sts=SetTorqueUnits(cal,"N-m");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid torque units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set tool transform.
// This line is only required if you want to move or rotate the
// sensor's coordinate system.

sts=SetToolTransform(cal,SampleTT,"mm","degrees");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid distance units"); return 0;
    case 3: printf("Invalid angle units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Variables

int i = 0;
int j = 0;
int k = 16;

double inc = .015625; // .5 in/sec @ 32 hz
float contactThreshold = 20.00;

// Set constraints and scaling. Note that positions use 1;
// orientations use 0.

for (i=0;i<6;i++) //initialize memory
{
    parmRW->armCtrl.axesConstr[i]=1.0;
    parmRW->armCtrl.axesScal[i]=1.0;
    parmRW->armCtrl.armMode=IDLE;
    parmRW->armCtrl.cartesCtrl[i]=parmR->armRightCar[i];
    if(i>2)
    {
        parmRW->armCtrl.cartesCtrl[i]=0.0;
    }
}

// Read the starting Cartesian position (where you are now) from
// shared memory.

for (i = 0; i < 6; i++)
{

```

```

    qZero[i] = parmR->armRightCar[i];
}

// Read the starting joint angles (where you are now) from shared memory.
// This is for end-effector orientation calculations.

    for (i = 0; i < 6; i++)
    {
        qJoints[i] = parmR->armRight[i];
    }

// timestamp

    now = time(NULL);
    fprintf(fp, "\n%s\n", ctime(&now));

// Trajectory begins here.////////////////////////////////////////////////////

    for (j = 0; j < 64; j++) // back away from pipe after contact
    {

        // Check forces/torques for contact; terminate if contact above
        // threshold and minimum distance is reached.

        for(chan = 0; chan < n_chans0; ++chan)
        {

            maxdata0 = comedi_get_maxdata(device0, subdev, chan);

            comedi_data_read_delayed(device0, subdev, chan, range, aref, &data0,
                                   10000);

            SampleReading[chan] = comedi_to_phys(data0, comedi_get_range(device0,
                                   subdev, chan, range), maxdata0);

        }

        // Bias the sensor once only.

        if(biasFlag==1)
        {
            for (i = 0; i < 6; i++)
            {

                SampleBias[i] = SampleReading[i];

            }

            Bias(cal, SampleBias);

            biasFlag = 0;
        }

        // convert a loaded measurement into forces and torques

        ConvertToFT(cal, SampleReading, FT);

// read current Titan position and write to data file

```

```

for (i = 0; i < 6; i++)
{
    Data[i] = parmR->armRightCar[i];
}

fprintf(fp, "\n %d %d %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f
%9.6f %9.6f %9.6f\n", j, toolOnIN, FT[0], FT[1], FT[2], FT[3], FT[4],
FT[5], Data[0], Data[1], Data[2], Data[3], Data[4], Data[5]);

// Calculate incremental positions once through each loop.

qNow[0] = qZero[0] - j * inc * cos(qJoints[0] + qJoints[4]); // X
qNow[1] = qZero[1] - j * inc * sin(qJoints[0] + qJoints[4]); // Y

qNow[2] = qZero[2] - j * inc * \
sin(qJoints[1] + qJoints[2] + qJoints[3] -.0174); // Z
// (note cumulative joint error offset)

// Don't move the wrist joints

qNow[3] = qZero[3]; // rX stays the same
qNow[4] = qZero[4]; // rY stays the same
qNow[5] = qZero[5]; // rZ stays the same

// Write joint positions back to shared memory.

for (i = 0; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

// Calculate delta position once through each loop (for wrist).

for (i = 0; i < 6; i++)
{
    qNowV[i] = qNow[i] - qNowOld[i];
}

// Write joint positions back to shared memory.
// Position uses qNow; orientation uses qNowV.
// 0, 1, 2 are qNow for positions;
// 3, 4, 5 are qNowV for velocities.

for (i = 0; i < 3; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

for (i = 3; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = 0; //qNowV[i];
}

```

```

// Flag and write to Cartesian
grabSem(0,&sb,semid);
parmRW->armCtrl.updFlag=1;
parmRW->armCtrl.armMode=CART;

// Xfer current new positions to old positions
for (i = 0; i < 6; i++)
{
    qNowOld[i] = qNow[i];
}

// Return semaphore
retSem(0,&sb, semid);

// Check forces/torques for contact; terminate if contact above threshold
if (FT[0] > contactThreshold)
{
    if (senseContact == 0)
    {
        printf("FT trip values\n");
        printf("FT:\n");
        printf("%d %f %f %f %f %f\n\n", \
j, FT[0], FT[1], FT[2], FT[3], FT[4], FT[5]);

        // timestamp
        now = time(NULL);
        fprintf(fp, "\n%s\n",ctime(&now));

        senseContact = 1;
    }

    if (k == 0)
    {
        j = 64;
    }

    k = k - 1;
}

// Delay to control loop rate
nanosleep(&ts, NULL);

// Loop

```

```
    }  
  
    // timestamp  
    now = time(NULL);  
    fprintf(fp, "\n%s\n", ctime(&now));  
  
    // exit mode...clean up and get out  
    grabSem(0, &sb, semid);  
    parmRW->armCtrl.armMode=IDLE;  
    retSem(0, &sb, semid);  
}
```

```

/*****
*
*   bBackV.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*****/

/*****
*
*   Obligatory Acknowledgements for libraries used in this file.
*
*   ATIDAQ F/T C Library
*   v1.0.1
*   Copyright (c) 2001 ATI Industrial Automation
*
*   The MIT License
*
*   Permission is hereby granted, free of charge, to any person obtaining a
*   copy of this software and associated documentation files (the "Software")
*   to deal in the Software without restriction, including without limitation
*   the rights to use, copy, modify, merge, publish, distribute, sublicense,
*   and/or sell copies of the Software, and to permit persons to whom the
*   Software is furnished to do so, subject to the following conditions:
*
*   The above copyright notice and this permission notice shall be included
*   in all copies or substantial portions of the Software.
*
*   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
*   OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
*   MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
*   IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
*   CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
*   TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
*   SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*
*****/

/*****
*   Comedilib
*   Copyright (c) 1999,2000 David A. Schlee <ds@schlee.org>
*
*   This file may be freely modified, distributed, and combined with
*   other software, as long as proper attribution is given in the
*   source code.
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

#include "comediFT.h"

```

```

static int biasFlag = 1; // for sampleBias switching initializing F/T
int read_writeIO(void); // reads comedi0 analog/digital IO

int bBackV(void)
{
    int QUIT = 0;

    int shmidR,shmidRW, semid; // IPC identifiers
    key_t key_memRW,key_memR, key_sem; // keys for shared mem and semphores.
    struct sembuf sb; // semaphore control structure

    /**/

void safe_quit(void)
{
    QUIT=1;
}

/**/

/**/

int grabSem(int semNum, struct sembuf *sb, int semid)
//semNum should be zero for this program so far.
{
    sb->sem_op=-1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    // make sure you're using the semaphore when it is necessary.
    {
        perror("semaphore access problem");
        QUIT=1;
    }
    return 1;
}

/**/

int retSem(int semNum, struct sembuf *sb, int semid)
//semNum should be zero for this program so far.
{
    sb->sem_op=1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    {
        perror("semaphore return problem ");
        QUIT=1;
    }
    return 1;
}

/**/

```



```

// Calculate for each DOF; numbers in inches, used in trajectory calcs.

double qZero[6];    // Start point of robotic move
                   // (where you are now)

double qNow[6];     // Current calculated point in
                   // robotic trajectory

double qNowV[6];    // Incremental velocity for wrist
                   // orientations

double qNowOld[6];  // Used for incremental velocity calcs

// Stored instantaneous joint positions

double qJoints[7];

// Following joint positions are ordered as follows:
// shoulder azimuth, shoulder pitch, elbow, wrist pitch, wrist yaw,
// wrist roll

    double Data[6]; // current manipulator position

// Position increment instead of time but run at sample time.

    int senseContact = 0;

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
//    ts.tv_nsec = 31250000; // set to 32 hz
    ts.tv_nsec = 24400000; // calibrated for actual runtime 32 hz

// time-stamping variables

    time_t time(time_t *tp);

    time_t now;

// file for data capture

    FILE *fp;

    if ((fp = fopen("backV_data", "wb"))==NULL)
    {
        printf("Cannot open file.\n");
        exit(1);
    }

// Setup shared memory

    child2( );

```

```

// Set up Force/Torque Sensor

char *calfilepath;    // name of calibration file
unsigned short index; // index of calibration in file
Calibration *cal;     // struct containing calibration information

short sts;            // return value from functions

// ATI F/T sensor variables

float SampleBias[7]; // measures preloads on sensor before starting

float SampleReading[7]; // raw sensor values as read from comedil

float SampleTT[6]={0,0,0,0,0,0}; //sensor axis transform
// Translate along/about {x translate, y translate, z translate,
// x rotate, y rotate, z rotate}

float FT[6]={0,0,0,0,0,0}; // array to hold the resultant
// force/torque vector.

// comedil variables

int subdev = 0;          // analog port (comedil not used for anything
                        // other than F/T sensor)
int range = 0;          // 0 = +/-10VDC
int aref = AREF_DIFF;   // Differential Input

int n_chans0;
int maxdata0;
comedil_t *device0;
int chan=0;
lsampl_t data0;

device0 = comedil_open("/dev/comedil");

n_chans0 = comedil_get_n_channels(device0, subdev);

for(chan = 0; chan < n_chans0; ++chan)
{
    maxdata0 = comedil_get_maxdata(device0, subdev, chan);
    comedil_data_read(device0, subdev, chan, range, aref, &data0);
    SampleReading[chan] = comedil_to_phys(data0, comedil_get_range(device0, subdev,
        chan, range), maxdata0);
}

// Set up ATI functions

calfilepath="FT5240.cal";
index = 1;

// create Calibration

cal=createCalibration(calfilepath,index);
if (cal==NULL) {
    printf("\nSpecified calibration could not be loaded.\n");
    scanf(".");
    return 0;
}

```

```

// NOTE: BELOW FT SETUP KEPT IN EVENT OF FUTURE USE!

// Set force units.
// This step is optional; by default, the units are inherited from
// the calibration file.

sts=SetForceUnits(cal,"N");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid force units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set torque units.
// This step is optional; by default, the units are inherited from the
// calibration file.

sts=SetTorqueUnits(cal,"N-m");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid torque units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set tool transform.
// This line is only required if you want to move or rotate the
// sensor's coordinate system.

sts=SetToolTransform(cal,SampleTT,"mm","degrees");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid distance units"); return 0;
    case 3: printf("Invalid angle units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Variables

int i = 0;
int j = 0;
int k = 32;

double inc = .015625; // .5 in/sec @ 32 hz
float contactThreshold = 0.00; //

// Set constraints and scaling. Note that positions use 1; orientations
// use 0.

for (i=0;i<6;i++) //initialize memory
{
    parmRW->armCtrl.axesConstr[i]=1.0;
    parmRW->armCtrl.axesScal[i]=1.0;
    parmRW->armCtrl.armMode=IDLE;
    parmRW->armCtrl.cartesCtrl[i]=parmR->armRightCar[i];
    if(i>2)
    {

```

```

        parmRW->armCtrl.cartesCtrl[i]=0.0;
    }

}

// Read the starting Cartesian position (where you are now) from
// shared memory.

    for (i = 0; i < 6; i++)
    {
        qZero[i] = parmR->armRightCar[i];
    }

// Read the starting joint angles (where you are now) from shared memory.
// This is for end-effector orientation calculations.

    for (i = 0; i < 6; i++)
    {
        qJoints[i] = parmR->armRight[i];
    }

// timestamp

    now = time(NULL);
    fprintf(fp, "\n%s\n", ctime(&now));

// Trajectory begins here.////////////////////////////////////////////////////

    for (j = 0; j < 320; j++) // back away from pipe after contact
    {

// Check forces/torques for contact; terminate if contact above threshold

        for(chan = 0; chan < n_chans0; ++chan)
        {

            maxdata0 = comedi_get_maxdata(device0, subdev, chan);

            comedi_data_read_delayed(device0, subdev, chan, range, aref,      &data0,
                                   10000);

            SampleReading[chan] = comedi_to_phys(data0, comedi_get_range(device0,
                                   subdev, chan, range), maxdata0);

        }

// Bias the sensor once only.

        if(biasFlag==1)
        {
            for (i = 0; i < 6; i++)
            {
                SampleBias[i] = SampleReading[i];
            }

            Bias(cal, SampleBias);

```

```

        biasFlag = 0;
    }

    // convert a loaded measurement into forces and torques
    ConvertToFT(cal, SampleReading, FT);

    // read current Titan position and write to data file
    for (i = 0; i < 6; i++)
    {
        Data[i] = parmR->armRightCar[i];
    }

    fprintf(fp, "\n %d %d %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f
        %9.6f %9.6f %9.6f\n", j, toolOnIN, FT[0], FT[1], FT[2], FT[3], FT[4],
        FT[5], Data[0], Data[1], Data[2], Data[3], Data[4], Data[5]);

    // Calculate incremental positions once through each loop.
    qNow[0] = qZero[0];                // X stays the same
    qNow[1] = qZero[1];                // Y stays the same
    qNow[2] = qZero[2] + j * (inc/4.0); // Z moves positive
    // Don't move the wrist joints

    qNow[3] = qZero[3];                // rX stays the same
    qNow[4] = qZero[4];                // rY stays the same
    qNow[5] = qZero[5];                // rZ stays the same

    // Write joint positions back to shared memory.
    for (i = 0; i < 6; i++)
    {
        parmRW->armCtrl.cartesCtrl[i] = qNow[i];
    }

    // Calculate delta position once through each loop (for wrist).
    for (i = 0; i < 6; i++)
    {
        qNowV[i] = qNow[i] - qNowOld[i];
    }

    // Write joint positions back to shared memory.
    // Position uses qNow; orientation uses qNowV.
    // 0, 1, 2 are qNow for positions, 3, 4, 5 are qNowV for
    // velocities.
    for (i = 0; i < 3; i++)
    {
        parmRW->armCtrl.cartesCtrl[i] = qNow[i];
    }

```

```

for (i = 3; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = 0; //qNowV[i];
}

// Flag and write to Cartesian

grabSem(0,&sb,semid);
parmRW->armCtrl.updFlag=1;
parmRW->armCtrl.armMode=CART;

// Xfer current new positions to old positions

for (i = 0; i < 6; i++)
{
    qNowOld[i] = qNow[i];
}

// Return semaphore

retSem(0,&sb, semid);

// Check forces/torques for contact
// Terminate if contact above threshold and momentum goes to 0

if (FT[4] < contactThreshold)
{
    if (senseContact == 0)
    {
        printf("FT trip values\n");

        printf("FT:\n");
        printf("%d %f %f %f %f %f\n\n", \
j, FT[0], FT[1], FT[2], FT[3], FT[4], FT[5]);

        // timestamp

        now = time(NULL);

        fprintf(fp, "\n%s\n",ctime(&now));

        senseContact = 1;
    }

    if (k == 0)
    {
        j = 320;
    }

    k = k - 1; // Simulates momentum to guarantee FT sensor
               // clear of contact

```

```
    }  
    // Delay to control loop rate  
    nanosleep(&ts, NULL);  
    // Loop  
}  
  
// timestamp  
    now = time(NULL);  
    fprintf(fp, "\n%s\n", ctime(&now));  
  
// exit mode...clean up and get out  
grabSem(0,&sb,semid);  
parmRW->armCtrl.armMode=IDLE;  
retSem(0,&sb, semid);  
  
return 0;  
}
```

```

/*****
*
*   bRetractB.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical,
Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

int read_writeIO(void); // reads comedio analog/digital IO

int bRetractB(void)
{
    int QUIT = 0;

    int shmidR,shmidRW, semid; // IPC identifiers
    key_t key_memRW,key_memR, key_sem; // keys for shared mem and semaphores.
    struct sembuf sb; // semaphore control structure

    /***/

    void safe_quit(void)
    {
        QUIT=1;
    }

    /***/

    /***/

    int grabSem(int semNum, struct sembuf *sb, int semid)
    //semNum should be zero for this program.
    {
        sb->sem_op=-1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)

        {
            perror("semaphore access problem");
            QUIT=1;
        }
        return 1;
    }
}

```



```

//*****

int retSem(int semNum, struct sembuf *sb, int semid)
//semNum should be zero for this program.

{
    sb->sem_op=1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    {
        perror("semaphore return problem ");
        QUIT=1;
    }
    return 1;
}

//*****

// Calculate for each DOF; numbers in inches, used in trajectory calcs.

double qZero[6];    // Start point of robotic move
                   // (where you are now)

double qNow[6];      // Current calculated point in
                   // robotic trajectory

double qNowV[6];     // Incremental velocity for wrist
                   // orientations

double qNowOld[6];   // Used for incremental velocity calcs

// Stored instantaneous joint positions

double qJoints[7];

// Following joint positions are ordered as follows:
// shoulder azimuth, shoulder pitch, elbow, wrist pitch, wrist yaw,
// wrist roll

double Data[6];      // variable for data capture.

// Digital outputs for smart tool from comedi0

extern int toolOnIN; // tool control variables from read_writeIO()
extern int toolDirIN;

extern int toolOn;    // toolOn = 1 is on; use as either on/off or PWM.
extern int toolDir;   // toolDir = 0 is forward as default; reverse is 1.

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
//    ts.tv_nsec = 31250000; // set to 32 hz
    ts.tv_nsec = 24400000; // calibrated for actual runtime 32 hz

// time-stamping variables

    time_t time(time_t *tp);

    time_t now;

```

```

// file for data capture

FILE *fp;

if ((fp = fopen("retract_data", "wb"))==NULL)
{
    printf("Cannot open file.\n");
    exit(1);
}

// Setup shared memory

child2( );

// Variables

int i = 0;
int j = 0;

double inc = .015625; // .5 in/sec @ 32 hz

// Set constraints and scaling. Note that positions use 1; orientations use 0.

for (i=0;i<6;i++) //initialize memory
{
    parmRW->armCtrl.axesConstr[i]=1.0;
    parmRW->armCtrl.axesScal[i]=1.0;
    parmRW->armCtrl.armMode=IDLE;
    parmRW->armCtrl.cartesCtrl[i]=parmR->armRightCar[i];
    if(i>2)
    {
        parmRW->armCtrl.cartesCtrl[i]=0.0;
    }
}

// Read the starting Cartesian position (where you are now) from
// shared memory.

for (i = 0; i < 6; i++)
{
    qZero[i] = parmR->armRightCar[i];
}

// Read the starting joint angles (where you are now) from shared memory.
// This is for end-effector orientation calculations.

for (i = 0; i < 6; i++)
{
    qJoints[i] = parmR->armRight[i];
}

// timestamp

now = time(NULL);

fprintf(fp, "\n%s\n", ctime(&now));

// Trajectory begins here.///////////////////////////////////////////////////

```

```

    for (j = 0; j < 256; j++)          // 256 points = 32hz X 8 seconds
                                        // move enough to clear task
    {

// read current Titan position and write to data file
        for (i = 0; i < 6; i++)
        {
            Data[i] = parmR->armRightCar[i];
        }

fprintf(fp, "\n %d %d %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f\n", j, toolOnIN,
        Data[0], Data[1], Data[2], Data[3], Data[4], Data[5]);

// Calculate incremental positions once through each loop.
qNow[0] = qZero[0] - j * inc * cos(qJoints[0] + qJoints[4]); // X
qNow[1] = qZero[1] - j * inc * sin(qJoints[0] + qJoints[4]); // Y
// Don't move the wrist joints or Z motion.

qNow[2] = qZero[2];                // Z stays the same
qNow[3] = qZero[3];                // rX stays the same
qNow[4] = qZero[4];                // rY stays the same
qNow[5] = qZero[5];                // rZ stays the same

// Write joint positions back to shared memory.
for (i = 0; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

// Calculate delta position once through each loop (for wrist).
for (i = 0; i < 6; i++)
{
    qNowV[i] = qNow[i] - qNowOld[i];
}

// Write joint positions back to shared memory.
// Position uses qNow; orientation uses qNowV.
// 0, 1, 2 are qNow for positions, 3, 4, 5 are qNowV for velocities.

for (i = 0; i < 3; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

for (i = 3; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = 0; //qNowV[i];
}

```

```

    // Flag and write to Cartesian

    grabSem(0,&sb,semid);
    parmRW->armCtrl.updFlag=1;
    parmRW->armCtrl.armMode=CART;

    // Xfer current new positions to old positions

    for (i = 0; i < 6; i++)
    {
        qNowOld[i] = qNow[i];
    }

    // Return semaphore

    retSem(0,&sb, semid);

    // Delay to control loop rate

    nanosleep(&ts, NULL);

    // Loop ////////////////////////////////////////
}

// timestamp

now = time(NULL);

fprintf(fp, "\n%s\n",ctime(&now));

// exit mode...clean up and get out

grabSem(0,&sb,semid);
parmRW->armCtrl.armMode=IDLE;
retSem(0,&sb, semid);
}

```

```

/*****
*
*   bRetractS.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

#include "comediFT.h"

int read_writeIO(void); // reads comedi0 analog/digital IO

int bRetractS(void)
{

int QUIT = 0;

    int shmidR,shmidRW, semid; // IPC identifiers
    key_t key_memRW,key_memR, key_sem; // keys for shared mem and semphores.
    struct sembuf sb; // semaphore control structure

    /***/

void safe_quit(void)
{
    QUIT=1;
}

/***/

/***/

int grabSem(int semNum,  struct sembuf *sb, int semid)
//semNum should be zero for this program.
{
    sb->sem_op=-1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
        // make sure you're using the semaphore when it is necessary.
        {
            perror("semaphore access problem");
            QUIT=1;
        }
    return 1;
}

/***/

```

```

int retSem(int semNum, struct sembuf *sb, int semid)
//semNum should be zero for this program so far.
{
    sb->sem_op=1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    {
        perror("semaphore return problem ");
        QUIT=1;
    }
    return 1;
}

//*****

// Calculate for each DOF; numbers in inches, used in trajectory calcs.

extern double qZero[6];    // Start point of robotic move
                          // (where you are now)

extern double qNow[6];     // Current calculated point in
                          // robotic trajectory

double qNowV[6];           // Incremental velocity for wrist
                          // orientations

double qNowOld[6];         // Used for incremental velocity calcs

// Stored instantaneous joint positions

extern double qJoints[7];

// Following joint positions are ordered as follows:
// shoulder azimuth, shoulder pitch, elbow, wrist pitch, wrist yaw,
// wrist roll

    double Data[6];        // current manipulator position

// Digital outputs for smart tool from comedi0

    extern int toolOnIN;    // tool control variables from read_writeIO()
    extern int toolDirIN;

    extern int toolOn;      // toolOn = 1 is on; use as either on/off or PWM.
    extern int toolDir;     // toolDir = 0 is forward as default; reverse is 1.

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
//    ts.tv_nsec = 31250000; // set to 32 hz
    ts.tv_nsec = 24400000; // calibrated for actual runtime 32 hz

// time-stamping variables

    time_t time(time_t *tp);

    time_t now;

```

```

// file for data capture

FILE *fp;

if ((fp = fopen("retract_data", "wb"))==NULL)
{
    printf("Cannot open file.\n");
    exit(1);
}

// Setup shared memory

child2( );

// Set up Force/Torque Sensor

char *calfilepath;    // name of calibration file
unsigned short index; // index of calibration in file (second parameter;
default = 1)
Calibration *cal;     // struct containing calibration information
short sts;            // return value from functions

// ATI F/T sensor variables—Note: Many for future use!

float SampleBias[7];  // measures preloads on sensor before starting task

float SampleReading[7]; // raw sensor values as read from comedi1

float SampleTT[6]={0,0,0,0,0,0}; //sensor axis transform
// Translate along/about {x translate, y translate, z translate, x rotate, y
rotate, z rotate}

float FT[6];          // array to hold the resultant force/torque vector.

// comedi1 variables

int subdev = 0;        // analog port (comedi1 not used for anything other than
F/T sensor)
int range = 0;         // 0 = +/10VDC
int aref = AREF_DIFF; // Differential Input

int n_chans0;
int maxdata0;
comedi_t *device0;
int chan=0;
lsampl_t data0;

device0 = comedi_open("/dev/comedi1");

n_chans0 = comedi_get_n_channels(device0, subdev);

for(chan = 0; chan < n_chans0; ++chan)
{
    maxdata0 = comedi_get_maxdata(device0, subdev, chan);
    comedi_data_read(device0, subdev, chan, range, aref, &data0);
    SampleReading[chan] = comedi_to_phys(data0, comedi_get_range(device0, subdev,
        chan, range), maxdata0);
}

```

```

// Set up ATI functions

calfilepath="FT5240.cal";
index = 1;

// create Calibration

cal=createCalibration(calfilepath,index);
if (cal==NULL) {
    printf("\nSpecified calibration could not be loaded.\n");
    scanf(".");
    return 0;
}

// NOTE: BELOW FT SETUP KEPT IN EVENT OF FUTURE USE!

// Set force units.
// This step is optional; by default, the units are inherited from the
// calibration file.

sts=SetForceUnits(cal,"N");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid force units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set torque units.
// This step is optional; by default, the units are inherited from the
calibration file.
sts=SetTorqueUnits(cal,"N-m");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid torque units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set tool transform.
// This line is only required if you want to move or rotate the sensor's
// coordinate system.
// This example tool transform translates the coordinate system 20 mm along the
// Z-axis
// and rotates it 45 degrees about the X-axis.
sts=SetToolTransform(cal,SampleTT,"mm","degrees");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid distance units"); return 0;
    case 3: printf("Invalid angle units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Variables

int i = 0;
int j = 0;

double inc = .015625; // .5 in/sec @ 32 hz

```



```

// Set constraints and scaling. Note that positions use 1; orientations use 0.

for (i=0;i<6;i++) //initialize memory
{
    parmRW->armCtrl.axesConstr[i]=1.0;
    parmRW->armCtrl.axesScal[i]=1.0;
    parmRW->armCtrl.armMode=IDLE;
    parmRW->armCtrl.cartesCtrl[i]=parmR->armRightCar[i];
    if(i>2)
    {
        parmRW->armCtrl.cartesCtrl[i]=0.0;
    }
}

// Read the starting Cartesian position (where you are now) from
// shared memory.

for (i = 0; i < 6; i++)
{
    qZero[i] = parmR->armRightCar[i];
}

// Read the starting joint angles (where you are now) from shared memory.
// This is for end-effector orientation calculations.

for (i = 0; i < 6; i++)
{
    qJoints[i] = parmR->armRight[i];
}

// timestamp

now = time(NULL);

fprintf(fp, "\n%s\n", ctime(&now));

// Trajectory begins here.////////////////////////////////////////////////////

for (j = 0; j < 256; j++)          // 256 points = 32hz X 8 seconds
                                    // move enough to clear task

{

// read current Titan position and write to data file

for (i = 0; i < 6; i++)
{

    Data[i] = parmR->armRightCar[i];

}

fprintf(fp, "\n %d %d %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f
%9.6f %9.6f %9.6f\n", j, toolOnIN, FT[0], FT[1], FT[2], FT[3], FT[4],
FT[5], Data[0], Data[1], Data[2], Data[3], Data[4], Data[5]);

```

```

// Calculate incremental positions once through each loop.
qNow[0] = qZero[0] - j * inc * cos(qJoints[0] + qJoints[4]); // X
qNow[1] = qZero[1] - j * inc * sin(qJoints[0] + qJoints[4]); // Y
qNow[2] = qZero[2]; // Z, no motion necessary since the blade
// cleared the pipe during cutting.

// Don't move the wrist joints

qNow[2] = qZero[2]; // rX stays the same
qNow[3] = qZero[3]; // rX stays the same
qNow[4] = qZero[4]; // rY stays the same
qNow[5] = qZero[5]; // rZ stays the same

// Write joint positions back to shared memory.
for (i = 0; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

// Calculate delta position once through each loop (for wrist).
for (i = 0; i < 6; i++)
{
    qNowV[i] = qNow[i] - qNowOld[i];
}

// Write joint positions back to shared memory.
// Position uses qNow; orientation uses qNowV.
// 0, 1, 2 are qNow for positions, 3, 4, 5 are qNowV for velocities.
for (i = 0; i < 3; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

for (i = 3; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = 0; //qNowV[i];
}

// Flag and write to Cartesian
grabSem(0,&sb,semid);
parmRW->armCtrl.updFlag=1;
parmRW->armCtrl.armMode=CART;

// Xfer current new positions to old positions
for (i = 0; i < 6; i++)
{
    qNowOld[i] = qNow[i];
}

// Return semaphore
retSem(0,&sb, semid);

```

```

        // Delay to control loop rate
        nanosleep(&ts, NULL);

        // Loop //////////////////////////////////////
    }

    // timestamp
    now = time(NULL);
    fprintf(fp, "\n%s\n", ctime(&now));

    // exit mode...clean up and get out
    grabSem(0, &sb, semid);
    parmRW->armCtrl.armMode=IDLE;
    retSem(0, &sb, semid);
}

```

```

/*****
*
*   bWristR.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

int bWristR(void)
{

    int QUIT = 0;

    /*****/

    void safe_quit(void)
    {
        QUIT=1;
    }
    /*****/

    /*****/

    int grabSem(int semNum, struct sembuf *sb, int semid)
    //semNum should be zero for this program.
    {
        sb->sem_op=-1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)
            // make sure you're using the semaphore when it is necessary.
            {
                perror("semaphore access problem");
                QUIT=1;
            }
        return 1;
    }

    /*****/

    int retSem(int semNum, struct sembuf *sb, int semid)
    //semNum should be zero for this program.
    {
        sb->sem_op=1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)
            {
                perror("semaphore return problem ");
                QUIT=1;
            }
        return 1;
    }

```

```

}

//*****

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
//    ts.tv_nsec = 31250000; // set to 32 hz
    ts.tv_nsec = 24400000; // calibrated for actual runtime 32 hz

// Setup shared memory

    child2( );

// Variables

    int i = 0;
    int j = 0;

// global variables

extern double qZero[6];
extern double qFinal[6];
extern double qNow[6];

// Read the starting position (where you are now) from shared memory.

    for (i = 0; i < 6; i++)
    {
        qZero[i] = parmR->armRight[i];
    }

// Set the target position (where you want to go) per stored memory.

    for (i = 0; i < 6; i++)
    {
        qFinal[i] = qZero[i]; // no motion except in specified joint.
    }

    qFinal[5] = -1.604185; // level wrist roll

// Set joint control mode

    parmRW->armCtrl.armMode = 4; // mode = JOINT

// Trajectory begins here.////////////////////////////////////////////////////

    for (j = 0; j < 64; j++) // 64 points = 32hz X 2 seconds
    {

// Calculate incremental positions once through each loop.

//    Quintic Trajectory Equation

        for (i = 0; i < 6; i++)
        {
            // Quintic equation

            qNow[i] = qZero[i]

```

```

        + 25 * ((qFinal[i] - qZero[i]) / 65536.0) * pow(j, 3)\
        - 75 * ((qFinal[i] - qZero[i]) / 8388608.0) * pow(j, 4)\
        + 15 * ((qFinal[i] - qZero[i]) / 268435456.0) * pow(j, 5);
    }

    // Write joint positions back to shared memory.
    for (i = 0; i < 6; i++)
    {
        parmRW->armCtrl.jointCtrl[i] = qNow[i];
    }

    // Delay to control loop rate
    nanosleep(&ts, NULL);

    // Loop //////////////////////////////////////
    }

    // Set joint control mode
    parmRW->armCtrl.armMode = 0; // mode = IDLE
    return(0);
}

```

```

/*****
*
*   bCut128S.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*   128 Hz loop rate to examine saw freq data
*
*****/

/*****
*
*   Obligatory Acknowledgements for open source libraries
*
*   ATIDAQ F/T C Library
*   v1.0.1
*   Copyright (c) 2001 ATI Industrial Automation
*
*   The MIT License
*
*   Permission is hereby granted, free of charge, to any person obtaining a
*   copy of this software and associated documentation files (the
*   "Software"), to deal in the Software without restriction, including
*   without limitation the rights to use, copy, modify, merge, publish,
*   distribute, sublicense, and/or sell copies of the Software, and to
*   permit persons to whom the Software is furnished to do so, subject to
*   the following conditions:
*
*   The above copyright notice and this permission notice shall be included
*   in all copies or substantial portions of the Software.
*
*   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
*   OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
*   MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
*   IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
*   CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
*   TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
*   SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*
////////////////////////////////////
*
*   Comedilib
*   Copyright (c) 1999,2000 David A. Schlee <ds@schlee.org>
*
*   This file may be freely modified, distributed, and combined with
*   other software, as long as proper attribution is given in the
*   source code.
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

#include "comediFT.h"

```

```

int read_writeIO(void);           // reads comedi0 analog/digital IO

int bCut128S(void)
{

// System level communications

int QUIT = 0;

    int shmidR,shmidRW, semid; // IPC identifiers
    key_t key_memRW,key_memR, key_sem; // keys for shared mem and semphores.
    struct sembuf sb; // semaphore control structure

    /*******
void safe_quit(void)
{
    QUIT=1;
}
    /*******

    /*******
int grabSem(int semNum, struct sembuf *sb, int semid)
    //semNum should be zero for this program so far.
{
    sb->sem_op=-1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
        // make sure you're using the semaphore when it is necessary.
    {
        perror("semaphore access problem");
        QUIT=1;
    }
    return 1;
}

    /*******
int retSem(int semNum, struct sembuf *sb, int semid)
    //semNum should be zero for this program so far.
{
    sb->sem_op=1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    {
        perror("semaphore return problem ");
        QUIT=1;
    }
    return 1;
}

    /*******

// Calculate for each DOF; numbers in inches, used in trajectory calcs.

double qZero[6];    // Start point of robotic move
                    // (where you are now)

double qNow[6];     // Current calculated point in
                    // robotic trajectory

double qNowV[6];    // Incremental velocity for wrist
                    // orientations

```



```

double qNowOld[6]; // Used for incremental velocity calcs

// Stored instantaneous joint positions
double qJoints[7];

// Following joint positions are ordered as follows:
// shoulder azimuth, shoulder pitch, elbow, wrist pitch, wrist yaw,
// wrist roll

double Data[6]; // current manipulator position

// Recursive Filter variables

float ryFilt = 0;
float ryFiltOld = 0;

// Data Analysis Variables

float ryFiltAbs = 0;

// for sampleBias switching initializing the F/T
static int biasFlag = 1;

// Signature Analysis Variables

int CONTACT1 = 0;
int senseContact = 0;

// Force control variables

float setpoint = 10.0;
float error = 0.0;
float gain = .02;
float controlF = 0.0;
float control = 0.0;
float controlFFilt = 0.0;
float controlFFiltOld = 0.0;

// Digital outputs for smart tool from comedi0

extern int toolOnIN; // tool control variables from read_writeIO()
extern int toolDirIN;

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
    ts.tv_nsec = 2405555; // calibrated runtime 128 hz

// time-stamping variables

    time_t time(time_t *tp);

    time_t now;

// file for data capture

```

```

FILE *fp;

if ((fp = fopen("cut_data128", "wb"))==NULL)
{
    printf("Cannot open file.\n");
    exit(1);
}

// Setup shared memory

child2( );

// Variables

int i = 0;
int j = 0;
int k = 128;

double inc = .015625; // .5 in/sec @ 32 hz
float contactThreshold = 500.00; // set to avoid tripping

// Set constraints and scaling.
// Note that positions use 1; orientations use 0.

for (i=0;i<6;i++) //initialize memory
{
    parmRW->armCtrl.axesConstr[i]=1.0;
    parmRW->armCtrl.axesScal[i]=1.0;
    parmRW->armCtrl.armMode=IDLE;
    parmRW->armCtrl.cartesCtrl[i]=parmR->armRightCar[i];
    if(i>2)
    {
        parmRW->armCtrl.cartesCtrl[i]=0.0;
    }
}

// Read the starting Cartesian position (where you are now) from
// shared memory.

for (i = 0; i < 6; i++)
{
    qZero[i] = parmR->armRightCar[i];
}

// Read the starting joint angles (where you are now) from shared memory.
// This is for end-effector orientation calculations.

for (i = 0; i < 6; i++)
{
    qJoints[i] = parmR->armRight[i];
}

// timestamp

now = time(NULL);

fprintf(fp, "\n%s\n", ctime(&now));

```

```

// Set up Force/Torque Sensor—NOTE: much of this not used in
// current iteration

char *calfilepath;    // name of calibration file
unsigned short index; // index of calibration in file
                    // (second parameter; default = 1)
Calibration *cal;     // struct containing calibration information
short sts;            // return value from functions

// ATI F/T sensor variables

float SampleBias[7]; // measures preloads on sensor before
                    // starting task

float SampleReading[7]; // raw sensor values as read from comed1

float SampleTT[6]={0,0,0,0,0,0}; //sensor axis transform

float FT[6];          // array to hold the resultant force/torque
                    // vector.

// comed1 variables

int subdev = 0;        // analog port (comed1 not used for anything
                    // other than F/T sensor)
int range = 0;         // 0 = +/-10VDC
int aref = AREF_DIFF;  // Differential Input

int n_chans0;
int maxdata0;
comed1_t *device0;
int chan=0;
lsampl_t data0;

device0 = comed1_open("/dev/comed1");

n_chans0 = comed1_get_n_channels(device0, subdev);

for(chan = 0; chan < n_chans0; ++chan)
{
    maxdata0 = comed1_get_maxdata(device0, subdev, chan);
    comed1_data_read(device0, subdev, chan, range, aref, &data0);
    SampleReading[chan] = comed1_to_phys(data0, comed1_get_range(device0, subdev,
chan, range), maxdata0);
}

// Set up ATI functions

calfilepath="FT5240.cal";
index = 1;

cal=createCalibration(calfilepath,index);
if (cal==NULL) {
    printf("\nSpecified calibration could not be loaded.\n");
    scanf(".");
    return 0;
}

// Set force units.
// This step is optional; by default, the units are inherited from the
// calibration file.

```

```

sts=SetForceUnits(cal,"N");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid force units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set torque units.
// This step is optional; by default, the units are inherited from the
// calibration file.
sts=SetTorqueUnits(cal,"N-m");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid torque units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set tool transform.
// This line is only required if you want to move or rotate the
// sensor's coordinate system.

sts=SetToolTransform(cal,SampleTT,"mm","degrees");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid distance units"); return 0;
    case 3: printf("Invalid angle units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Trajectory begins here.////////////////////////////////////////////////////

    for (j = 0; j < 12000; j++) // Governs increments and times out if
                                // thresholds go wrong.

    {

// Check forces/torques for contact; terminate if contact above threshold

        for(chan = 0; chan < n_chans0; ++chan)
        {

            maxdata0 = comedi_get_maxdata(device0, subdev, chan);

            comedi_data_read_delayed(device0, subdev, chan, range, aref,    &data0,
                                    10000);

            SampleReading[chan] = comedi_to_phys(data0, comedi_get_range(device0,
                                    subdev, chan, range), maxdata0);

        }

// Bias the sensor once only.

if(biasFlag==1)
{
    for (i = 0; i < 6; i++)

    {

```

```

        SampleBias[i] = SampleReading[i];
    }

    Bias(cal, SampleBias);

    biasFlag = 0;
}

// convert a loaded measurement into forces and torques
    ConvertToFT(cal, SampleReading, FT);

// Recursive filter on ry axis, saw blade torque, for 128hz
    ryFilt = (1.0/128.0) * FT[4] + (127.0/128.0) * ryFiltOld;
    ryFiltOld = ryFilt;
    ryFiltAbs = fabs(ryFilt);

// Turn Saw ON after initializing the FT
    toolOnIN = 0;
    toolDirIN = 0; //0 = unbolt, 1 = bolt

    read_writeIO();

// Read current joint angles from shared memory.
    for (i = 0; i < 6; i++)
    {
        qJoints[i] = parmR->armRight[i];
    }

// read current Titan position and write to data file
    for (i = 0; i < 6; i++)
    {
        Data[i] = parmR->armRightCar[i];
    }

// Force-based Trajectory Control
    error = setpoint - ryFilt;
    controlF = gain * error;
    control = inc/32.0 + controlF;

// read current Titan position and write to data file
    for (i = 0; i < 6; i++)
    {
        Data[i] = parmR->armRightCar[i];
    }

```

```

}

fprintf(fp, "\n %d %d %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f
          %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f\n", j, toolOnIN, FT[0], FT[1],
          FT[2], FT[3], FT[4], FT[5], Data[0], Data[1], Data[2], Data[3], Data[4],
          Data[5], ryFilt, ryFiltAbs, controlF, control);

// Calculate incremental positions once through each loop.

// Only motion in -Z

qNow[0] = qZero[0];           // X stays the same
qNow[1] = qZero[1];           // Y stays the same

qNow[2] = qZero[2] - j * inc/32.0 - controlF;   // Z motion, P + F

// Fixed orientation

qNow[3] = qZero[3];           // rX stays the same
qNow[4] = qZero[4];           // rY stays the same
qNow[5] = qZero[5];           // rZ stays the same

// Write joint positions back to shared memory.
for (i = 0; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

// Calculate delta position once through each loop (for wrist).
for (i = 0; i < 6; i++)
{
    qNowV[i] = qNow[i] - qNowOld[i];
}

// Write joint positions back to shared memory.
// Position uses qNow; orientation uses qNowV.
// 0, 1, 2 are qNow for positions, 3, 4, 5 are qNowV for velocities.
for (i = 0; i < 3; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = qNow[i];
}

for (i = 3; i < 6; i++)
{
    parmRW->armCtrl.cartesCtrl[i] = 0; //qNowV[i];
}

// Flag and write to Cartesian

grabSem(0,&sb,semid);
parmRW->armCtrl.updFlag=1;
parmRW->armCtrl.armMode=CART;

```

```

// Xfer current new positions to old positions
for (i = 0; i < 6; i++)
{
    qNowOld[i] = qNow[i];
}

// Return semaphore
retSem(0,&sb, semid);

// Delay to control loop rate
nanosleep(&ts, NULL);

// Logic rules to control cutting
// Detect pipe contact.
if(ryFiltAbs > 1.0 && senseContact == 0)
{
    senseContact = 1;
    printf("\nj= %d, pipe contact \n", j);
}

// Announce cut threshold reached.
if(ryFiltAbs > 10.0 && CONTACT1 == 0)
{
    CONTACT1 = 1;
    printf("\nj= %d, cut threshold reached\n", j);
}

// If fyFiltAbs goes high after going low, reset k to max.
// Account for common condition on main pipe section.
if(ryFiltAbs > 10.0)
{
    k = 128;
}

// If threshold reached and k not 0, start count down.
if(ryFiltAbs < 1.0 && CONTACT1 == 1 && k > 0)
{
    k = k - 1;
}

```

```

// Quit loop if cut is done.
if(ryFiltAbs < 1.0 && k==0)
{
    toolOnIN = 1;
    toolDirIN = 1;

    read_writeIO();

    printf("\nj= %d, cut complete\n", j);

    j = 12000;
}
}

// timestamp
now = time(NULL);
fprintf(fp, "\n%s\n", ctime(&now));

// make sure saw is off in case of any errors

    toolOnIN = 1;
    toolDirIN = 1;

    read_writeIO();

// exit mode...clean up and get out
grabSem(0,&sb,semid);
parmRW->armCtrl.armMode=IDLE;

retSem(0,&sb, semid);

// free memory allocated to Calibration structure
destroyCalibration(cal);

comedi_close(device0);

return 0;
}

```



```

/*****
*
*   bUnboltB.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>
#include <stdlib.h>

#include "comediFT.h"

static int biasFlag = 1;    // for sampleBias switching initializing F/T

int read_writeIO(void);    // reads comedi0 analog/digital IO

int bUnboltB(void)
{
    int QUIT = 0;

    int shmidR,shmidRW, semid; // IPC identifiers
    key_t key_memRW,key_memR, key_sem;    // keys for shared mem
                                           // and semaphores.
    struct sembuf sb; // semaphore control structure

/*****
void safe_quit(void)
{
    QUIT=1;
}
*****/

/*****
int grabSem(int semNum, struct sembuf *sb, int semid)
//semNum should be zero for this program so far.
{
    sb->sem_op=-1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    // make sure you're using the semaphore when it is necessary.
    {
        perror("semaphore access problem");
        QUIT=1;
    }
    return 1;
}
*****/

```

```

//*****
int retSem(int semNum, struct sembuf *sb, int semid)
//semNum should be zero for this program so far.
{
    sb->sem_op=1;
    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    {
        perror("semaphore return problem ");
        QUIT=1;
    }
    return 1;
}
//*****

// Variables

int i      = 0;
int j      = 0;

int test   = 1;
int set    = 0;

float fxstart = 0;
float fxstop  = 0;

// Recursive Filter variables

float fxFilt    = 0;
float fxFiltOld = 0;

double contactThreshold = -1000.00; // bypass contactThreshold

double Data[6]; // current manipulator position

// Position increment instead of time but run at sample time.

int senseContact = 0;

// Digital outputs for smart tool from comedi0

extern int toolOnIN; // tool control variables from read_writeIO()
extern int toolDirIN;

extern int toolOn; // toolOn = 1 is on; use as either on/off or PWM.
extern int toolDir; // toolDir = 0 is forward as default; reverse is 1.

// Loop timing management using nanosleep( )

struct timespec ts;
ts.tv_sec = 0;
ts.tv_nsec = 2405555; // calibrated for 128 hz for FFT look

// time-stamping variables

time_t time(time_t *tp);

time_t now;

```

```

// file for data capture

FILE *fp;

if ((fp = fopen("unbolt_data", "wb"))==NULL)
{
    printf("Cannot open file.\n");
    exit(1);
}

// timestamp

now = time(NULL);

fprintf(fp, "\n%s\n", ctime(&now));

// Setup shared memory

child2( );

// Set up Force/Torque Sensor

char *calfilepath;    // name of calibration file
unsigned short index; // index of calibration in file
                    // (second parameter; default = 1)
Calibration *cal;     // struct containing calibration information
short sts;            // return value from functions

// ATI F/T sensor variables—Note: Many for future use!

float SampleBias[7];  // measures preloads on sensor before starting task

float SampleReading[7]; // raw sensor values as read from comedil

float SampleTT[6]={0,0,0,0,0,0};    //sensor axis transform

float FT[6]={0,0,0,0,0,0};          // array to hold the resultant
                                   // force/torque vector.

// comedil variables

int subdev = 0;                // analog port (comedil not used for anything
                              // other than F/T sensor)
int range = 0;                // 0 = +10VDC
int aref = AREF_DIFF;         // Differential Input

int n_chans0;
int maxdata0;
comedi_t *device0;
int chan=0;
lsampl_t data0;

device0 = comedi_open("/dev/comedi1");

n_chans0 = comedi_get_n_channels(device0, subdev);

for(chan = 0; chan < n_chans0; ++chan)
{
    maxdata0 = comedi_get_maxdata(device0, subdev, chan);
}

```

```

    comedi_data_read(device0, subdev, chan, range, aref, &data0);
    SampleReading[chan] = comedi_to_phys(data0, comedi_get_range(device0, subdev,
        chan, range), maxdata0);
}

// Set up ATI functions

calfilepath="FT5240.cal";
index = 1;

// create Calibration

cal=createCalibration(calfilepath,index);
if (cal==NULL) {
    printf("\nSpecified calibration could not be loaded.\n");
    scanf(".");
    return 0;
}

// NOTE: BELOW FT SETUP KEPT IN EVENT OF FUTURE USE!

// Set force units.
// This step is optional; by default, the units are inherited
// from the calibration file.

sts=SetForceUnits(cal,"N");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid force units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set torque units.
// This step is optional; by default, the units are inherited from the
// calibration file.
sts=SetTorqueUnits(cal,"N-m");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid torque units"); return 0;
    default: printf("Unknown error"); return 0;
}

// Set tool transform.
// This line is only required if you want to move or rotate the sensor's
// coordinate system.
sts=SetToolTransform(cal,SampleTT,"mm","degrees");
switch (sts) {
    case 0: break; // successful completion
    case 1: printf("Invalid Calibration struct"); return 0;
    case 2: printf("Invalid distance units"); return 0;
    case 3: printf("Invalid angle units"); return 0;
    default: printf("Unknown error"); return 0;
}

```

```

// LOOP begins here //////////////////////////////////////
while(test==1)
{
    for (j = 0; j < 256; j++) // 128hz X 2 seconds
    {
// Check forces/torques
        for(chan = 0; chan < n_chans0; ++chan)
        {
            maxdata0 = comedi_get_maxdata(device0, subdev, chan);
            comedi_data_read_delayed(device0, subdev, chan, range, aref, &data0, 10000);
            SampleReading[chan] = comedi_to_phys(data0, comedi_get_range(device0, subdev,
                chan, range), maxdata0);
        }

// Bias the sensor once only.
        if(biasFlag==1)
        {
            for (i = 0; i < 6; i++)
            {
                SampleBias[i] = SampleReading[i];
            }

            Bias(cal, SampleBias);
            biasFlag = 0;
        }

// convert a loaded measurement into forces and torques
        ConvertToFT(cal, SampleReading, FT);

// Recursive filter on ry axis, saw blade torque, for 128hz
        fxFilt = (1.0/128.0) * FT[0] + (127.0/128.0) * fxFiltOld;
        fxFiltOld = fxFilt;

// Turn tool ON
        toolOnIN = 0;
        toolDirIN = 0; // 0 = unbolt, 1 = bolt
        read_writeIO();

// read current Titan position
        for (i = 0; i < 6; i++)

```

```

    {
        Data[i] = parmR->armRightCar[i];
    }

    fprintf(fp, "\n %d %d %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f %9.6f
        %9.6f %9.6f %9.6f %9.6f\n", j, toolOnIN, FT[0], FT[1], FT[2], FT[3],
        FT[4], FT[5], Data[0], Data[1], Data[2], Data[3], Data[4], Data[5],
        fxFilt);

// Delay to control loop rate
    nanosleep(&ts, NULL);

// Manage pushback variable

    if (j==1 && set==0)
    {
        fxstart = fxFilt;
        set = 1;
        printf("j= %d, fxstart = %f\n", j, fxstart);
    }

    if (j==255)
    {
        fxstop = fxFilt;
        printf("j= %d, fxstop = %f\n", j, fxstop);
    }

// Loop ////////////////////////////////////////
    }

// End test
    if(fabs(fxstop - fxstart) > 100.0)
    {
        test = 0;
        printf("unbolt done\n");
    }

}

```

```
    toolOnIN = 1;
    toolDirIN = 1;

    read_writeIO();

// timestamp
    now = time(NULL);
    fprintf(fp, "\n%s\n", ctime(&now));
    printf("return to operator\n");
    return 0;
}
```

```

/*****
*
*   functGoIdle.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*   Revision History
*
*   Date           Author           Description
*   -----
*   4/2010         Mark Noakes      function to switch to Idle mode.
*
*   -----
*
*****/

#include "newChild.h"

int functGoIdle(void)
{
    int QUIT = 0;

    /*******
    void safe_quit(void)
    {
        QUIT=1;
    }
    /*******

    /*******
    int grabSem(int semNum, struct sembuf *sb, int semid)
    //semNum should be zero for this program so far.
    {
        sb->sem_op=-1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)
        // make sure you're using the semaphore when it is necessary.
        {
            perror("semaphore access problem");
            QUIT=1;
        }
        return 1;
    }
    /*******
    int retSem(int semNum, struct sembuf *sb, int semid)
    //semNum should be zero for this program so far.
    {
        sb->sem_op=1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)
        {
            perror("semaphore return problem ");

```



```
        QUIT=1;
    }
    return 1;
}
//*****

// Setup shared memory

    child2( );

// Set joint control mode

    parmRW->armCtrl.armMode = 0; // mode = IDLE
return(0);
}
```

```

/*****
*
*   functMoveHome.c
*
*   DISSERTATION SOFTWARE
*
*   Behavior-based Telerobotic Tool Control
*   Mark W. Noakes
*   Dept of Mechanical, Aerospace, and Biomedical Engineering
*   University of Tennessee at Knoxville
*
*   Revision History
*
*   Date           Author           Description
*   -----
*   4/2010         Mark Noakes      function for joint level move to Home
*                                   position from any current location.
*
*   -----
*
*****/

#include "newChild.h"

#include <time.h>
#include <math.h>

int functMoveHome(void)
{
    int QUIT = 0;

    /*******
    void safe_quit(void)
    {
        QUIT=1;
    }
    /*******

    /*******
    int grabSem(int semNum,  struct sembuf *sb, int semid)
        //semNum should be zero for this program so far.
    {
        sb->sem_op=-1;
        sb->sem_num=semNum;
        if(semop(semid, sb,1)==-1)
            // make sure you're using the semaphore when it is necessary.
            {
                perror("semaphore access problem");
                QUIT=1;
            }
        return 1;
    }
    /*******
    int retSem(int semNum,  struct sembuf *sb, int semid)
        //semNum should be zero for this program so far.
    {
        sb->sem_op=1;

```

```

    sb->sem_num=semNum;
    if(semop(semid, sb,1)==-1)
    {
        perror("semaphore return problem ");
        QUIT=1;
    }
    return 1;
}
//*****

// Loop timing management using nanosleep( )

    struct timespec ts;
    ts.tv_sec = 0;
    ts.tv_nsec = 31250000; // set to 32 hz

// Setup shared memory

    child2( );

// Variables

    int i = 0;
    int j = 0;

// global variables

extern double qZero[6];
extern double qFinal[6];
extern double qHome[6];
extern double qNow[6];

// Read the starting position (where you are now) from shared memory.

    for (i = 0; i < 6; i++)
    {
        qZero[i] = parmR->armRight[i];
    }

// Set the target position (where you want to go) per stored memory.

    for (i = 0; i < 6; i++)
    {
        qFinal[i] = qHome[i];
    }

// Set joint control mode

    parmRW->armCtrl.armMode = 4; // mode = JOINT

// Trajectory begins here.////////////////////////////////////////////////////

    for (j = 0; j < 320; j++) // 32hz X 10 seconds
    {

```

```

// Calculate incremental positions once through each loop.
// Quintic Trajectory Equation
for (i = 0; i < 6; i++)
{
    // Quintic equation
    qNow[i] = qZero[i] + ((qFinal[i] - qZero[i]) / 3276800.0) *
        pow(j, 3) - 3 * ((qFinal[i] - qZero[i]) / 2097152000.0) *
        pow(j, 4) + 3 * ((qFinal[i] - qZero[i]) / 1677721600000.0) *
        pow(j, 5);
}

// Write joint positions back to shared memory.
for (i = 0; i < 6; i++)
{
    parmRW->armCtrl.jointCtrl[i] = qNow[i];
}

// Delay to control loop rate
nanosleep(&ts, NULL);

// LOOP ////////////////////////////////////////

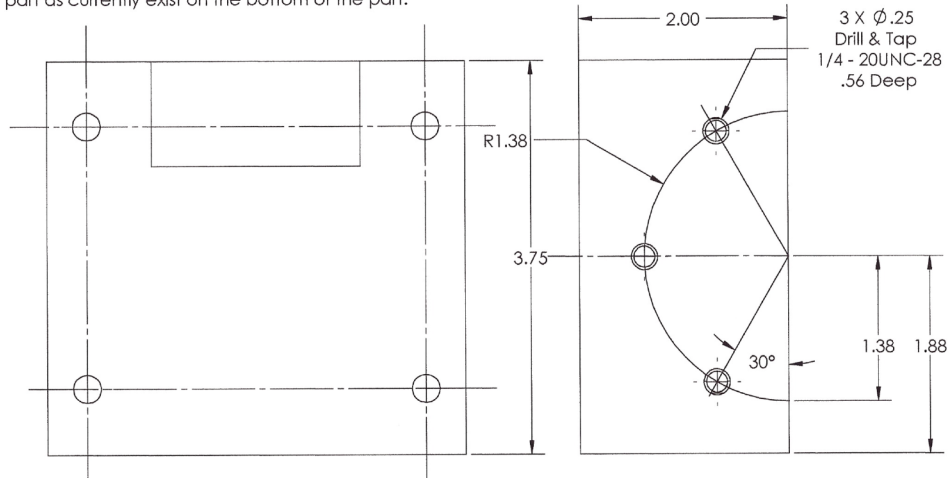
}

// Set joint control mode
parmRW->armCtrl.armMode = 0; // mode = IDLE
return(0);
}

```

Appendix B
Mechanical Drawings

NOTE: This is a modification of an existing part to place three additional tapped mounting holes on the side of the part as currently exist on the bottom of the part.

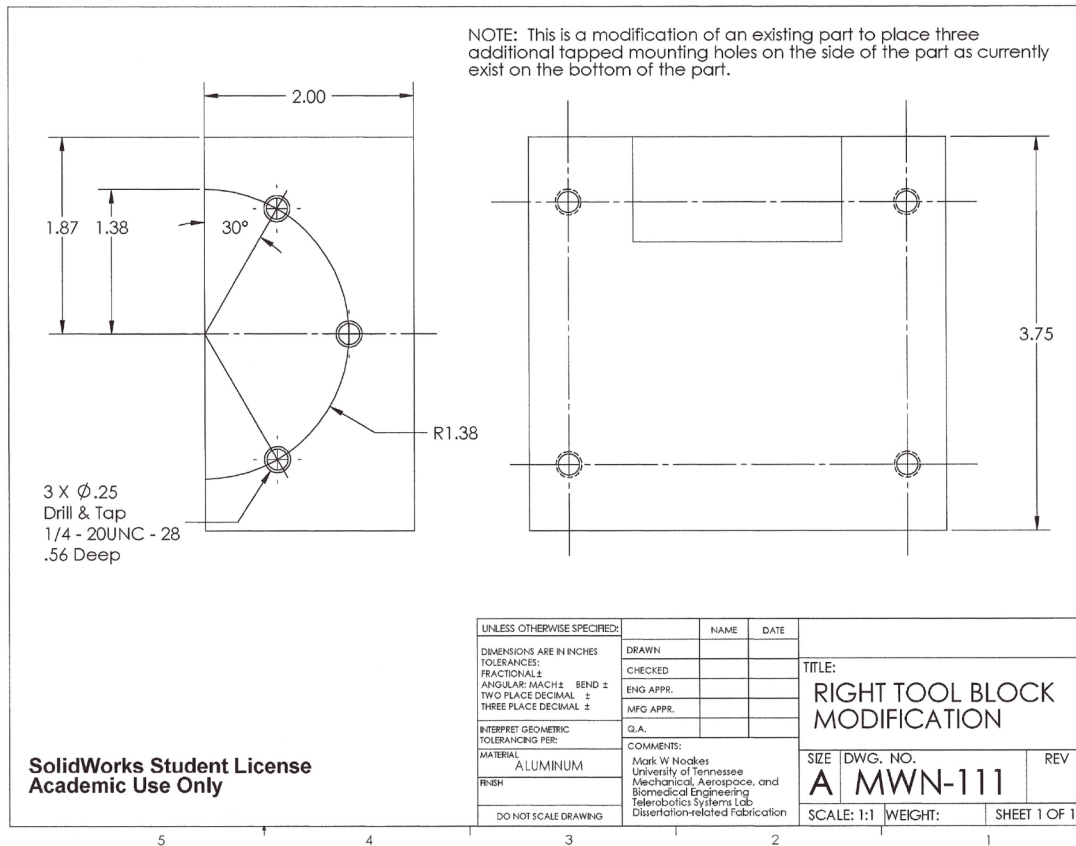


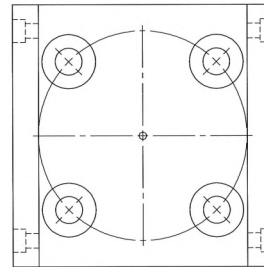
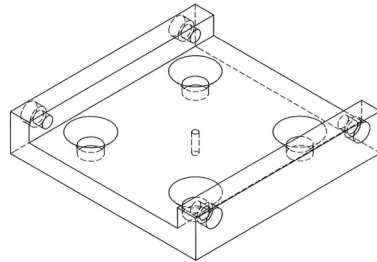
SolidWorks Student License
Academic Use Only

UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN INCHES		DRAWN	
TOLERANCES:		CHECKED	
FRACTIONAL \pm		ENG APPR.	
ANGULAR: MACH \pm BEND \pm		MFG APPR.	
TWO PLACE DECIMAL \pm		Q.A.	
THREE PLACE DECIMAL \pm		COMMENTS:	
INTERPRET GEOMETRIC TOLERANCING PER:		Mark W. Nuckles	
MATERIAL: ALUMINUM		University of Tennessee	
FINISH:		Mechanical, Aerospace, and	
DO NOT SCALE DRAWING		Biomedical Engineering	
		Telerobotics Systems Lab	
		Dissertation-related Fabrication Support	
		SIZE	DWG. NO.
		A	MWN-110
		SCALE: 1:1	WEIGHT:
		SHEET 1 OF 1	REV

TITLE:
LEFT TOOL BLOCK
MODIFICATION

SCALE: 1:1 WEIGHT: SHEET 1 OF 1





NOTE:
EXISTING PART. NO MODIFICATION
NECESSARY.

**SolidWorks Student License
Academic Use Only**

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
ALUMINUM

FINISH

DO NOT SCALE DRAWING

DRAWN

CHECKED

ENG APPR.

MFG APPR.

Q.A.

COMMENTS:

Mark W Nookes
University of Tennessee
Mechanical, Aerospace, and
Biomedical Engineering
Telerobotics Systems Lab
Dissertation-related Fabrication

NAME DATE

TITLE:
**FORCE/TORQUE
SENSOR FIXTURE PLATE**

SIZE DWG. NO. REV

A MWN-120

SCALE: 1:2 WEIGHT: SHEET 1 OF 1

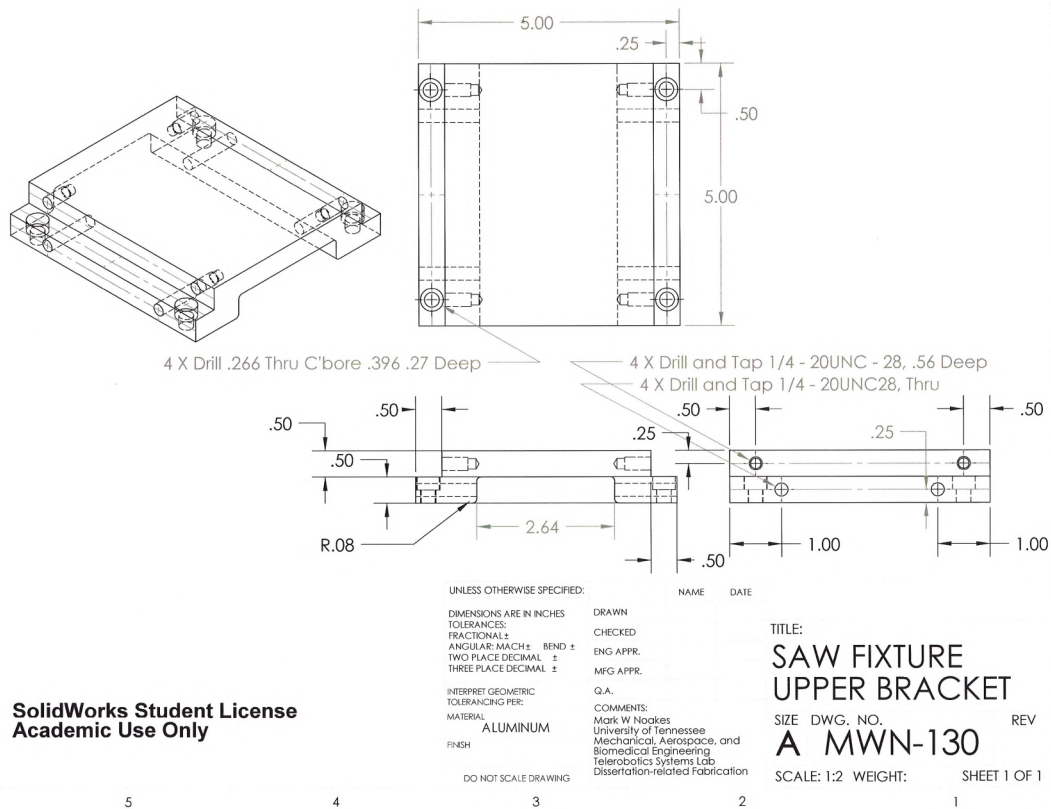
5

4

3

2

1



SolidWorks Student License
Academic Use Only

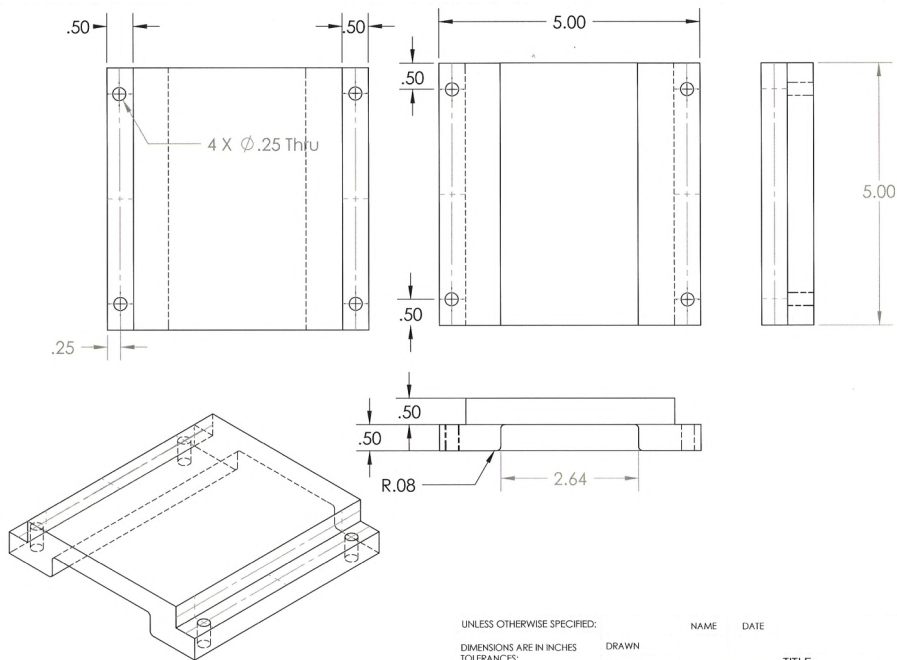
5

4

3

2

1



SolidWorks Student License
Academic Use Only

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES

TOLERANCES:

FRACTIONAL ±

ANGULAR: MACH ± BEND ±

TWO PLACE DECIMAL ±

THREE PLACE DECIMAL ±

INTERPRET GEOMETRIC

TOLERANCING PER:

MATERIAL ALUMINUM

FINISH

DO NOT SCALE DRAWING

DRAWN

CHECKED

ENG APPR.

MFG APPR.

Q.A.

COMMENTS:

Mark W Nookes
 University of Tennessee
 Mechanical, Aerospace, and
 Biomedical Engineering
 Telerobotics Systems Lab
 Dissertation-related Fabrication

NAME DATE

TITLE:

**SAW FIXTURE
 LOWER BRACKET**

SIZE DWG. NO. REV

A MWN-131

SCALE: 1:2 WEIGHT: SHEET 1 OF 1

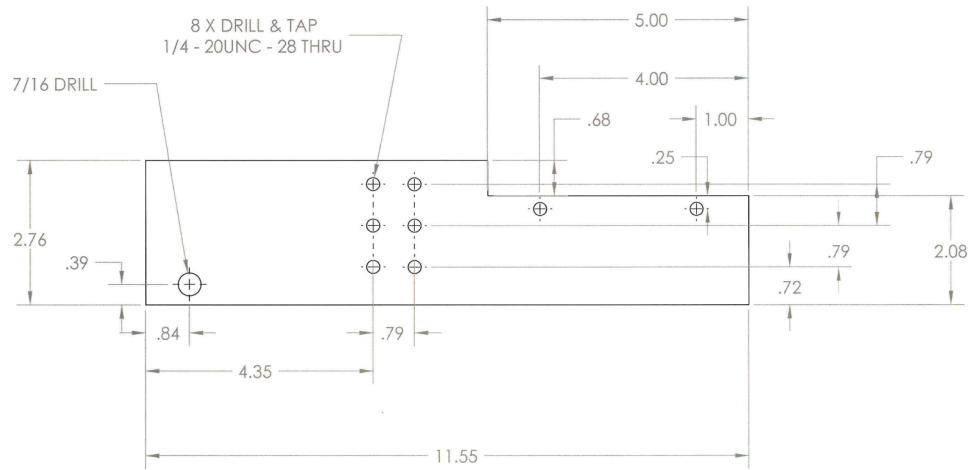
5

4

3

2

1



NOTE:

1. MATERIAL IS .25 INCHES THICK.
2. FABRICATE 2 PIECES.

**SolidWorks Student License
Academic Use Only**

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

INTERPRET GEOMETRIC
TOLERANCING PER:
MATERIAL
ALUMINUM
FINISH

DO NOT SCALE DRAWING

DRAWN
CHECKED
ENG APPR.
MFG APPR.

G.A.
COMMENTS:
Mark W Noakes

University of Tennessee
Mechanical, Aerospace, and
Biomedical Engineering
Tele robotic Systems Lab
Dissertation-related Fabrication

NAME DATE

TITLE:

**SAW FIXTURE
SIDE PLATE**

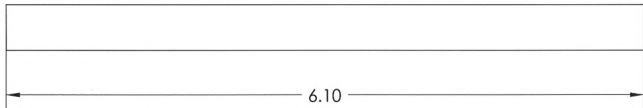
SIZE DWG. NO.

A MWN-132

REV

SCALE: 1:2 WEIGHT:

SHEET 1 OF 1



NOTE: CUT TO LENGTH AT FINAL ASSEMBLY.



7/16 - 14 THREADED ROD

SolidWorks Student License
Academic Use Only

UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN INCHES	DRAWN		
TOLERANCES:	CHECKED		
FRACTIONAL: ±	ENG APPR.		
ANGULAR: MACH ± BEND ±	MFG APPR.		
TWO PLACE DECIMAL ±			
THREE PLACE DECIMAL ±			
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL	COMMENTS:		
FINISH	Mark W Nookes		
	University of Tennessee		
	Mechanical Aerospace, and		
	Biomedical Engineering		
	Telerobotics Systems Lab		
	Dissertation-related Fabrication		
DO NOT SCALE DRAWING			

TITLE:
**POT THREADED
ROD**

SIZE	DWG. NO.	REV
A	MWN-133	
SCALE: 1:1	WEIGHT:	SHEET 1 OF 1

5

4

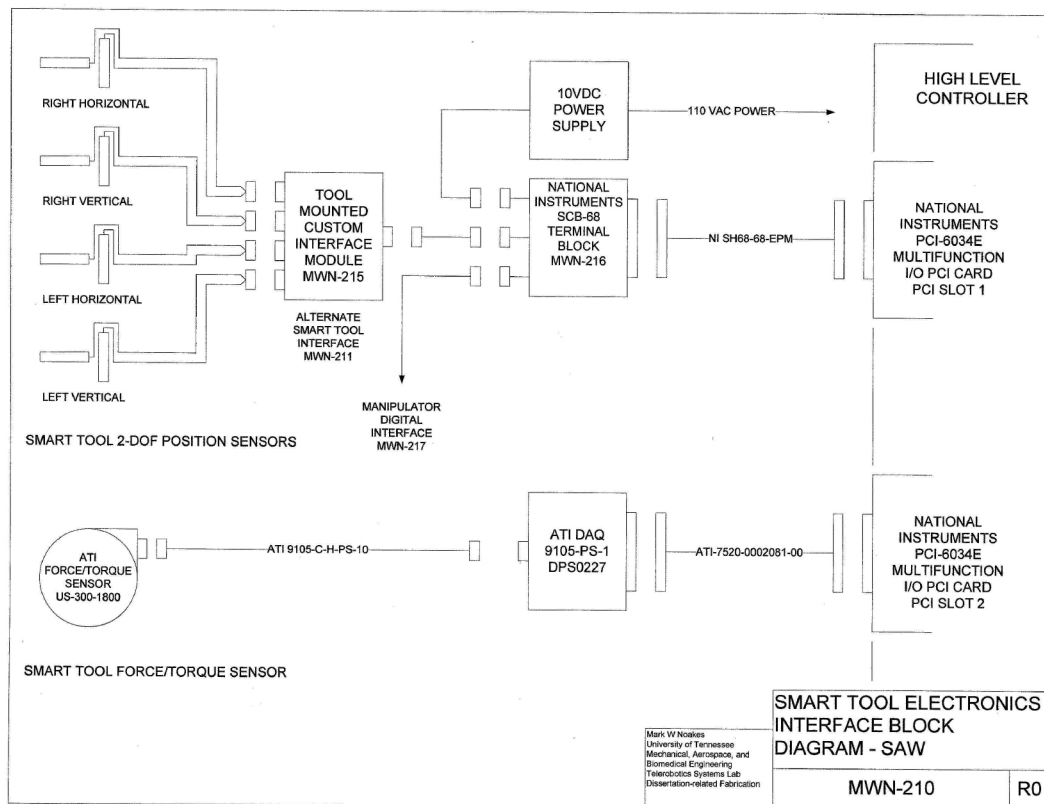
3

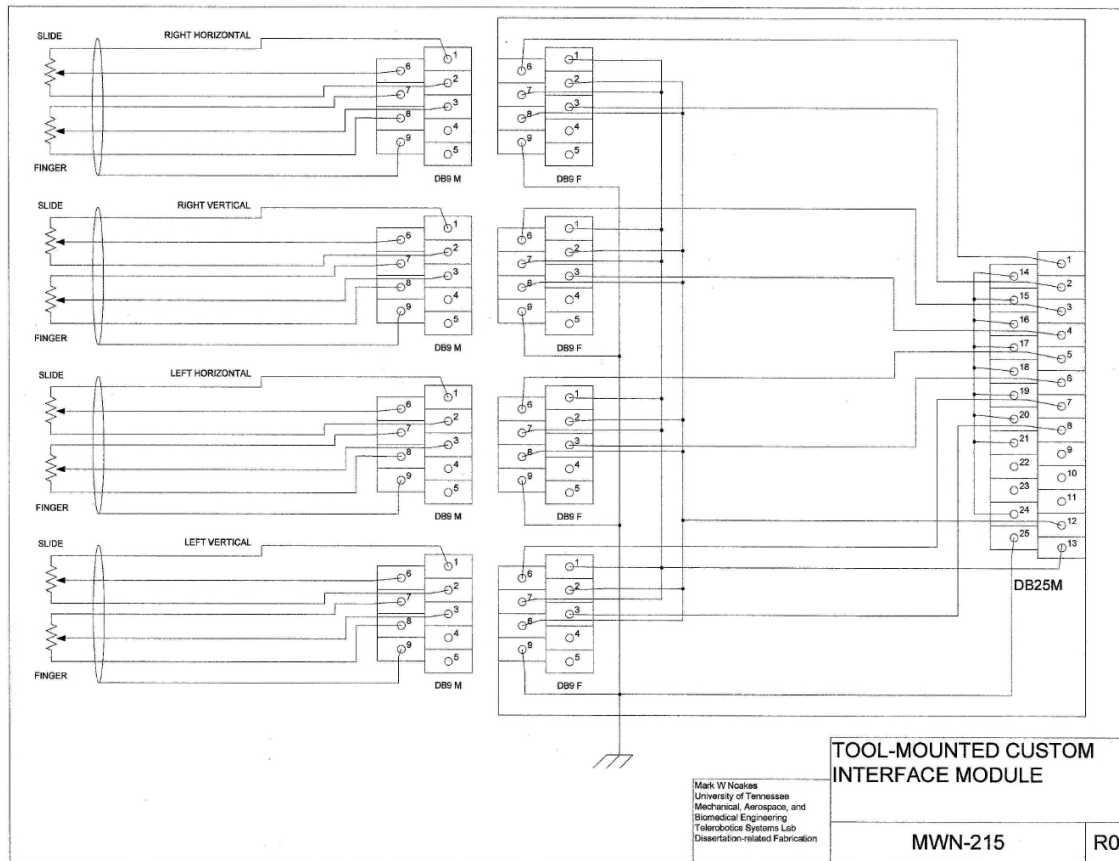
2

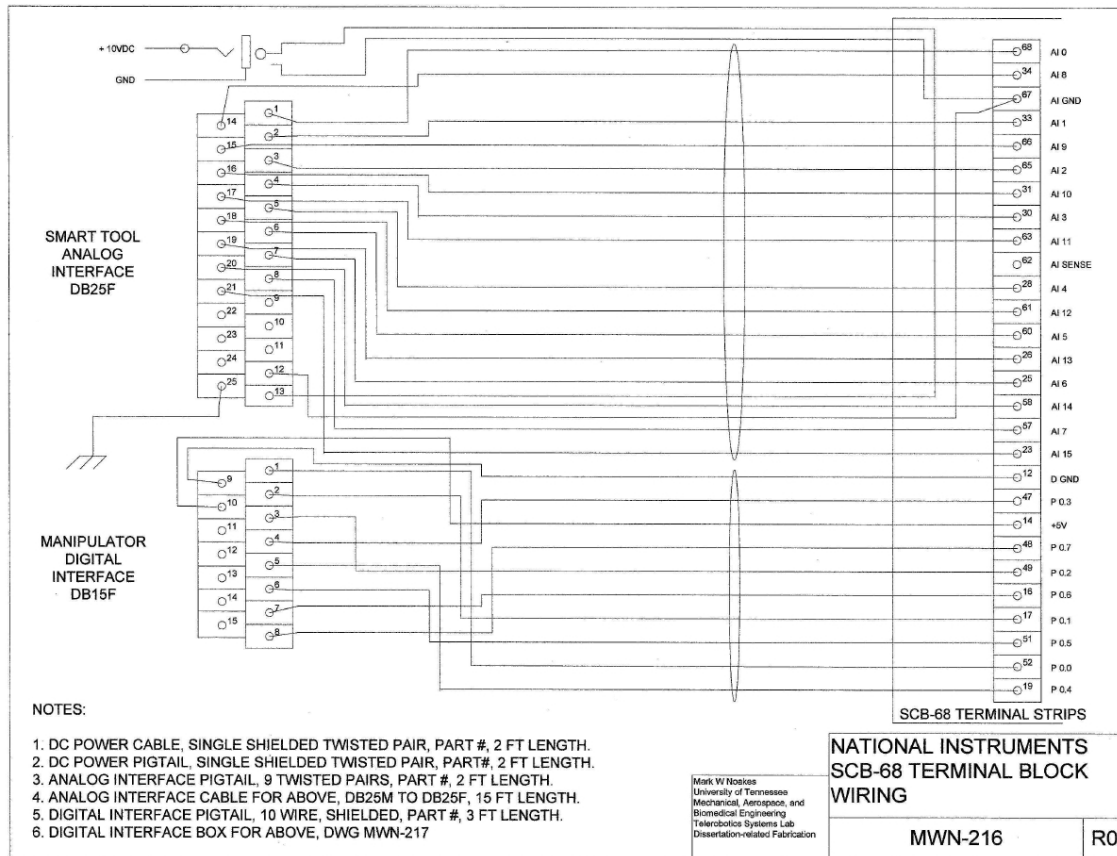
1

Appendix C

Schematics







Vita

Mark William Noakes was born in 1956 in upstate New York. He received his Bachelor's degree in Electrical Engineering from Tennessee Technological University (TTU) in August 1979. He received a Master of Science degree in Electrical Engineering from the University of Tennessee at Knoxville in December 1989. He received his professional engineering license in Electrical Engineering in 1993. His employment as an engineer began as a co-op student at NASA-Langley in Virginia in 1975 and 1976 working in the Instrument Research Division, Flight Instrumentation Division, and the Flight Dynamics and Controls Division. After graduating from TTU, he worked in industry for four years first designing sensors and hybrid microelectronics for first generation automotive computerized engine controls and later providing industrial and process controls, power systems plant engineering, and industrial robotics and automation support to a large chemical processing complex. He joined Oak Ridge National Laboratory in October 1983 and has worked as a research and development staff member since that time, specializing in development, deployment, operations, and maintenance of teleoperated and telerobotic systems, remote tooling, human-machine interfaces, facility and process design in support of remote operations, and the adaptation of those technologies to other areas of interest such as telesurgery.