ABSTRACT

Title of Thesis: HUMAN FACTORS EVALUATION OF OPERATOR INTERFACES FOR TELEOPERATION OF A DEXTEROUS MANIPULATOR

Kevin Davis, Master of Science, 2014

Thesis directed by: Associate Professor David L. Akin
Department of Aerospace Engineering

Ground teleoperation of a satellite servicing spacecraft is a challenging task for a human operator, especially when there is significant communications delay between the control station and spacecraft. On-orbit operations are further complicated by a communications time delay between the ground and spacecraft. Operator performance can be improved with the use of a graphical simulation of the robot. By displaying the robot's commanded position, graphical simulation can also mitigate some effects of time delay. This work implemented a visualization tool and commanded display to assist operation of a remote dexterous manipulator. A Fitts' Law experiment was designed to determine the effectiveness of the commanded display in reducing the impact of time delay. The experiment was conducted with a six degree of freedom manipulator over a range of time delays, from 0.0 to 6.0 seconds. The experimental results were analyzed to assess the reduction of task completion time and operator workload.

HUMAN FACTORS EVALUATION OF OPERATOR INTERFACES FOR TELEOPERATION OF A DEXTEROUS MANIPULATOR

by

Kevin Patrick Davis

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2014

Advisory Committee:
Associate Professor David L. Akin, Chair/Advisor
Associate Professor Raymond J. Sedwick
Dr. Craig R. Carignan

Acknowledgments

First and foremost, I want to thank my advisor Dr. David Akin for his support and guidance during my time at the Space Systems Laboratory. His lab has enabled me to participate in so many exciting projects over the years. I would also like to thank the other members of my committee. Thank you to Dr. Craig Carignan and Dr. Raymond Sedwick, for their time and support through this process. I would like to thank the Satellite Servicing Capabilities Office at NASA Goddard Space Flight Center for supporting this research.

I also want to recognize several people whose assistance has been helped make this research possible. Thanks to Nicholas D'Amore, who provided the support I needed to interface with the NBV-1 arm, and added several needed features to the robot's control software. Furthermore, his advice and experience was indispensable when troubleshooting some of the bugs within my own software. Thanks to Michael Schaffer, Erick Arce, and Daniel Goodley, for helping set up some of the test hardware. I also want to thank the rest of my lab mates at the Space Systems Laboratory for your support over the years, and for your advice during the thesis writing and preparation process. Finally, I want to acknowledge the test subjects who participated in this research for the hours of repetitious, monotonous, and uncompensated testing you performed. It is greatly appreciated.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

SSL	Space Systems Laboratory
UMd	University of Maryland
IRB	Institutional Review Board
NBV	Neutral Buoyancy Vehicle
RTSX	Ranger Telerobotic Shuttle Experiment
OSG	OpenSceneGraph
NASA	National Aeronautics and Space Administration
GSFC	Goddard Space Flight Center
SSCO	Satellite Servicing Capabilities Office
GUI	Graphical User Interface
DMU	Data Management Unit
TDU	Time Delay Unit
KDL	(Orocos) Kinematics and Dynamics Library

Chapter 1: Introduction

1.1 Motivation

Control of a telerobotic satellite servicing spacecraft is a complex and challenging task for a human operator. As with any telerobotic system, the interfaces between the operator and robot are critical elements. Their design determines how effectively an operator can command the robot. These interfaces can be improved with computer systems, which can be used to create interfaces that increase situational awareness, reduce operator workload, and improve overall task performance.

Advanced interfaces can be used to alleviate operator performance degradation caused by round trip time delay. Round trip time delay, in this context, refers to the time between when a command is sent to the robot to the time when the operator receives feedback. If an operator issues a command to move forward, and it takes one second for the command to reach the vehicle, and one second for the confirmation to reach the operator, then the total round trip is two seconds. This communication time delay is an inherent property of any communication network, and has a significant effect on the ability to remotely operate robotic systems. It may be on the order of microseconds, such as on a local area network, or minutes for a spacecraft millions of kilometers from Earth. For a spacecraft in low Earth orbit (LEO), the

minimum round trip time delay is 0.4 seconds due to the speed of the light. In practice, this time delay can be upwards of 6 seconds due to multiple bounces between communication satellites and other network equipment [21] [14]. Because LEO and higher Earth orbits are likely targets for robotic servicing spacecraft [3], round trip time delay has serious implications for ground-based teleoperation. In addition to spacecraft servicing, high time delay can be a concern with any remotely operated system, including telesurgical robots [13], remote underwater vehicles, and ground vehicles.

To mitigate this issue, this work seeks to implement a commanded display to represent a dexterous robotic arm's commanded position. By overlaying the commanded display on a display based on the robot's actual telemetry, the operator receives immediate visual feedback from commanded inputs. This allows the operator to control a simulated arm in real time, without having to wait for telemetry. After the duration of time delay has passed, the operator can verify that the actual, telemetry-based display converges to the commanded position. By eliminating the delay between input and visual feedback, this research aims to improve task performance and reduce operator workload during teleoperation tasks. In doing so, this work will be expanding upon previous work performed at the University of Maryland's (UMd) Space Systems Laboratory(SSL), particularly that of Lane, who demonstrated a commanded display on a purely virtual system [1] [2]. This work extends the commanded display to a physical robotic system and assesses its performance in a real-world manipulator positioning task.

1.2 Requirements

The primary goal of this work was to implement the commanded display on a teleoperation workstation. Implementing the display necessitated the development of a visualization software tool capable of working with existing robotic systems at the Space Systems Laboratory. To work with SSL systems, the system had to:

1. send joint pose commands to SSL DMU software
2. receive telemetry from SSL DMU software
3. display a graphical simulation of a serial-link manipulator with up to 8 degrees of freedom
4. update a the pose of a graphical arm model with real telemetry
5. overlay a graphical arm model of a manipulator's commanded position over the telemetry display
6. receive input from existing SSL hand controller clients

These abilities summarize the requirements that had to be met in order to implement the visualization tool and integrate with existing SSL systems. Considerations such as software reusability, modularity, and cross-platform capabilities also heavily influenced the development.

1.3 Thesis Organization

The following chapters will describe the specific aspects of this work. Chapter 2 provides background information relevant to this research. It begins with a discussion of time delay and previous work in time delay mitigation. The last part of the chapter describes several existing systems at the SSL that were used in the experimental setup, as well as the Open Source software libraries that were leveraged by the software design. Chapter 3 presents the software design and implementation of the commanded display. Chapter 4 describes the experiment, experimental setup, and results. Chapter 5 is the final chapter. It includes a summary of this work's findings and suggestions for future work.

Chapter 2: Background

This chapter begins with a discussion of time delay and its impact on task performance. It then discusses previous work done with graphical displays to ameliorate time delay's negative effects. The chapter concludes with an overview of existing resources used in this research, including robotic hardware used in this work and the open source software solutions leveraged for the visualization software design.

2.1 Time Delay

Human task performance generally suffers due to time delay. Research has shown how time delay impacts the realtime teleoperation of remote robotic systems [21] [27], including six or higher degree of freedom (DOF) dexterous manipulators [31] [1] [13] [14]. Significant round trip time delay inhibits an operator's ability to control robotic vehicles, reducing overall performance and increasing the operator's workload. Held found that delays as low as 0.3 seconds can cause operators to have difficulty adapting, causing them to decorrelate their movement from the system's response [28].

At higher time delays, it has been found that subjects lose the ability to

adapt completely and transition to a "move and wait" strategy [27]. Instead of continuously controlling the robot, the operator will input their commands, then wait to receive feedback before sending new commands. The transition to this strategy occurs at around one second of time delay. This strategy allows the operator to trade positional accuracy with completion time. In other words, they may spend more time waiting and verifying their motions in order to get a precise placement, or they may command broad motions before stopping to observe the robot's new position. In any case, the time spent waiting to verify and adjust the robot's position can drastically increase task completion time.

In many cases, time delay varies over time. On Internet protocol (IP) networks, data packets may be routed differently, causing commands to take different amounts of time to arrive at their destination. On-orbit, the communication path may change depending on which communication satellites and ground stations are in view from the spacecraft. This may result in periodic spikes in latency or high variability in the delay. Short variable delays can cause greater impact than longer fixed delays, due to the inability for subjects to predict and adapt to the delay period.

Time delay has a significant effect, regardless of task complexity. Thompson [25] showed the effects of time delay and a task's degree of constraint in degrading performance. He defined the degree of constraint as a restriction along an axis of motion for positioning an object. For example, placing a robot's tool-tip against a flat plate has no degrees of constraint. The robot may move freely in any direction in its path to contact the plate. In contrast, insertion of a square peg into a square hole would have five degrees of constraint. The tool tip may not rotate in any direction

without colliding into the hole's wall, and it may not translate against walls. The only remaining degree of freedom available is insertion into the hole. Increasing the degree of constraint linearly increased a task's completion time. This relationship was shown for several time delays, and the effects of time delay were additive to the effects of increased degree of constraint.

In this work, time delay was simulated using the netem network emulation tool. Netem was used to introduce a bi-directional time delay between a robot's local area network and the control station's local area network. This differs from several of the works discussed here, which only simulate time delay on commands transmitted from the operator to the robot [1]; they do not model the communication delay for the telemetry from the robot. Instead, they add the full round trip time delay to the transmission of the command, sometimes by simply holding commands in a buffer. In this way, it still appears that a full round trip delay occurred between issuing of a command and receipt of telemetry. Because netem actually emulates network conditions, it was felt that netem was a more appropriate tool for introducing a communication delay. Netem allowed more flexibility in the experimental network design, despite challenges of data transmission over an actual network delay. Netem also allows emulation of variable time delay and other adverse network conditions, however the study presented in this work investigates only a fixed time delay.

2.1.1 Predictive Displays

Real-time teleoperation is increasingly difficult at higher and higher time delays. To mitigate this issue, supervisory control techniques and increased levels of robot autonomy have been developed and studied. One of these techniques is the predictive display. The predictive display attempts to graphically display a system's future state to the operator. First introducted in 1953 [32], predictive displays are common tools for reducing the performance degradation due to time delay. Predictive displays help maintain the effectiveness of an operator when performing realtime control [24] [26] [21] [1]. Predictors employ a mathematical simulation of the system to estimate the state of the robot after it responds to the command. This state is immediately presented to the user in the form of a graphical display. In a typical case, the predictive display is rendered with transparency or in a wireframe mode, and overlaid on top of an opaque model based on the actual system. An example of this type of overlay is shown in Figure 2.1.

One of the early predictive displays for robotic manipulation was developed by Noyes [26]. He used a predictive display to assist in teleoperation of a six DOF arm to perform a block moving manipulation task and a path tracing task. For the tasks, he overlaid a wireframe model of the manipulator over video with extremely low update rates. The rates considered were once every 0.5 seconds and once every 1.5 seconds. The low update rates are somewhat analogous to communication time delay, in that the operator must wait for a period to receive feedback from the system. Using the same robot and overlay system, Mar demonstrated the predictive

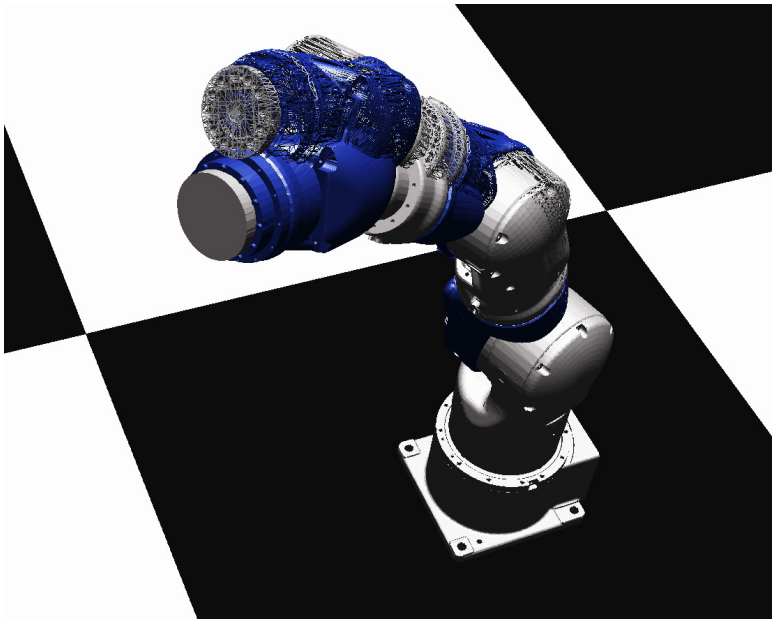


Figure 2.1: Wireframe Rendering Overlay on a Solid Model of a Motoman 10D. OpenSceneGraph model file courtesy of NASA Goddard Space Flight Center.

display times of 1.5, 3, and 5 seconds. [24]. She found improvements of 15-25% when using the predictive display at these delays.

Figure 2.2 illustrates a general predictive display algorithm. These early systems had open-loop control systems, and responded to force commands instead of higher level position or rate commands. The system's operator was required to close the control loop by commanding the adjustments necessary to maintain a desired operating parameter. Their force commands would be sent to both the actual and simulated system. Slight differences in when commands are sent can cause the calibration between the predictive simulation and the actual system to diverge, decreasing the predictive display's effectiveness and requiring periodic recalibration or sophisticated calibration algorithms.

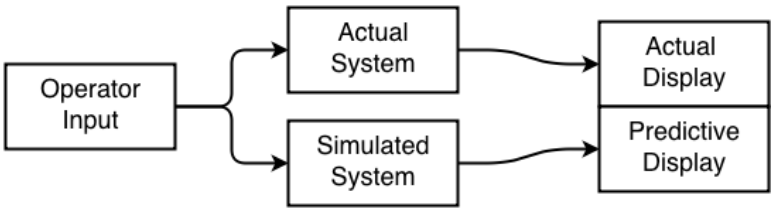


Figure 2.2: General Algorithm of Predictive Display System.

2.2 Supervisory Control

Early predictive systems were open-loop, requiring fine adjustment from the operator to close the control loop. Advances in technology led to increased local autonomy in robotic systems, in turn allowing more sophisticated levels of supervisory control [21]. With these systems, operators could control a graphical display to generate scripted or symbolic commands for robotic systems to autonomously follow [31] [30]. Operator input could be recorded and modified before sending to the actual robot, allowing any desired course adjustments.

Conway used a display to generate commands for a manipulator using the concept of position and time clutching [29]. The operator could disengage the time clutch and control the simulation faster or slower than they would control the real robot to set waypoints for the robot to follow. The commands would then be sent to the real robot, which would interpolate between the waypoints as it moved to the commanded positions. This technique effectively reduced the impact of time delay, and even improved performance for teleoperation without time delay.

2.3 The Commanded Display

Many modern robots are able close their control loops autonomously, allowing them to respond to higher level commands, such as holding a desired position. This allows systems where a control station can generate a commanded position, and send that position to the robot. Lane used this concept to develop a commanded display for realtime teleoperation [1]. The operator will controls a graphical display which continuously sends commanded positions a physical robot. The general algorithm is shown in Figure 2.3.

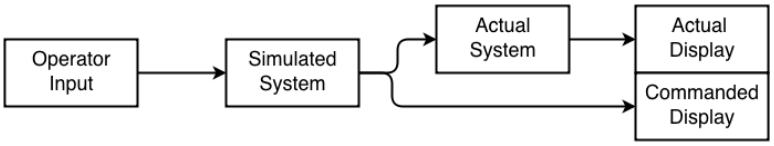


Figure 2.3: General Algorithm of Predictive Display System.

In a commanded display, as in a predictive display, the operator's inputs are passed into a simulation of the arm, which updates a graphical model to give the operator visual feedback. However, these operator's inputs are not directly sent to the actual robot. Instead, the commanded display simulation generates the commanded position of the robot. That commanded position is then sent to the arm. This allows the operator to drive the commanded robot in realtime while waiting for telemetry from the actual robot. After the time delay passes, the actual robot converges to the commanded robot's position. Because the commanded display produces a joint pose or position command, and because the robot will nominally be able to close the loop and autonomously go to that position, the simulation and actual system

will not diverge due to growing calibration errors.

Lane found that the commanded display eliminated up to 91% of the increased task completion caused by time delays of 1.5, 3.0, and 6.0 seconds in a virtual environment. Additionally, he compared the commanded display to a simple predictive display, which yielded reductions of only 40% to 50% at the same time delays.

Figure 2.4 illustrates the overall algorithm of the commanded display used in this study. The operator commands the velocity of the robot's tool-tip, $\dot{x}_c$, with the input device. The velocity is expressed as the commanded rate of the tool-tip's Cartesian position and orientation, x_c. The rate command is passed into a kinematic simulation of the arm. The simulation performs the inverse kinematics computations and position integration to calculate the commanded joint pose, q_c. The commanded position is simultaneously sent to the actual system and the commanded display. In the commanded display, the operator sees the results of their inputs immediately. Before reaching the actual system, the joint command is delayed by the communication distance. The robot receives the delayed joint command, q'_c, and updates its desired joint position, q_d. The robot then sends its current measured joint pose, q_m, back to the operator. The measured joint pose passes through the communication time delay and becomes the delayed measured joint pose, q'_m. The delayed pose then updates the operator's actual display. After the operator issues a command, they will see the results immediately on the commanded display. After the duration of the round-trip time delay has passed, they should see that the actual robot position converges to the commanded position.

The ultimate goal of this research is to increase the effectiveness of realtime

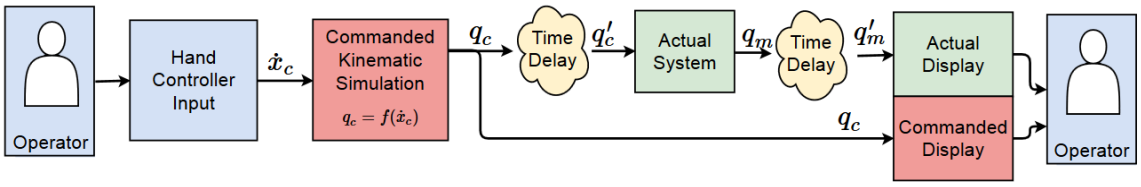


Figure 2.4: Block Diagram of Commanded Display Algorithm.

telemanipulation at high time delays. In particular, it seeks to implement the commanded display on a real-world robotic system. Previously, Lane had shown the performance benefits of the commanded display on a purely virtual system. Extending the commanded display to a physical system will verify Lane's findings and justify the use of the commanded display in future real-world applications.

2.4 Ranger NBV-I Arm

The Ranger NBV-I arm is a six degree of freedom (DOF) manipulator designed at the University of Maryland (UMd) Space Systems Laboratory (SSL). Figure 2.5 shows NBV-I mounted to an optical bench in the SSL's Advanced Robotics Development Laboratory. The arm is a serial link manipulator with six revolute joints, and the joints are arranged in a roll-pitch-pitch-roll-pitch-roll configuration.

It was originally designed in the 1990's as a camera arm for the Ranger Neutral Buoyancy Vehicle(NBV), which was designed to demonstrate spacecraft servicing tasks in neutral buoyancy. A recent overhaul by Ellsberry and D'Amore has returned the arm to operational status [6] [5], and it is now used as a research platform at the SSL. Table 2.1 gives the Denavit-Hartenberg(DH) parameters as measured by

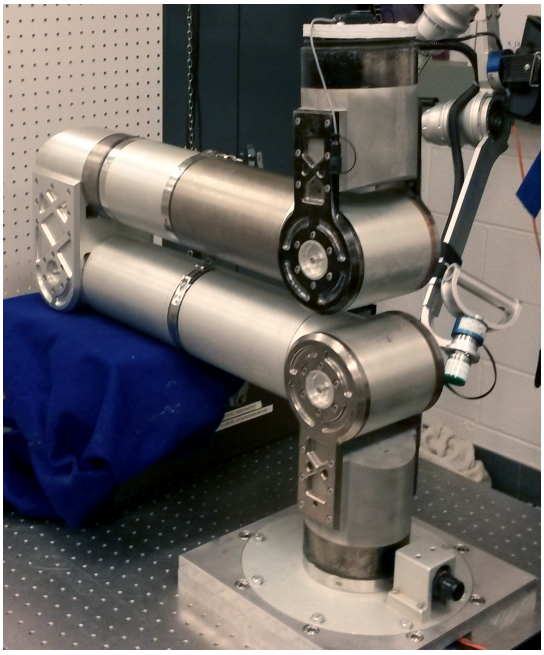


Figure 2.5: NBV-I Mounted to an Optical Bench in the SSL's Advanced Robotics Development Laboratory

Ellsberry, and Figure 2.6 shows the attached link frames. The DH parameters are listed using the modified Denavit-Hartenberg convention [12]. The arm is typically controlled by commanding the rate of the end effector Cartesian position with pair of hand controllers. The DH parameters are used by kinematic routines in the commanded display simulation in order to generate commanded poses from rate inputs. The tool-tip parameter was modified to account for the experiment's end-effector.

The visualization software developed for this study was required to communicate with NBV-I's Data Management Unit (DMU) software. The DMU presents 3 Universal Datagram Protocol (UDP) ports for communication. One is for general commands, and one each for streaming of Cartesian rate and joint pose commands. The streaming ports are used for realtime teleoperation. Teleoperation using the

i	α_{i-1} (deg)	$a_{i-1}(m)$	$d_i(m)$	θ_i(deg)
1	0	0	0.2491	θ_1
2	90	0	0	θ_2
3	0	0.5589	0	θ_3
4	-90	0.1514	0.5388	θ_4
5	90	0	0	θ_5
6	90	0	0	θ_6
T	0	0	0.2666	0

Table 2.1: Denavit-Hartenberg Parameters of NBV-I as Measured by Ellsberry [6]

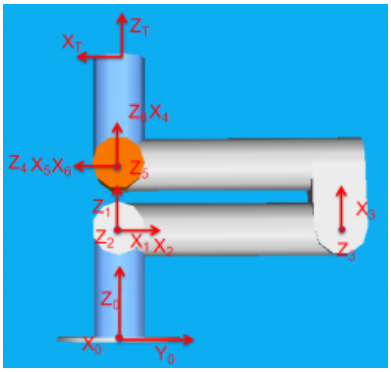


Figure 2.6: Rendering of NBV with attached link frames, from [6]

commanded display required controlling the arm in joint space, so the joint command port was the most important of these.

2.4.1 Input Devices

2.4.1.1 Two 3DOF Hand Controllers

The use of a pair of two 3DOF hand controllers is a common method of controlling a robotic arm on-orbit. The operator inputs desired Cartesian rates of motion of the robot's end effector or tool tip. The translational hand controller is mapped to the Cartesian translation of the robot's end effector. Likewise, the rotational hand controller is mapped to the end effector's orientation. Figure 2.7

shows the hand controllers used in the present work, along with their mappings to end effector motion.

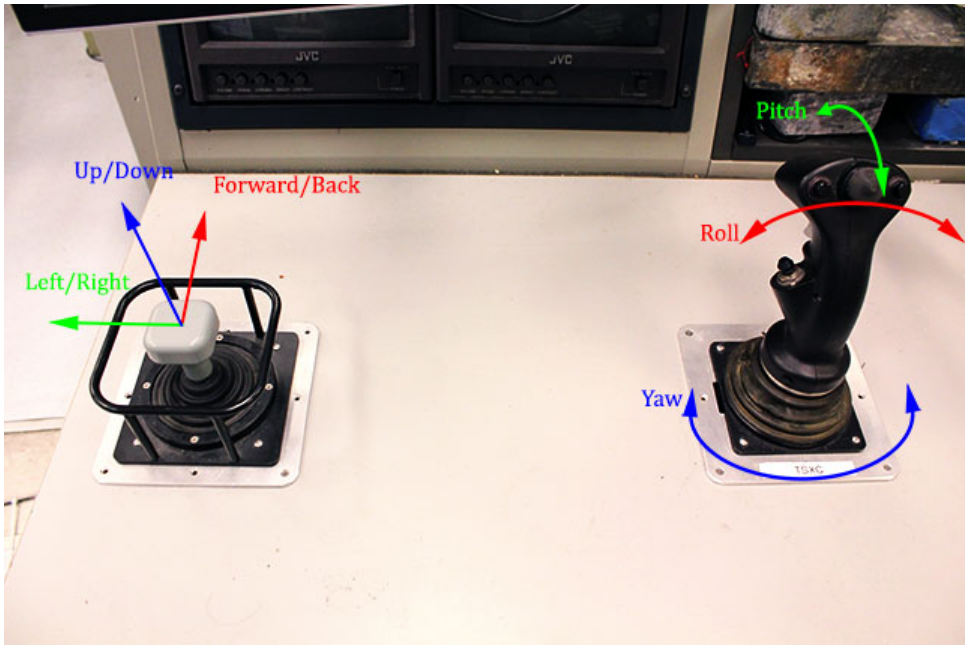


Figure 2.7: Translational and rotational hand controllers with axes and mappings shown.

2.4.1.2 SpaceNavigator

The SpaceNavigator™ is a six DOF human input device manufactured by 3Dconnexion. It combines three translational and three rotational input axes in its pressure sensitive handle. This allows the user to control the position and orientation of a 3D object with a single hand. It sees wide use in computer aided design packages, 3D modeling and animation, and other visualization applications. Although it is designed primarily for interacting with 3D environments, it is also suitable to Cartesian control of a robotic manipulator. Unlike the hand controllers discussed in the previous section, the SpaceNavigator is highly portable. It is a USB

device and is compatible with Mac OSX, Microsoft Windows, and Linux/X11. It also does not require mounting into a work station table.

2.4.2 Software Libraries

Software reuse has the potential to save a significant amount of time and cost from a development effort. The present work utilizes several existing software libraries and frameworks.

The following subsections describe the software libraries chosen for the current work. Each has been designed with the goal of facilitating software development and offers a unique set of functionality.

2.4.2.1 OpenSceneGraph

OpenSceneGraph(OSG) is an open source 3D graphics toolkit [16] [17]. It is commonly used by developers to create visual simulations, virtual reality applications, and provide scientific data visualization. Designed to be highly portable, OSG is written in standard C++ and uses OpenGL. OpenGL is an API that gives developers access to the graphics hardware. OpenSceneGraph is rendering middleware, which abstracts many of the low level calls and provides an object oriented interface to the developer. OSG runs on most common operating systems, including Windows, OSX, and GNU/Linux. OSG supports many common 3D file formats, including COLLADA and 3D Studio Max. It also defines its own "osg" 3D file format.

OpenSceneGraph's main purpose is to provide rendering capabilities to the developer. As indicated by its name, OSG contains a scene graph implementation. A scene graph is a data structure that contains and arranges the logical representation of elements within a graphical scene. Nodes in the scene graph represent objects in the scene. Nodes are arranged in a tree structure, in which a node may have a parent and children nodes. This allows the developer to define spatial relationships between nodes. A geometric node will often have a transformation matrix that defines its spatial position with respect to its parent node. If the parent node moves, the child node retains that relationship, and moves along with the parent.

The core of OSG provides the functionality to load, manipulate and render the scene graph. It defines classes for scene geometry, transformations, and a variety of other node types. It also defines a number of geometric primitives, including points, polygons, a vectors. These classes include methods for vector and matrix operations. OSG also provides an abstraction of OpenGL functionality. It provides the rendering backend for the scene graph, which must transform the scene graph into the necessary OpenGL calls.

The OpenSceneGraph library also contains a number of NodeKits. NodeKits provide additional functionality beyond the basic 3D capabilities. These typically implement frequently used functionality, such as text, particle generation, and post-processing. Of particular importance to this work is the osgQt NodeKit. This NodeKit provides an easy path for integration of OSG within Qt applications. It also enables the developer to embed Qt GUI components within an OSG application.

2.4.2.2 Qt

Qt is a software framework designed by the Qt Project for developing application software [18]. Qt is used primarily for developing applications with a graphical user interface(GUI), but it is also suitable for applications without a GUI. It is cross-platform, supporting all major desktop operating systems, as well as several mobile platforms. Qt is currently developed by Digia and the Qt Project, and available under commercial and open source licenses.

Qt provides several useful features for application development. As with any GUI toolkit, Qt provides an application programming interface (API) to make use of widgets. A widget is simply an element in a GUI. Developers may choose from a number of predefined widgets, or they may create their own. Custom widgets inherit from Qt's existing widget classes, removing the need for developers to re-implement existing widget functionality.

One unique feature in Qt is its "signals and slots" mechanism [19]. This mechanism provides infrastructure for communication between objects. Objects may use this to pass event and other state information to other objects. Objects send this information with a signal. The receiving object accepts the information on a slot. Slots are functions that are called whenever they receive a signal. This method offers several advantages over other GUI frameworks. Traditionally, a processing function will be passed a pointer to a callback function. The callback function is then called when appropriate. One major drawback to this method is that the processing and callback functions are tightly coupled. The processing function must

be able to call the right callback function. Signals and slots allows a more modular approach. Slots simply know what type of information to expect. This allows a more modular approach to application programming. Qt already has predefined signals, and the developer is free to define their own.

Qt also provides functionality relevant to general application development. The ones most relevant to this work include XML parsing, thread management, network sockets, and file handling.

2.4.2.3 Orocos Kinematics and Dynamics Library

The Open Robot Control Software (Orocos) Project [4] was initiated to facilitate advanced robotics research. Orocos promotes software reuse and rapid software implementation through object-oriented and component-based programming paradigms. The project itself encompasses several smaller projects. The project of most relevance to this work is the Kinematics and Dynamics Library (KDL).

The Orocos KDL provides functionality for kinematic and dynamic calculations for serial-link systems. It defines a number of classes to represent kinematic chains. These include the KDL::Chain and KDL::Joint classes. The chain class contains information about the kinematic chain, while the KDL::Joint class represents each degree of freedom in the chain. KDL provides solvers to compute the forward and inverse kinematics of any kinematic chain. KDL also defines classes useful for any dynamic system. This includes a number of geometric primitives such as vectors, rotations, and three-dimensional transformations.

2.4.2.4 OpenCV

The Open Source Computer Vision Library (OpenCV) was designed to support computer vision projects. The library provides over two thousand algorithms for computer vision and machine learning. These algorithms can be used to track human gestures, identify objects, extract 3D models of objects, and identify features for augmented reality overlays. The present work simply uses OpenCV to obtain imagery from a camera source to display to a user, however OpenCV has many possible applications in the development of future visual interfaces.

2.4.3 Software Utilities

This work employed several software utilities in parts of the experimental setup. The following subsections describe each utility and its function.

2.4.3.1 netem

The netem™ utility is a packet scheduling tool designed to perform network emulation of wide area networks [7]. It was designed to simulate non-ideal network conditions in order to test software on a local network. Conditions netem can emulate include fixed time delay, time delay jitter, packet loss, packet re-ordering, and bandwidth limitations. This utility was used in this research to emulate high time delays.

2.4.3.2 VLC media player

VLC media player™ is a versatile media player and media streaming server developed by the VideoLAN Organization [8]. Its first open source release occurred in 2001, and it has been continuously developed since then. VLC supports most modern operating systems, including Linux, Windows, and OSX. Additionally, it can play almost all common media sources and formats. Of particular interest to this work is H.264 video compression format. It can both encode and decode video sources in this format. The current application was to encode raw video from a camera at the worksite into an H.264 format, and stream it over a network. The VLC instance on the receiving end then decoded the stream and displayed it to the computer screen. VLC supports a variety of streaming options, including RTSP/RTP. RTSP, or Real Time Streaming Protocol, is a network protocol widely used by streaming media servers; RTP, or Real Time Transport Protocol, defines a packet format for carrying media over a network.

Chapter 3: Software Development

This chapter describes the visualization software developed for this research. The goal was to develop a flexible visualization and simulation system that would support the current and future teleoperation research. Modularity and code reusability were emphasized in the design. It is hoped that this approach will facilitate future development for future projects. Software development leveraged open source projects to reduce development time and for community support.

3.1 Overview

Figure 3.1 illustrates the current software design. The software is a single Qt GUI application with two threads. The blocks represent the individual class objects that comprise the system. The illustration shows the main connections and data flow between objects. The primary messaging infrastructure is provided by Qt's signals and slots mechanism. Each block inherits from Qt::Object, either directly or through defined Qt GUI classes, giving each object the signal and slot messaging capability.

The software is multi-threaded. The main application thread handles simulation and GUI computations. This thread is starts upon application startup. The

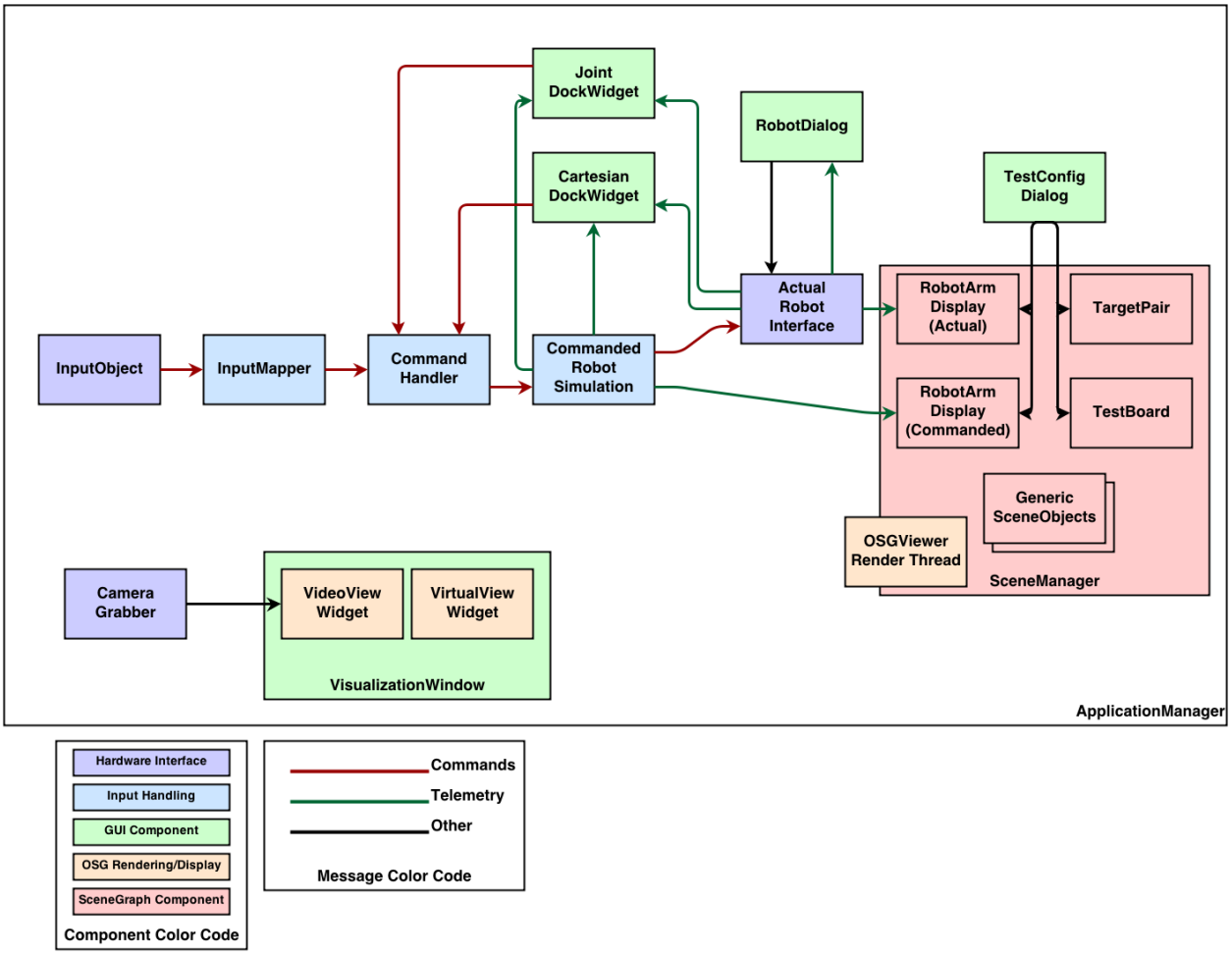


Figure 3.1: Overview Block Diagram of Software Design Showing Main Components and Data Flow.

second threads is dedicated to rendering. The OpenSceneGraph rendering object, VisualizationWidget, is loaded after the program completes initialization. This object contains the root node of the scenegraph and runs the update and rendering loop. Threading is handled by Qt's Qt::Thread class. The main application instantiates a Qt::Thread object and moves the rendering object is moved to it. Because signals and slot messaging is thread safe, thread safety is ensured.

3.2 Commanded Robot Kinematic Simulation

The kinematic simulation forms the basis of the commanded display. It is implemented in the CommandedRobotSimulation object. This object's purpose is to transform the operator's Cartesian rate inputs into a commanded joint pose for the robot arm. This object inherits from a GenericRobotInterface class which defines the needed messaging interfaces between components.

The GenericRobotInterface is an abstract class that is the basis for representations of robotic manipulators. It defines the signals and slots needed to receive commands and transmit telemetry. By inheriting this class, robot objects can be implemented and connected without reimplementing interfaces. It also ensures consistent interfaces between these objects. The key slots include Joint pose and Cartesian position commands, and commands to initialize and set the internal state variables. The telemetry is sent through several signals. Of particular importance are the current joint pose and Cartesian positions. In the commanded display implementation, the joint telemetry signal from the CommandedRobotSimulation is connected to the joint pose command slot of the ActualRobotInterface.

The CommandedRobotSimulation object keeps track of the simulated joint pose and Cartesian state. When it receives a rate command, it performs an inverse velocity kinematic routine to calculate corresponding joint rates. It then integrates the joint rate over the time step to get the change in joint angles. Finally, it sums the change with the current joint angles to get the new pose. This new pose is the commanded pose that is sent to the actual robot. This pose also is sent to a forward

position kinematic routine to calculate the new Cartesian tool-tip position. Both the pose and telemetry are sent to the joint and Cartesian DockWidgets, respectively.

Orocos KDL provides the kinematics routines used in the simulation. The KDL::ChainIkSolverVel_wdls class performs the computations for the inverse velocity kinematics, while KDL::ChainFkSolverPos_recursive performs the forward position kinematics. These solvers were wrapped in Qt::Object classes, which were then connected to the main CommandedRobotSimulation object. The kinematic objects set up the supporting functions, the kinematic chain representations, and initialization required to use the KDL solver classes. They initialize the kinematic chain using the real robot arm's D-H Parameters. The D-H parameters for NBV-I were used for the majority of this work, however these objects could be used with any manipulator, given its D-H Parameters.

Telemetry from the actual robot can be used to update the simulation's state. This is typically used for initialization of the object. For teleoperation, the joint pose of the commanded simulation and the actual robot must agree before the simulation begins sending commands to the actual robot. Different initial positions will result in large sudden movements of the actual robot as it moves to the commanded position. The operator may "latch" the commanded display output, however. Disabling the command stream from the commanded robot can allow the operator to move the commanded display without sending commands to the actual robot. The operator can then send the resulting position to the robot at a later time.

3.3 Actual Robot Interface

The hardware robot interface is implemented in the ActualRobotInterface object. This object inherits from GenericRobotInterface, described in the previous section. This concrete implementation of the abstract class defines the methods necessary to send and receive data from the SSL's Data Management Unit software.

Communication with the DMU occurs over three Universal Datagram Protocol networking sockets. The first UDP port is used for general robot commanding and offline teleoperation. Figure 3.2 shows the Qt::Dialog window that gives the user access to these commands. The key startup commands are: (1) requesting Command Authority, (2) enabling Robot Control, (3,4) enable Joint or Cartesian control modes, (5) load and issue trajectory commands, (6,7) enable online Joint or Cartesian teleoperation streams. The dialog highlights important information, such as the control mode of the robot, and whether or not the software currently has Command Authority to issue commands. The dialog also outputs error message strings from the robot. The robot and dialog objects reimplement much of the Text User Interface developed by D'Amore [5] in a Qt Framework.

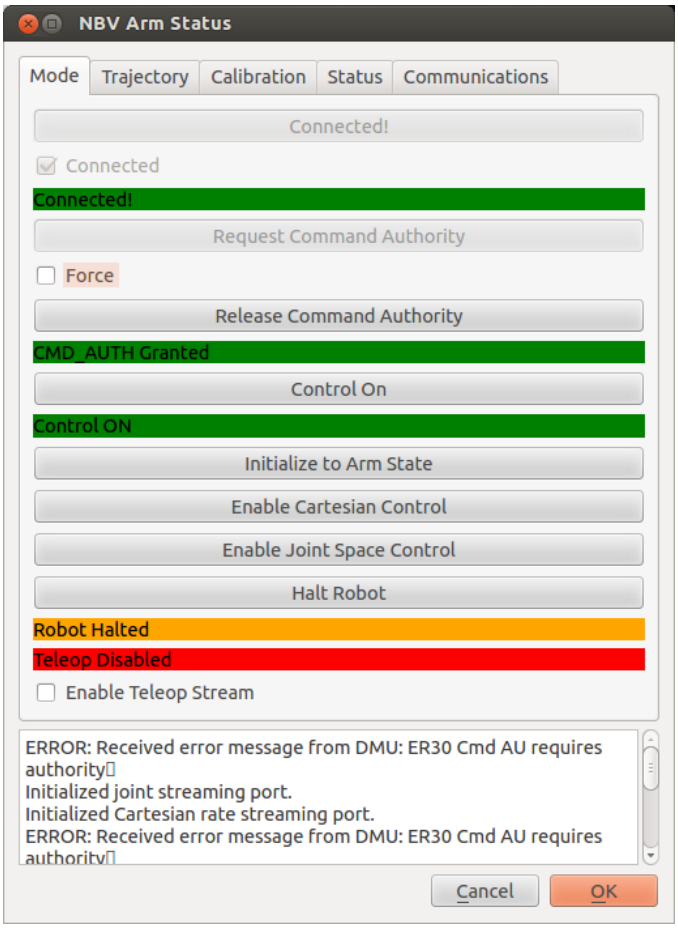


Figure 3.2: Dialog Box to Interface with NBV-I

The other two UDP ports receive streaming data for direct teleoperation. One socket is used for joint space streaming, and the other receives Cartesian rate commands. The commanded display algorithm generates commands in joint space, so the software primarily used the joint stream port.

3.4 Input Devices and Handling

The software was designed to enable rapid implementation of new input devices. The input chain performs two main functions: (1) receive incoming rate commands, (2) map incoming rate commands. The CommandHandler Each of the functional blocks inherit Qt::Object, enabling the use of signals and slots. All data interfaces between these objects are implemented as signals or slots. The input object defines its own interface to obtain the raw input.

The InputObject is responsible for receiving input from external sources. An abstract input object was defined with the interfaces required to send data to the rest of the input chain. This allows input object implementations to inherit the necessary interfaces from the generic object. The input device object can be instantiated and connected to the input chain at runtime. A configuration file identifies the specific input device to use. The software then loads the selected device during application initialization.

Two input device objects have been implemented, as of this writing. The first to be implemented was the handler for the SpaceNavigator 3D Mouse, described in Section 2.4.1.2. The SpaceNavigator object links against the device's library, which

defines the device API. It uses the functions provided by the API to check for new input data. The data contains the state of the six axes and two buttons of the device. The device axes are mapped to x, y, z, roll, pitch, and yaw before being packaged into a KDL::Twist object. The data is emitted after it is read and in the proper format.

The second input device object does not directly interface with an input device. Rather, it implements the SSL's input device client interface. The SSL's device clients read raw data from their respective input devices and transmit rate commands over a UDP socket. The object receives that stream of input data over its corresponding UDP socket. It creates the socket using the Qt::QUdpSocket class. This UDP input object is primarily used to receive data from the SSL's translational and rotational hand controllers, described in Section 2.4.1.1. Although only the hand controllers have been tested with this object, any device client that outputs data according to the SSL input device client interface would work with this software.

The InputMapper object remaps incoming rate commands to a desired scale. The linear mapping is primarily used to rescale the input. This is particularly useful for converting raw hand controller axis readings into the desired range of rates. The linear mapping can also be used to change the maximum allowable range of rates. An input device with a scale from -20 to 20 cm/s could be remapped to -10 to 10cm/s, for instance. The square mapping remaps a linear input scale according to a square law. This gives the operator a greater input range at low speeds. It is possible to perform both linear and scale mappings on the same input source.

The output of the InputMapper goes into the CommandHandler. In addition to an input device, a user may issue commands through the controls provided on the joint or Cartesian control dock widgets. The CommandHandler controls which of these input sources is passed through to the robot. The GUI windows supersede hand controller input when they issue a command.[1]

3.5 Application Manager and GUI

The ApplicationManager object is the application's main class. It inherits Qt::MainWindow and sets up the initial GUI window. It also handles application start-up and shut down.

The ApplicationManager is responsible for the application's initialization. It first reads the configuration file specified by the command line arguments. It creates and connects all of the main application components according to the configuration parameters. It also creates the rendering thread for the visualization.

The ApplicationManager is the "parent" of all the GUI windows and dialog boxes. The class itself inherits from QMainWindow and creates the main application window. This window is home to the CartesianDockWidget and JointDockWidget widgets. As their names suggest, these widgets "dock" with the main window, appearing to be part of the window.

The main window has the application's main menu. The menu is used to

[1]The main development of the current system focused on the visualization and commanded display of the robot. The GUI features only basic go-to commands. A more full-featured GUI would implement more sophisticated controls, and the CommandHandler should handle inputs in a more rigorous manner.

launch the other application windows. It has menu options for loading the visualization window, test configuration dialog, and robot dialog. It also has several functions to support the experimental testing discussed in Chapter 4. These include functions for data logging and saving camera views.

3.6 Scene Graph

The scene graph is the virtual representation of the robot worksite. It is constructed by the SceneGraphManager object. Its individual components are controlled by SceneObject components.

The SceneObject components represent and handle individual objects in the scene graph. The basic purpose of the SceneObject is to wrap OSG node in a Qt object, and expose the node's OSG functions to the signals and slots mechanism. The basic SceneObject class allows other objects to send messages to controlling the SceneObject's position and orientation.

Specialized SceneObjects represent objects in the scene graph that require additional functionality. The RobotArmDisplay object inherits from the basic SceneObject class, gaining its basic wrapped OSG functions. In addition to the position and orientation of the robot's base frame, the SceneObject must specify the joint angle of each degree of freedom in the arm model. When the RobotArmDisplay object first loads the arm model, it traverses the model tree to identfy the transforms that correspond to the joints. It creates a slot function that accepts a KDL::Joint to update the full joint pose of the model. The RobotArmDisplay also adds a frame

to the end of the robot to allow the attachment of an end-effector. For the present work, a basic SceneObject loads a model of the tool-tip. The SceneGraphManager then attaches the tool-tip's base frame to the robot model's tool-tip frame.

Specialized TestBoard and TargetPair SceneObjects were implemented for the experiment described in Chapter 4. The TestBoard object loads the task board model, which contains a frame for the target position. The object has a slot for controlling the position of the target object on the board's face. The TargetPair object is solely positioning of the target models, it loads no models itself. It consists of a frame that corresponds to its center, and it has two frames that correspond to the target positions. It has a slot for specifying the desired horizontal target separation. Two basic SceneObjects load the target model. One of those SceneObjects is attached to the TargetPair's left target frame, and one is attached to the right target frame.

The SceneGraphManager initializes all of the SceneObjects and builds the main scene graph. It specifies which model files each SceneObject loads, and how to attach the SceneObjects to other SceneObjects. The manager also contains the root node of the main scene graph. It attaches the base node of the SceneObjects to the root node. After program initialization, the ApplicationManager accesses the root node and passes it into the RenderDriver object and starts the visualization.

The RenderDriver is essentially a Qt wrapper for OSG's osgViewer::CompositeViewer class. This class provides the rendering functionality for the graphical environment and supports multiple views of the scene. The RenderDrive inherits osgViewer::CompositeViewer and implements the main rendering loop. This allows

the RenderDriver to be moved to a dedicated Qt::Thread.

3.7 Visualization Window

The VisualizationWindow object provides the operator's main visual interface. Its main purpose is to initialize and display the widgets that provide views of the workspace. This window was primarily designed to display virtual views, but optional video views have also been integrated into the window. The VirtualView widget displays a view of the graphical environment, and the VideoView widget displays video from a hardware camera.

The VirtualView widget is designed around an osgViewer::View object. The widget handles initialization for the view. It sets up the osg::Camera and event handler for the view, sets the view's scene graph data to the main scene graph's root node, and adds its osgViewer::View object to the RenderDriver's osgViewer::CompositeViewer object. After setup is complete, the VirtualView adds the view's GLWidget to its layout. The GLWidget is created by using the osgQt::GraphicsWindowQt class to set the camera's graphic context, where the graphic context is the window in which the graphical view will be drawn. This class enables the OpenSceneGraph and Qt integration.

The virtual camera may be a fixed view, or adjustable. OSG provides several useful classes for manipulating the camera view. For adjustable views, an osgGA::TrackBallManipulator is attached to the camera. This allows the user to position and rotate the camera using mouse input. If a fixed view is specified, the

VirtualView widget does not attach a manipulator to the camera. Instead, it manually sets the camera's view and perspective matrices based on previously saved camera views.

The VideoView widget displays video from a hardware camera source. This widget is very similar to the VirtualView widget. It utilizes OpenSceneGraph for rendering, and performs the same initialize steps described previously. The key difference is that the VirtualView does not use the main scene graph's root node for the view's scene graph data. It creates a textured quadrilateral polygon, and renders the incoming camera video to this polygon. The osg::Camera is given a fixed orthographic projection matrix to appropriately display the image to the view window.

Chapter 4: Fitts' Law Study

4.1 Overview

This work utilized a Fitts' Law tapping task to investigate the effectiveness of the commanded display. Table 4.1 lists the variables tested. The contact parameters were selected such that two of the IDs would overlap, giving 3 distinct IDs.

Table 4.1: Fitts' Law Task Summary

Task: Fitts' Law Task		
Independent Variables		Dependent Variables
Time Delay:	0.0, 0.33, 1.0, 2.0, 4.0, 6.0 seconds	Task Completion Time
Display Method:	No Overlay, Commanded Display	NASA Task Load Index
Target Size:	2.0, 4.0 in	
Target Separation:	11.0, 25.0 in	
Total Test Cells:	48	

4.1.1 Participants

Five adult males participated in the study. Technical background was a key criteria for recruiting test subjects, and three test subjects had direct previous experience operating robotic systems. All subjects had a background in an engineering discipline. Each subject was right-handed. Four subjects had vision correctable to 20/20, and one subject had vision corrected to 20/100.

UMd's Institutional Review Board (IRB) approved the use of human test subjects for the testing conducted in this work. A key component of the IRB process is obtaining informed consent from each participant. The approved consent form is included in Appendix C. Each part of the form was explained to each new test subject. Key areas of discussion included the potential risks, the voluntary nature of the study, and protection of anonymity. If the prospective subject agreed to volunteer for the study, they then signed the consent form. Each Subject's identity was tracked for this work, due to the need for multiple test sessions. The subject's anonymity was protected with the use of a keyed list kept in a secure facility.

4.2 Fitts' Law Task

The experiment was a two DOF Fitts' Law task. Fitts proposed a relationship to characterize a simple human manipulation task. [11] In the task, a human moved their finger from one target to another. Fitts' Law relates the time to move between targets to the distance between the targets and the target size. Based on these parameters, Fitts defined the Index of Difficulty to characterize the difficulty of the manipulation task. The index of difficulty is given by the following expression:

$$ID = log2(2 * D/W) \tag{4.1}$$

where ID is the Index of Difficulty, D is the distance between targets, and W is the target width. The difficulty of a task is directly proportional to the size of the targets and distance between them, and task completion time has been found to relate linearly with ID for many tasks. For each target configuration considered, targets

were placed in a horizontal line with some separation. The goal of the operator was to move the robot tool-tip back and forth between the targets as quickly as possible, and to contact the targets with the tool-tip during each traverse.

4.3 Experimental Setup

Figure 4.1 shows the overall view of the physical and virtual robot workspace used for this task. The physical workspace includes the robot arm, task board, targets, and contact sensors. The virtual environment models the key workspace elements, particularly the robot arm, task board, and targets.

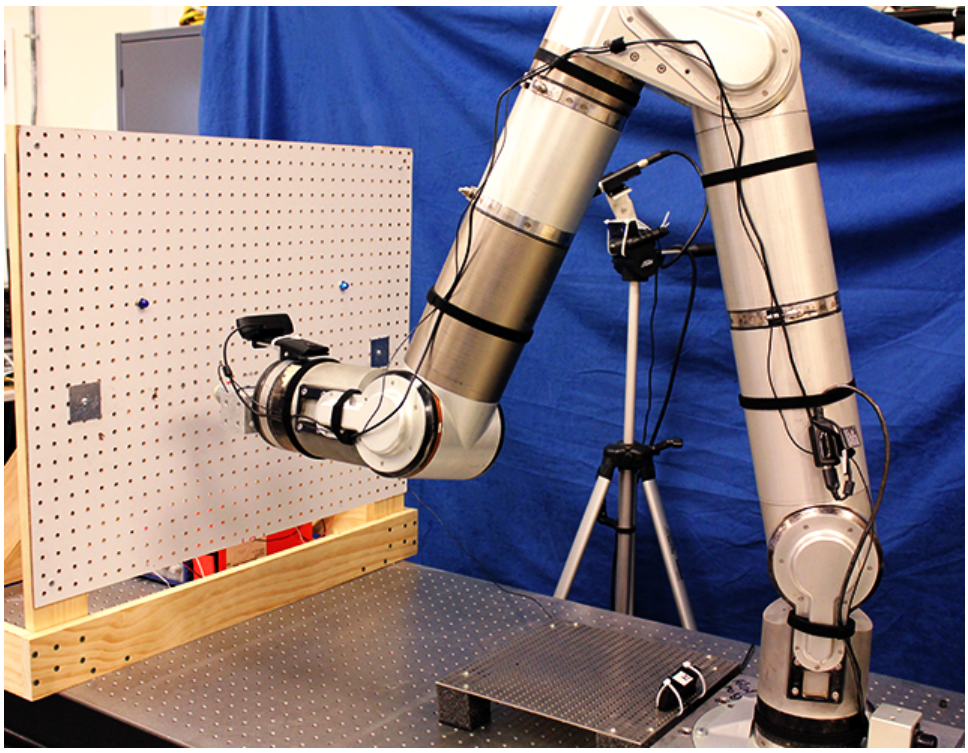


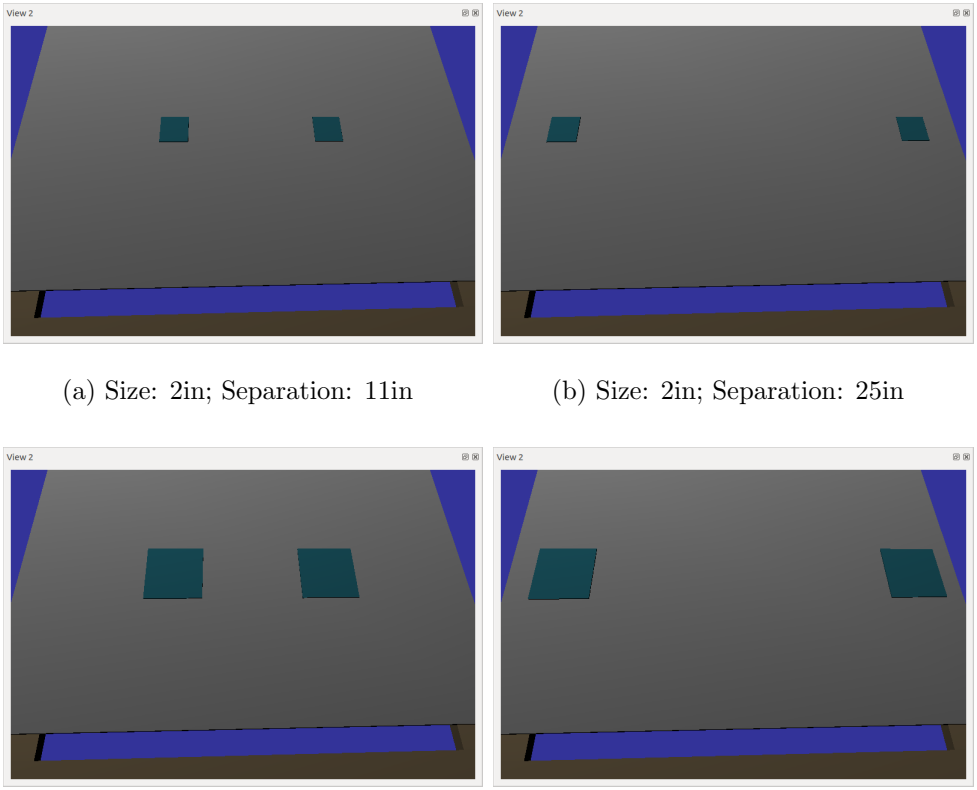
Figure 4.1: NBV-I Mounted to an Optical Bench in the SSL's Advanced Robotics Development Laboratory

4.3.1 Task Board

The task board held the targets for each task. The physical task board was a sheet of pegboard, with a grid of holes spaced one inch on-center in both the vertical and horizontal directions. Thus, the targets could be rapidly mounted in one inch increments vertically or horizontally. The task board itself was mounted to an optical bench, which also had holes with a one inch spacing. The entire test board could be moved forward or back in one inch increments, but its location remained fixed throughout the experiment. The visualization software has corresponding controls for the task board and target positioning in the virtual environment. Only the horizontal target spacing was varied between tasks in this study. Figure 4.2 shows a virtual view of each target configuration considered.

The physical task board had one lamp indicator on each side of the board. The lamp above the active target was illuminated. This gave the operator a visual indication of successful contact. When the operator made contact with the active target, the lamps switched. The active target becames the inactive target, and its light turned off. The previous target became the active target, and its light was turned on.

The task board's contact sensors and indicators were controlled by an Arduino microcontroller, shown in Figure 4.3. The key elements are the contact detection and the lamp control. The targets themselves were the contact sensors; they were aluminum plates that were held at digital high through a 10KΩ pull-up resistor, and one of the microcontroller's digital input/output(IO) pins was connected to



(a) Size: 2in; Separation: 11in

(b) Size: 2in; Separation: 25in

(c) Size: 4in; Separation: 11in

(d) Size: 4in; Separation: 25in

Figure 4.2: Virtual View of Target Configurations

each plate. Contact was detected when the electrode on the arm's tool-tip made contact with the plate. The tool-tip electrode was grounded, so the target contact was pulled to ground during contact. The microcontroller detects this change on its IO pin and updates the target states. The embedded software only changed the target states when contact was detected on the active target, so multiple hits on the non-active target would not be registered. The microcontroller also controlled the state of the indicator lamps. The active target's lamp would always be on, and the non-active lamp would be off. The lamps were powered by a benchtop power supply The lamps are driven with N-type MOSFETs in a low-side switch configuration. To

turn on a lamp, the microcontroller drives the mosfet gate high with a digital I/O pin. Likewise, setting the IO pin low turns off the lamp.

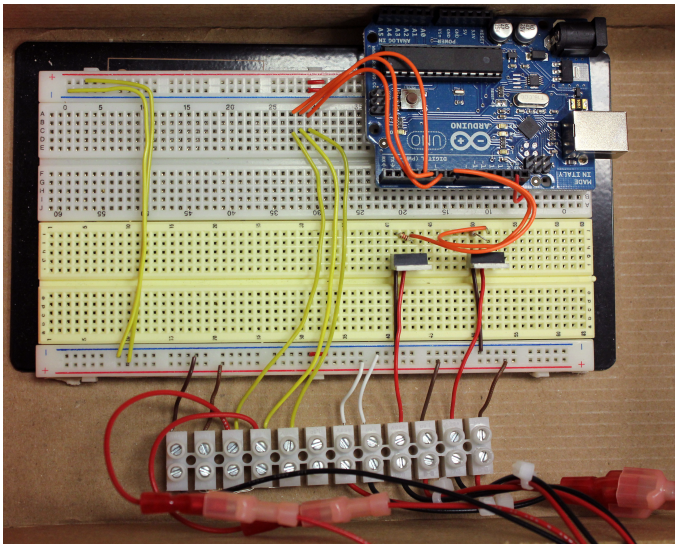


Figure 4.3: Electronics for task board control and contact detection.

The Arduino was connected to the task server, described in 4.3.2. A program on the task server relayed target state data from the Arduino to the visualization software on the workstation computer. It opened the a serial connection with the Arduino, read and parsed the incoming serial data, and then transmitted the data over the network via UDP. The software on the workstation then synced the target data with the telemetry data. In this way, the task board state was treated as another piece of telemetry from the remote worksite, and experienced the same delay as the robot telemetry before it reached the operator's workstation.

4.3.2 Network Overview

The experiment utilized five computers connected on a gigabit Ethernet local area network. Figure 4.4 illustrates the network layout. The network was divided into two main sections. The first section is the workstation side. These are the computers that handled the operator's interfaces for the tasks. These were the main workstation computer and the hand controller server. The other section, the arm-side, contained the computers involved with the arm and telemetry. These computers were the arm's Data Management Unit (DMU) and the Test Server. The two sides are separated by the time delay unit (TDU). Figure 4.5 illustrates the messages sent between each of the computers. The workstation computer was the center of this configuration. It handled the operator's inputs, telemetry from both the task server and DMU, and issued commands to the DMU. The TDU did not issue or handle any task data, aside from introducing the time delay.

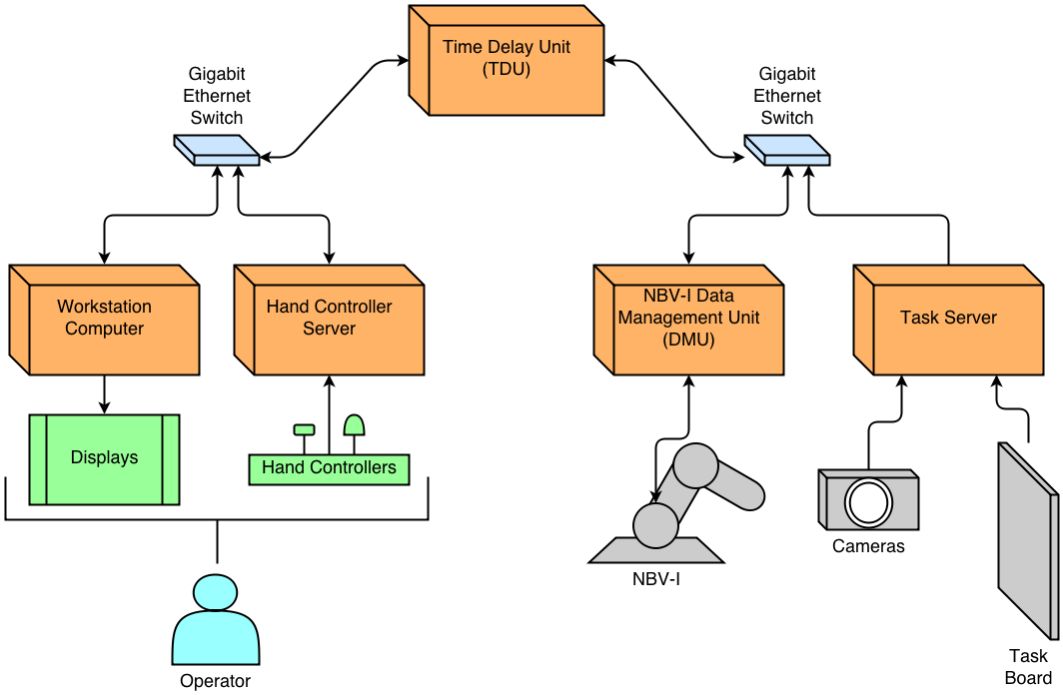


Figure 4.4: Diagram of the local area network in the test configuration.

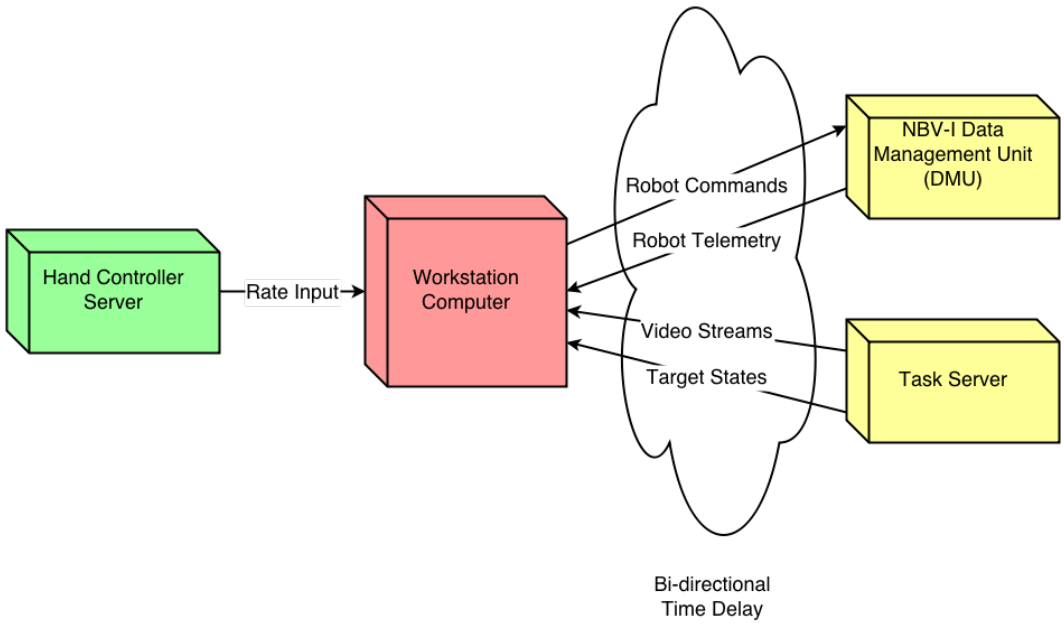


Figure 4.5: Illustration of message handling in test configuration.

4.3.3 Operator Workstation

The operator workstation contained the display and input interfaces used by each test subject. Figure 4.6 shows how the workstation was configured for this experiment.

Figure 4.6: Overall view of the operator workstation. Shows screen arrangement and hand controllers.

The three DOF translational and three DOF rotational hand controllers were the primary input interface for this experiment. They are mounted into the workstation table. The hand controller server read the inputs and mapped the values to commanded Cartesian rates according to a square law. It then streamed the mapped rates to the main workstation computer.

The operator's visual interface was provided by two LCD monitors connected to the main workstation computer. The workstation displayed both the graphical

simulation and camera video of the arm. Four graphical views and four camera views were provided. The graphical views included an orthogonal view from the side and bottom of the target board. One view gave an overview view of the task board. A fourth view gave an overall view of the robot and its workspace. The latter view was primarily so the operator could ensure a safe arm configuration, while the the three former views gave the operator a closer view of the task site. The actual camera views corresponded roughly with the virtual views, with one exception. The task board overview was replaced with a tool-tip camera. The camera was attached to the robot's distal linkage and included the tool-tip contact in its field of view. All views were displayed simultaneously. In order to delay the camera video, it was necessary to use video from a network source. The workstation receives the camera feed from the test server. The workstation then opened and displayed the stream using the VLC media player.

4.3.4 Time Delay Unit

The TDU introduced the communications time delay. The TDU sat between the robot and operator sides of the network. It was a Xubuntu 12.04LTS computer with two network interface cards (NIC). The ethernet switches on either side of the network connected to one of the two NICs. The network interfaces were bridged to connect the two network segments. The netem utility introduced the time delay emulation across the ethernet bridge. Netem adds a time delay to each network interface, resulting in a time delay that acts in both directions. With netem, the

TDU was capable of generating variable time delays, packet loss, corruption, and other poor network conditions. A more detailed description of netem can be found in Chapter 2. Each time delay treatment in this study was emulated as a constant, bi-directional time delay. The time delay in each direction was half of the total round trip time delay.

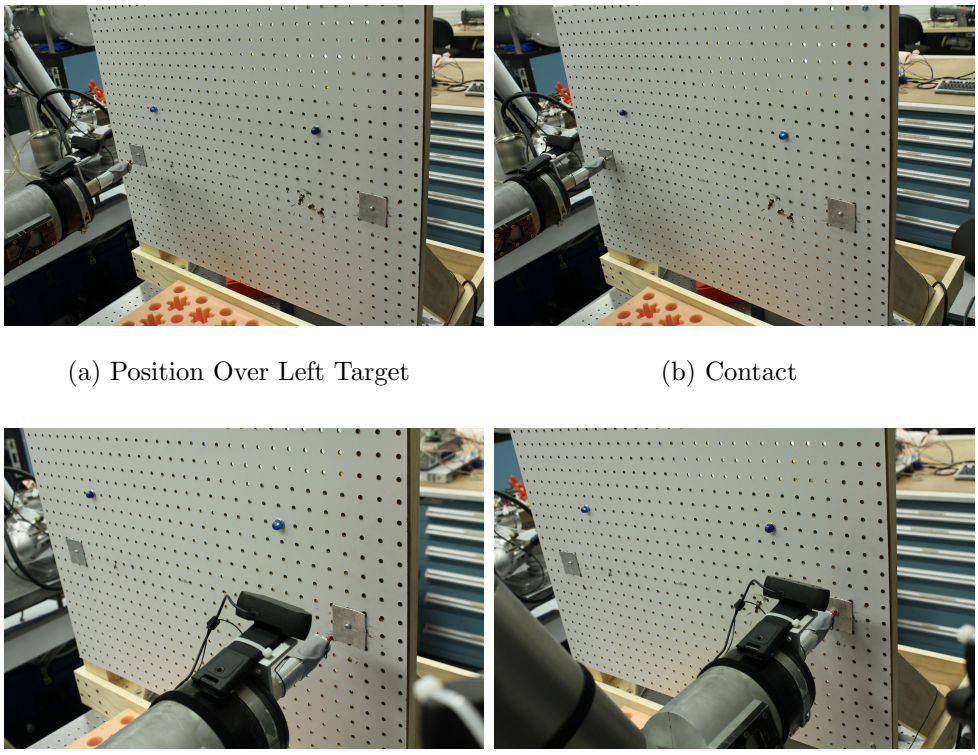
4.4 Procedure

Each new test subject was first familiarized with the task. They were shown the workstation, robot, and task board. Each component, its functions, and possible configurations were described. The subject was then shown how to operate the NBV-I arm using the hand controllers and visualization software. The subject then underwent a training period before performing the experiment. They informally practiced moving the arm's tool-tip between targets and hitting the contacts. They then performed several test trials. After the test trials, they filled out a practice NASA Task Load Index form.

The experiment was split over two or three hour sessions, depending on the availability and fatigue of the particular subject. Each test session lasted between two to three hours. Subjects performed no more than a single test session in a day. Test sessions did not exceed three hours.

For each test, the subject performed a series back-and-forth traverses between the targets with NBV-I. Figure 4.7 shows the general sequence. The operator positioned the target over the first target, made contact, then moved the arm to the

next target, and again made contact. This was repeated five times for each test cell, for a total of 10 target hits. Lights over the target would change to indicate the active target and provide visual confirmation of successful target contact.



(a) Position Over Left Target

(b) Contact

(c) Position Over Right Target

(d) Contact

Figure 4.7: Sequence of Arm Positioning and Target Contact

For test cells in which the commanded display was enabled, the operator could position the commanded arm while waiting for feedback from the real arm. Figure 4.8 shows the commanded display being positioned at the target as the real arm lags behind.

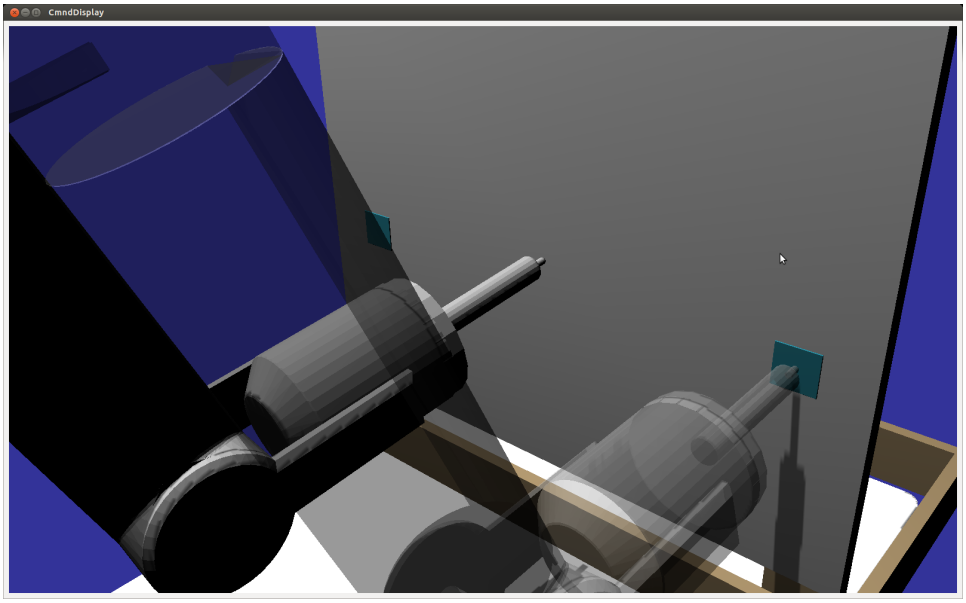


Figure 4.8: Commanded Display Positioned on a Target While the Delayed Actual Display Moves to the Commanded Position

Testing began at the lowest time delay treatment and progressed from lowest to highest. Within each time delay treatment, the subject began without an overlay and performed a test for each target configuration treatment. They then switched to the commanded display and repeated each target configuration. The target configurations were changed every each test. After each combined time delay and display method treatment, the subject filled out a NASA TLX form to assess their workload. Tests for each target configuration were considered too similar to fill out a TLX. The subject typically got through two time delay treatments in each session. Test sessions always ended after a full time delay treatment was completed. Table 4.2 gives the fully populated test matrix of the Fitts' Law Task.

Table 4.2: Fitts' Law Task Test Matrix

	Display Method	No Overlay				Commanded Display			
	Target Separation	11.0 in		25.0 in		11.0 in		25.0 in	
	Target Size	2.0 in	4.0 in	2.0 in	4.0 in	2.0 in	4.0 in	2.0 in	4.0 in
Time Delay	0.0 s	×	×	×	×	×	×	×	×
	0.33 s	×	×	×	×	×	×	×	×
	1.0 s	×	×	×	×	×	×	×	×
	2.0 s	×	×	×	×	×	×	×	×
	4.0 s	×	×	×	×	×	×	×	×
	6.0 s	×	×	×	×	×	×	×	×

4.5 Hypothesis

The experiment was designed to gauge operator performance and workload over a number a number of variables. It records the completion time of each test and the workload of each combined delay and display treatment. Previous work has shown that time delay decreases performance, and that graphical overlay techniques

can offset some of the impact. Lane demonstrated the benefits of the commanded display on a simulated robot. It was therefore hypothesized that completion time and workload would increase as the time delay increased. It was expected that using a commanded display with a real robot would be effective at mitigating the effects of time delay.

Different target configurations required different types of motion. The smaller targets require finer motion to position the tool-tip to make contact. The wider separations require a higher degree of gross motion to move between contacts. It was hypothesized that the commanded display might be more effective for different types of motion. Because the operator would need to verify final adjustments with real telemetry, it was expected that the commanded display would have greater benefits for gross positioning.

This task required operator training for each test subject. It was assumed that learning would be major factor, so training tasks were designed to bring the operator down the learning curve. As the subject completed more trials, it was expected that their task completion times would improve.

4.6 Results

The experimental results support the hypothesis that time delay decreases task performance and increases operator workload. The results also indicate that the commanded display reduces the impact due to time delay. However, the effectiveness of the commanded display was not shown to vary due to target configurations. A

summary of the data is provided in Appendix A, and the results of the analysis of variance to determine statistical significance of the results are included in Appendix B.

4.6.1 Task Completion Time

Figure 4.9 shows a comparison of the average completion time due to time delay and display method. It plots both a performance curve and a comparison with standard deviation error bars. The main effects of both time delay and display method were statistically significant to better than a $p < 0.01$ level. The interaction between display method and time delay was also statistically significant to the $p < 0.01$ level.

The plots show clear reductions in performance degradation for time delays of 2.0 seconds and higher when using the commanded display. The plots also indicate improvements over the no overlay case at all time delays, however improvement at low delay treatments was not significant. Table 4.3 summarizes the improvement and impact reduction for the high delays when using the commanded display. The Improvement column indicates the improvement as compared to the No Overlay treatment for a given time delay. The Delay Mitigation column indicates the reduction in performance degradation due to time delay. The performance degradation here is defined as the additional time it takes to complete a task, as compared to no delay. These results indicate that the commanded display eliminated nearly all performance degradation due to time delay for this task.

Table 4.3: Percentages of Performance Improvement using the Commanded Display compared to No Overlay, and Percentages of Time Delay Mitigation.

Time Delay (s)	Improvement (%)	Delay Mitigation (%)
2.0	48	96
4.0	64	96
6.0	67	93

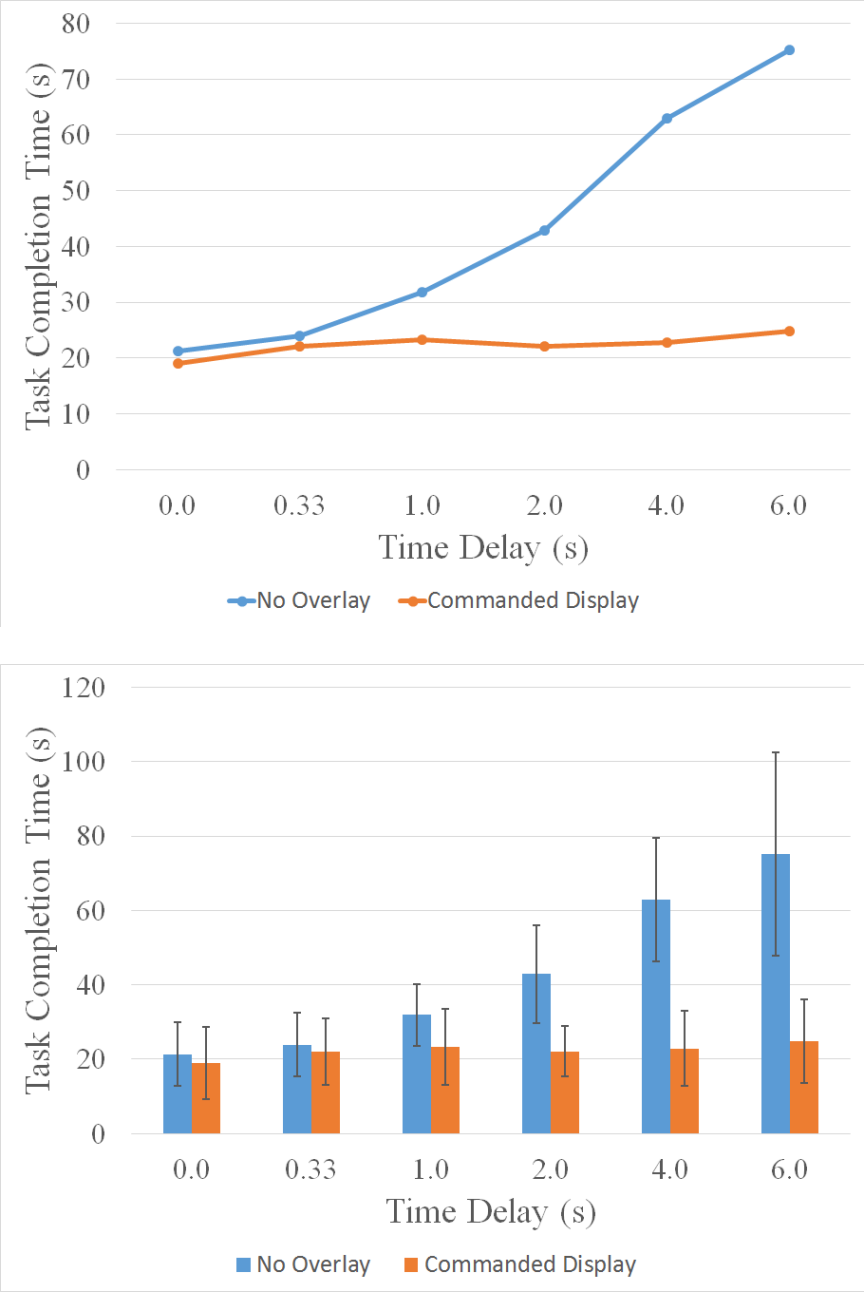


Figure 4.9: Performance(top) and Standard Error (bottom) Comparison of Time Delay by Display Method over Combined Target Separation and Size.

It had been the original intent of this work to analyze the task completion time as a function of the Index of Difficulty. However, the target size did not have a statistically significant effect. The target size was also examined for a possible interaction with the display method. However, as shown by Figure 4.10, the target size had no effect. Its main effects were not statistically significant.

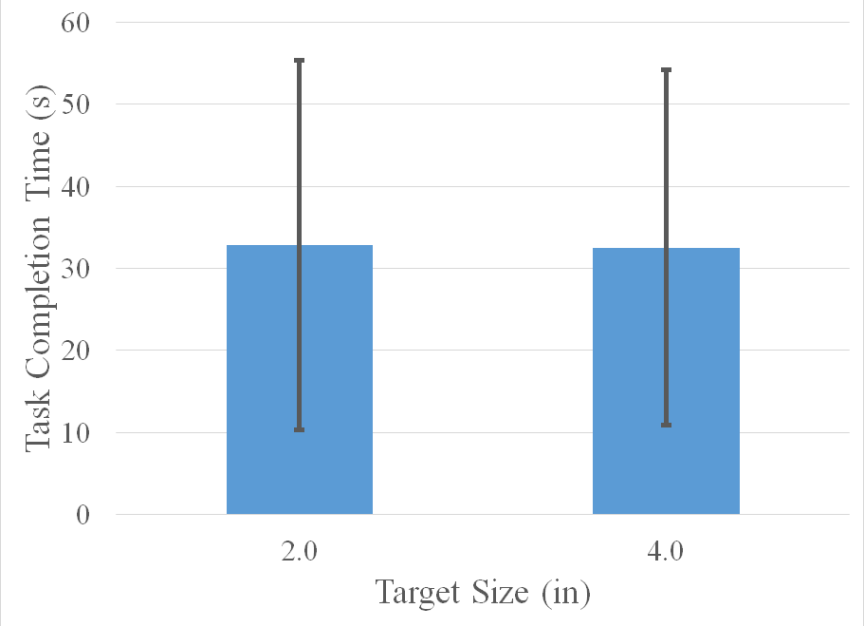


Figure 4.10: Comparison of Target Size over Combined Time Delay, Display Method, and Target Size and Separation.

The separation between targets did, however, have a statistically significant effect on completion time to the $p < 0.01$ level. The completion time due to separation is plotted with standard deviation error bars in Figure 4.11.

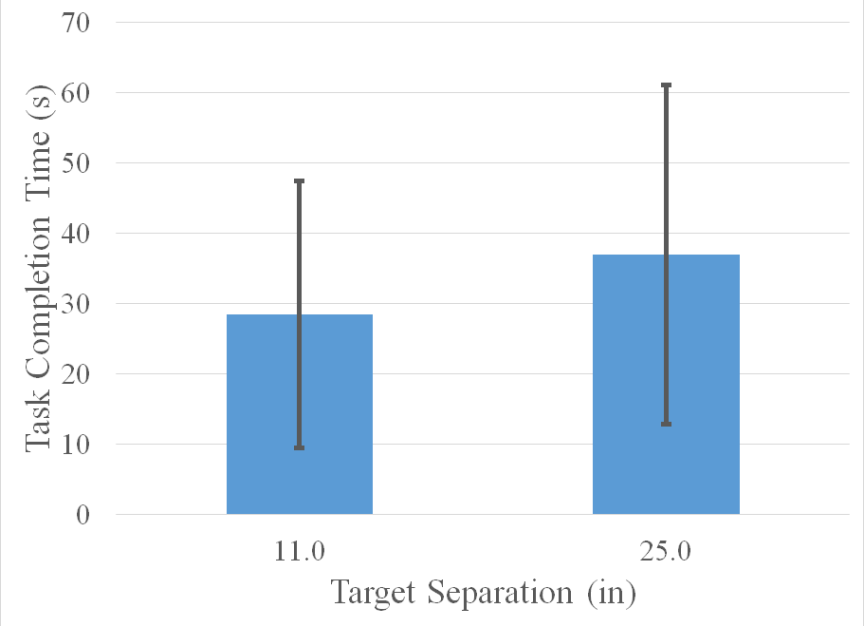


Figure 4.11: Comparison of display method over combined target configurations.

Of greater interest was a possible interaction between target separation and display method. Figure 4.12 shows the average completion time as a result of separation for each display method. No significant interaction between these variables was found. This suggests that the commanded display was equally effective for both the gross and fine movements required for this positioning task.

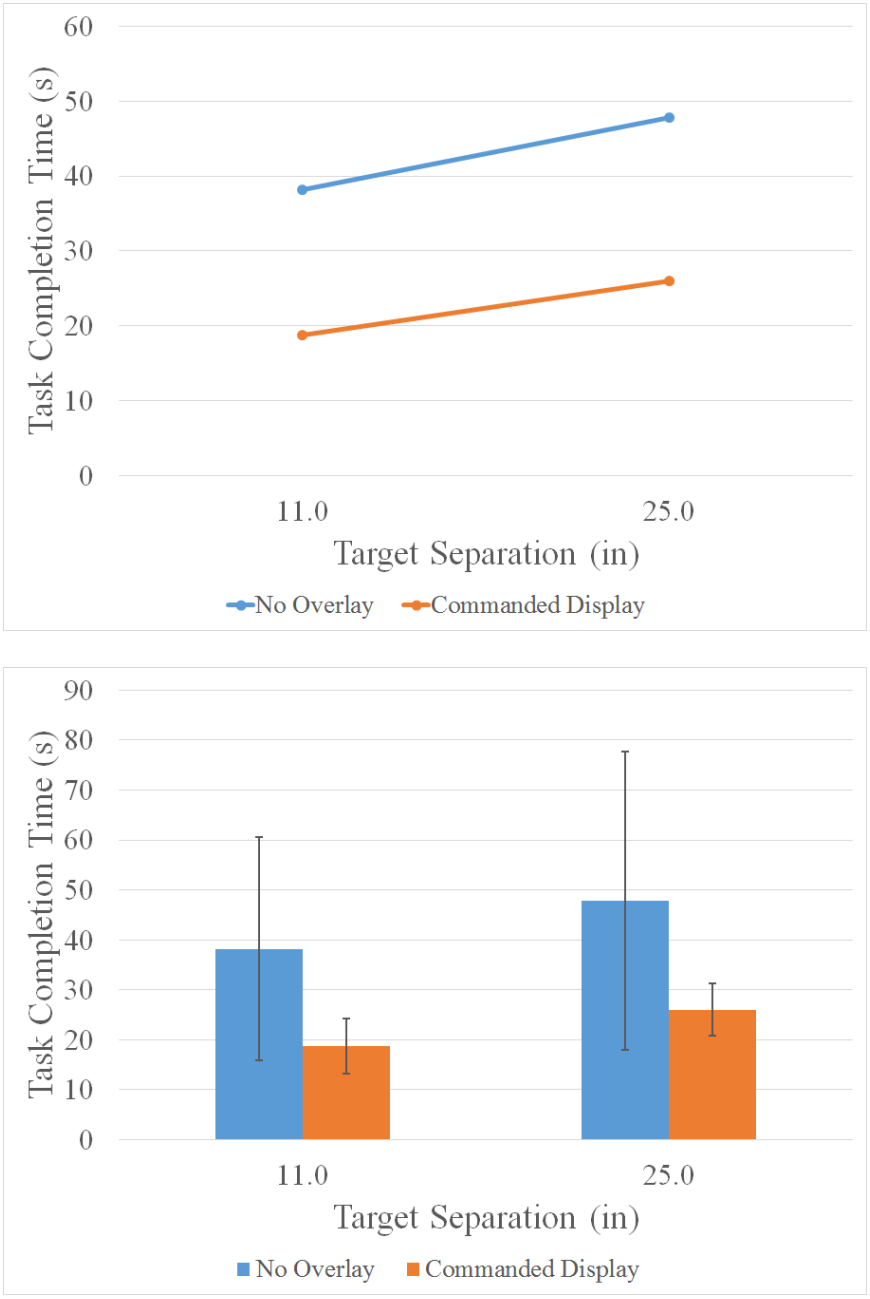


Figure 4.12: Comparison of Target Separation by Display Method over Combined Time Delay and Target Size and Separation.

4.6.2 NASA TLX

Figure 4.13 plots the average combined task load index with standard deviation bars for each time delay and display treatment. The plot indicates that workloads tends to increase as time delay increases without the commanded display. It also suggests that the commanded display reduces workload at time delay treatments of 2.0 seconds and higher. However, the main effects of time delay were not statistically significant. The display method was significant to the $p < 0.05$ level.

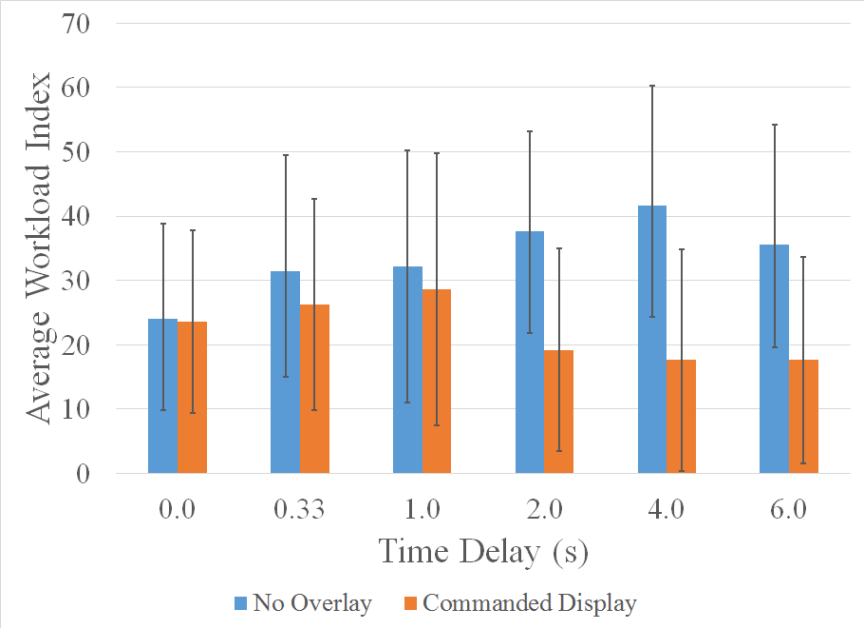


Figure 4.13: Workload Comparison of Time Delay by Display Method

Figure 4.14 plots the effect on workload due to display method for each time delay. At latencies of 2.0 seconds and higher, the commanded display appears to have resulted in a large decrease in workload, suggesting an interaction between time delay and display method. Without the commanded display, workload tended to increase as time delay increased. With the commanded display enabled, a similar rise in workload occured for time delay treatments from zero to one second. This trend may be due to the subjects relying more on the visualizer than the camera views at high delays. Several subjects reported that the commanded display was easier to use, and that they used it more after they could no longer compensate for the time delay.

Although these trends appear reasonable from the plots and consistent with subject polling, statistical significance could not be demonstrated. This may be exacerbated by the different internal scales used by each subject. This could also be due to unfamiliarity with filling out TLX forms, despite the practice during the training tasks.

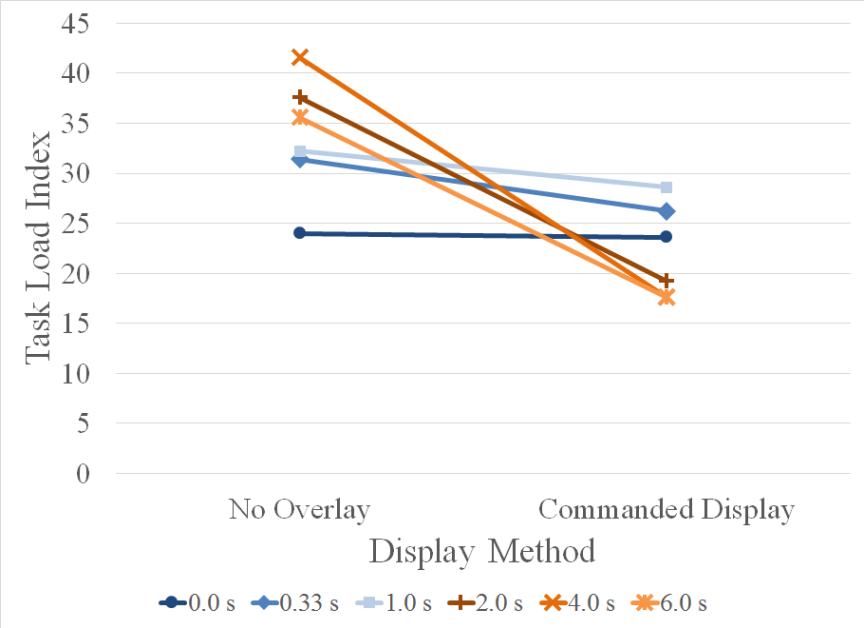


Figure 4.14: Workload comparison of Display Method by Time Delay

4.6.3 Learning Effects Results

Two brief follow-on studies were conducted to examine learning effects. Long term learning can be a major source of error in human factors testing. Each task contributes to a subject's cumulative learning. This can lead to better performance during later tasks than earlier tasks with similar treatments. In order to reduce this effect, each subject underwent training before performing the experiment. The goal of training was to move down the learning curve so that the task completion time leveled off. Additional task iterations should show little to no improvement in performance.

Task performance of one of the subjects was tracked during training. This test subject had no previous experience operating a robotic manipulator and had not yet participated in any testing. This test was conducted with the 2.0 inch targets with a 25.0 inch separation at no time delay. The commanded display was not used.The results of their first training task is shown in Figure 4.15.

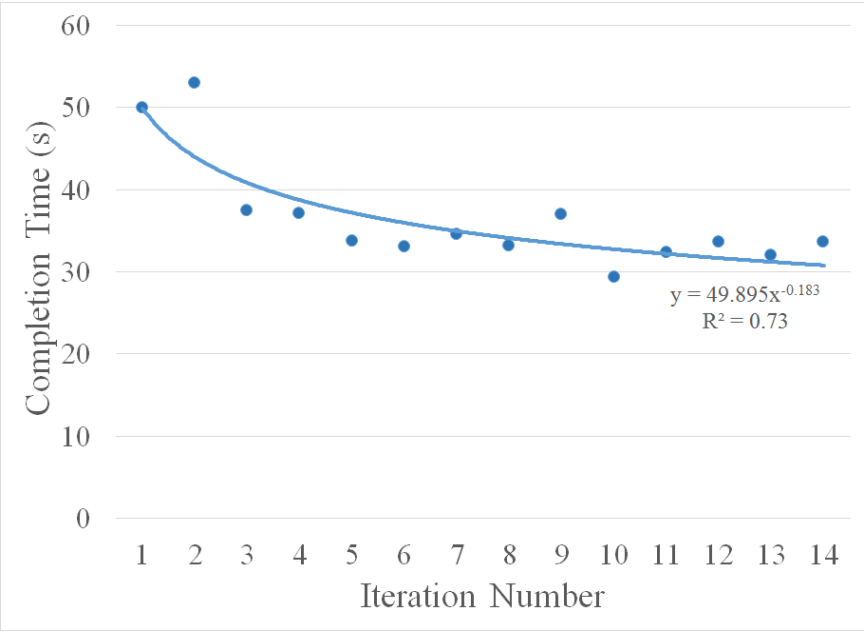


Figure 4.15: Completion Time Data and Learning Curve for One Subject During Their First Test.

Long term learning was still a concern during the experiment. A subset of the experiment was repeated with one test subject in order to assess the long-term learning effects. Table 4.4 identifies the test cells that were repeated. This study was designed to determine whether learning introduced an error that affected the previous results. Testing was conducted across all time delays and display treatments. Only a single target configuration was used: the 2.0 inch targets were used at 25.0 inch separation. Figure 4.16 shows the results, split by time delay and display method. The mean values indicate slightly better performance during the repeated test.

Table 4.4: Long-term Learning Test Matrix

	Display Method	No Overlay				Commanded Display			
	Target Separation	11.0 in		25.0 in		11.0 in		25.0 in	
	Target Size	2.0 in	4.0 in	2.0 in	4.0 in	2.0 in	4.0 in	2.0 in	4.0 in
Time Delay	0.0 s			×				×	
	0.33 s			×				×	
	1.0 s			×				×	
	2.0 s			×				×	
	4.0 s			×				×	
	6.0 s			×				×	

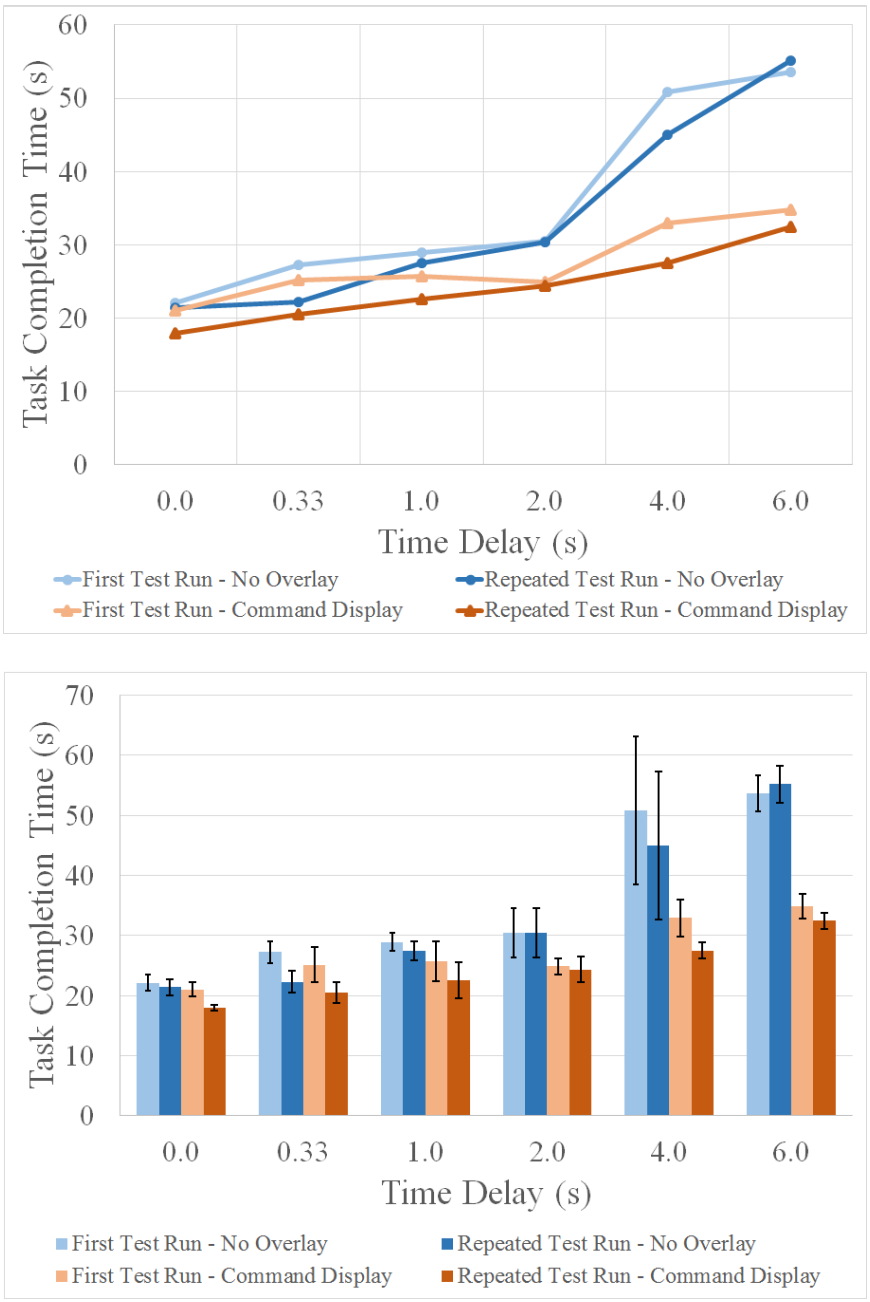


Figure 4.16: Task Completion Time Performance (top) and Standard Error (bottom) Comparisons of first and second times through a subset of the test matrix for a single subject.

4.7 Discussion

Time delay was clearly a significant factor in this experiment. The results show a strong relationship between time delay and the task completion time. As the delay increased, the task performance suffered. For the highest delay treatments, subjects had completion times several times higher than the no delay treatment. The use of the commanded display, however, nearly eliminated this performance decrease.

The study clearly demonstrates the effectiveness of the commanded delay at ameliorating time delay effects for a robot positioning task. At high delay treatments, the commanded display reduced the completion time due to delay by upwards of 90%. This strongly corroborates Lane's study, which found improvements in the range of 84-91% for a similar Fitts' Law task [1].

The results indicate a slightly larger amelioration of time delay with the commanded display than what Lane found. This result was somewhat unexpected. Rather, it was expected that testing on a physical system would introduce some performance degradation due to real-world system dynamics. In comparison, Lane's study employed a purely kinematic simulation with no system dynamics. It could be argued that this discrepancy is simply not significant, but there are several possibilities that warrant consideration.

One explanation is the task complexity. Lane's modified Fitts' Law task required movement along all three of the robot tool-tip's translational axes, where the present work only considered a two DOF task. Additionally, in this study, the subjects knew where the targets were for the duration of the task. Several subjects

would begin moving the commanded robot to the next target before delayed video feedback indicated a target hit. In Lane's Fitts' Law study, only one target was in a fixed location; the simulation generated a new target at a random location after each contact with the fixed target. The subjects' knowledge of the target location very likely allowed subjects to shave extra seconds off their completion times.

The rigorous training protocol and increased testing duration in Lane's study could be another explanation. In the present work, it was considered sufficient that the test subjects knew how to operate the robot arm and move down the initial learning curve for the experimental task. Lane's subjects performed trial runs for each test combination of test treatments. Increased learning may have better prepared subjects to deal with time delay, so that the resulting improvement with commanded display was not as pronounced.

The commanded display also shows a slight improvement in the mean completion times at the zero and low delay cases. However, this improvement is very small and was not shown to be significant in the post-hoc tests. It does not seem that, in its current implementation, the commanded display helped until higher delay treatments. Subjects reported relying primarily on video until they could no longer compensate for the delay. Only at that point did subjects begin to rely more heavily on the commanded display. A commanded display overlay on the video feed may offer increased benefit even at lower time delay cases.

The initial learning results demonstrate the importance of initial training. As expected, practice with the task lowered yielded lower completion times. As the subject hit the targets, their completion time improved until it eventually leveled

off. The initial training was meant to prevent learning effects from impacting the experimental data, but it was still necessary to examine the effects of long-term learning.

The long-term learning results suggest that learning was not a major source of error. Data from the second round of testing shows the same trends present in the first sessions. In most cases, the completion times for the second round were lower, however this improvement was small. Additionally, the improvement appears to be unrelated to either the time delay or display treatment. If long-term learning had a major effect, it would be expected to see higher levels of improvement in the first tests performed. Because of the order of testing, the improvement would be more significant as the delay delay treatment decreased. Instead, the level of improvement is relatively constant across all test cases.

The TLX results support the hypothesis that the commanded display reduces operator workload for high time delay treatments. The benefit, however, was not significant at low time delays. The surprising result is that workload actually decreased from 1.0 second to 2.0 seconds of delay when using the commanded display. These trends corroborate with reports from test subjects. Several subjects described relying mostly on the video until they could no longer compensate for the time delay. When the delay was too high, they began to rely more on the commanded display. This result is not unexpected, as it has been shown that a one second delay has been shown to be the transition region between a subject's ability to compensate and when they use a move-and-wait strategy [21] [1]. In addition to the TLX data, several subjects reported that the commanded display eliminated any frustration

they experienced due to time delay at high delays. It should be noted that statistical significance could only be demonstrated for the display treatment's main effects. However, these trends appear reasonable and are corroborated by verbal reports from several subjects, and thus are worth noting. Much of this issue stems for the various internal scales used by the subjects to fill out the TLX forms, resulting in high variances in the data.

Chapter 5: Conclusions and Future Work

5.1 Summary

This primary goal of this research was to extend the commanded display to a real robotic system. First demonstrated as a realtime teleoperation tool by Lane, the commanded display had been shown to effectively reduce task completion time for a purely simulated system. Experimental testing with the NBV-I manipulator demonstrated the effectiveness of the commanded display on a real-world system, and confirmed Lane's findings. The commanded display nearly eliminated the negative effects of time delay for a robot positioning task.

This work also demonstrated the commanded display's effectiveness at reducing operator workload. This display method significantly reduced task completion time and operator workload at time delays above 1.0 second. Unlike Lane, no significant performance improvement occurred at time delays below 1.0 second. Curiously, the overall workload from one second to two seconds decreased. This likely had to do with the implementation of the commanded display. The video feed and visualizer screens were separate, requiring test subjects to look away from the video to see the commanded display. These results may also have arisen because 1.0 second of delay tends to be in the transition region between ability to adapt and a move

and wait strategy. Several subjects reported using the video until they felt they absolutely needed the command display at the higher time delays. At that point, subjects primarily used the commanded display, and only used the video to verify task completion.

This work also resulted in the development of a visualization tool for robotic arms. The visualizer is capable of displaying a model of a robotic arms pose during teleoperation. The software is compatible with the labs DMU software, and thus can be used with most of the labs robotic arms. This work developed models for the NBV-I manipulator in order to test and demonstrate the capabilities of the system.

5.2 Future Application

The commanded display has great potential to benefit remote manipulation tasks during spacecraft servicing operations. Spacecraft communication suffers from long communication distances, and the resulting time delay hinders the ability to perform realtime teleoperation. The commanded display mitigates this issue, nearly eliminating the performance loss due to delay for robot positioning. It restores the ability of the operator to effectively perform interactive realtime teleoperation of the robot. The commanded display would also be beneficial to other remote manipulation platforms that incur delay. One example is remotely operated underwater vehicles (ROVs) performing tasks that may include underwater maintenance, surveying, and construction.

The visualizer developed for this work will see a number of uses at the Space

Systems Laboratory. Not only will it continue to support NBV-I operations, the visualizer will be extended to SAMURAI and Dymaflex arms, once appropriate models are created. As of this writing, the SSL is refitting its Ranger Telerobotic Shuttle Experiment (RTSX) arms with new motor controllers and electronics. When the refit is complete, RTSX will be run with the SSLs current DMU software. The visualizer will be used to verify DMU operation before actual robot motion.

The visualizer has a great amount of utility in operations that would otherwise require running an actual robot. With the DMU in simulation mode, an operator can control a robot virtually, without the additional overhead. This is especially useful for operator training. New operators may familiarize themselves with a robots controls and before they ever control hardware. Experienced operators may also find the visualizer useful for planning new tasks and operations.

Additionally, commanded display functionality will be integrated into the NASA Satellite Servicing Capabilities Office (SSCO) Robot Development Team systems, where it will be used to perform simulated satellite servicing tasks, such as those performed on the Robotic Refueling Mission [15]. The new commanded display simulation will employ Goddard Space Flight Centers in-house kinematics routines, and it will integrate into their existing GUI and visualization software. Initially, this functionally will be deployed and tested on a Motoman 10D industrial robot. The ultimate goal will be a full end-to-end satellite servicing mission simulation that utilizes the commanded display.

5.3 Future Studies

Future research should compare the commanded display to other predictive displays on a real world system. More sophisticated predictive displays may incorporate the systems dynamics and environment models, and may be more beneficial in some situations. Methods of combining commanded and predictive displays should also be explored.

One avenue of expansion would be the combination of a commanded display with a predictive display in a single interface. A possible benefit of a predictive display is the ability compensate for dynamic effects. To see the highest benefit of this system, the predictive display would need to have an accurate model of the robot's dynamics, while still being able to perform the computations quickly. A simple approach to combining displays would be to present a system with two options. Due to screen clutter, the operator would switch between the commanded and predicted displays depending on which was bested suited to the situation.

Another approach would be to develop a system in which a commanded robot simulation drives both the actual robot and the predictive display. Both the predictive simulation and actual simulation would be using a closed control loop to move to the same commanded position. This would prevent the predictive display's calibration from diverging due to the drifting calibration caused by small discrepancies in the computations. This commanded-predictive display would allow the operator to compensate for dynamic effects while still retaining the benefits of the commanded display.

The study presented here considered a commanded display overlay in a visualization that was independent of the actual camera views. Future study should examine the effects of having an overlay on the video feed. This will require care to be taken to ensure proper image registration between the graphical overlay and video. The main benefit of this method is that the operator would not have to continually scan between different areas on the screen to obtain the information provided by a commanded display and actual video. However, the visualizer has value in offering views that would otherwise be unattainable, and could allow an operator to move the viewpoint as the situation required. This is especially useful when the number of camera views is limited due to bandwidth constraints. Study is needed to determine how to best leverage both types of systems.

The past few years have seen dramatic advances in displays technology, leading to higher resolution screens, more sophisticated head mounted displays, and growing availability of stereoscopic displays and head-tracking systems. These technologies could be directly applied to the commanded display in order to improve the operator's situation awareness.

Appendix A: Data Summary

This appendix includes a summary of the test data for the Fitts' Law study.

A.1 Task Completion Time

Mean								
Time(s)	C1_NOCMD	C1_CMD	C2_NOCMD	C2_CMD	C3_NOCMD	C3_CMD	C4_NOCMD	C4_CMD
0	16.171	14.312	24.068	22.654	20.432	15.576	24.545	23.249
0.334	20.265	18.405	27.863	25.822	20.973	18.403	26.601	25.579
1	27.608	19.169	35.257	28.153	30.025	19.458	34.630	26.611
2	36.718	19.752	48.204	24.129	39.041	20.250	47.439	24.463
4	65.186	19.904	69.418	27.836	52.897	17.819	63.997	25.754
6	60.083	22.261	84.025	31.497	68.893	19.438	87.760	26.369

Standard Deviation								
Time(s)	C1_NOCMD	C1_CMD	C2_NOCMD	C2_CMD	C3_NOCMD	C3_CMD	C4_NOCMD	C4_CMD
0	3.404	2.644	3.182	3.175	7.188	3.579	3.334	3.210
0.334	3.697	3.327	2.867	3.417	4.865	3.677	4.118	3.630
1	6.637	4.430	4.392	7.601	4.436	5.374	3.816	4.134
2	8.047	4.653	17.986	3.272	7.331	9.455	14.922	4.864
4	24.747	4.584	25.268	3.473	17.198	5.861	19.712	4.092
6	22.935	3.804	40.446	5.502	17.695	6.113	39.279	6.559

C1 - Size: 2in Separation 11in
C2 - Size: 2in Separation 25in
C3 - Size: 4in Separation 11in
C4 - Size: 4in Separation 25in

Table A.1: Mean (top) and standard deviation(bottom) of Task Completion Time for varying Time Delay, Display Method, Target Separation, and Target Size.

A.2 NASA Task Load Index

Mean						
Display Method	Time Delay					
	0	0.334	1	2	4	6
No Overlay	24	31.4	32.2	37.6	41.6	35.6
Commanded Display	23.6	26.2	28.6	19.2	17.6	17.6

Standard Deviation						
Display Method	Time Delay					
	0	0.334	1	2	4	6
No Overlay	14.83	18.06	18.04	15.55	18.74	18.56
Commanded Display	14.24	16.45	21.14	15.79	17.26	16.09

Table A.2: Mean and standard deviation of NASA Task Load Index for varying time delay and display method.

Appendix B: Data Analysis Summary

B.1 Fitts' Law Task

B.1.1 Task Completion Time

An analysis of variance (ANOVA) was conducted for each of the tests conducted. Wherever the results indicated significance for variables with three or more treatments, Tukey and Duncan post-hoc tests were performed. This appendix contains tables of the ANOVA and post-hoc test results.

Variable	Treatments	N
Disp	CMD	120
	NO_CMD	120
TgtSep	11 in	120
	25 in	120
TgtSize	2 in	120
	4 in	120
Delay	0 s	40
	0.334 s	40
	1.0 s	40
	2.0 s	40
	4.0 s	40
	6.0 s	40

Source	Sum Squares	df	Mean Square	F	Sig.
Corrected Model	79916.665	47	1700.355	8.788	.000
Intercept	256422.171	1	256422.171	1325.254	.000
Disp	25547.569	1	25547.569	132.036	.000
TgtSep	4287.570	1	4287.570	22.159	.000
TgtSize	7.622	1	7.622	.039	.843
Delay	27283.154	5	5456.631	28.201	.000
Disp*TgtSep	82.526	1	82.526	.427	.514
Disp*TgtSize	18.399	1	18.399	.095	.758
Disp*Delay	20893.546	5	4178.709	21.597	.000
TgtSep*TgtSize	24.386	1	24.386	.126	.723
TgtSep*Delay	479.589	5	95.918	.496	.779
TgtSize*Delay	336.009	5	67.202	.347	.884
Disp*TgtSep*TgtSize	2.484	1	2.484	.013	.910
Disp*TgtSep*Delay	461.613	5	92.323	.477	.793
Disp*TgtSize*Delay	370.172	5	74.034	.383	.860
TgtSep*TgtSize*Delay	76.069	5	15.214	.079	.995
Disp*TgtSep*TgtSize*Delay	45.957	5	9.191	.048	.999
Error	37149.905	192	193.489		
Total	373488.741	240			
Corrected Total	117066.570	239			

Table B.1: General linear model table for task completion time between Time Delay, Display Method, Target Size, and Target Separation over combined test subjects.

Post-Hoc Test	Delay	N	Subset 1	2	3	4	5
Tukey HSD	0 s	40	20.12596				
	0.334 s	40	22.98891				
	1.0 s	40	27.61383	27.61383			
	2.0 s	40		32.49961			
	4.0 s	40			42.85148		
	6.0 s	40			50.04089		
	Sig.		0.159	0.619	0.195		
Duncan	0 s	40	20.12596				
	0.334 s	40	22.98891	22.98891			
	1.0 s	40		27.61383	27.61383		
	2.0 s	40			32.49961		
	4.0 s	40				42.85148	
	6.0 s	40					50.04089
	Sig.		0.358	0.139	0.118	1	1

Table B.2: Post-Hoc tests for Task Completion Time due to Time Delay over combined Display Methods, Target Separations, and Target Sizes. Means within subsets are not statistically significant.

B.1.2 NASA Task Load Index

Variable	Treatments	N
Disp	CMD	120
	NO_CMD	120
TgtSep	11 in	120
	25 in	120
TgtSize	2 in	120
	4 in	120
Delay	0s, NO_CMD	20
	0.334s, NO_CMD	20
	1.0s, NO_CMD	20
	2.0s, NO_CMD	20
	4.0s, NO_CMD	20
	6.0s, NO_CMD	20
	0s, CMD	20
	0.334s, CMD	20
	1.0s, CMD	20
	2.0s, CMD	20
	4.0s, CMD	20
	6.0s, CMD	20

Source	Sum Squares	df	Mean Square	F	Sig.
Corrected Model	79916.665a	47	1700.355	8.788	0
Intercept	256422.2	1	256422.2	1325.254	0
Sep	4287.57	1	4287.57	22.159	0
Size	7.622	1	7.622	0.039	0.843
CombDelay	73724.27	11	6702.206	34.639	0
Sep * Size	24.386	1	24.386	0.126	0.723
Sep * CombDelay	1023.727	11	93.066	0.481	0.914
Size * CombDelay	724.58	11	65.871	0.34	0.976
Sep * Size * CombDelay	124.51	11	11.319	0.058	1
Error	37149.91	192	193.489		
Total	373488.7	240			
Corrected Total	117066.6	239			

Table B.3: General linear model table for task completion time between combined Time Delay and Display Method, Target Size, and Target Separation over combined test subjects.

Post-Hoc Test	Delay	N	Subset 1	2	3	4	5
Tukey HSD	0s CMD	20	18.94799				
	0s NO_CMD	20	21.30394				
	0.334s CMD	20	22.05241				
	2s CMD	20	22.14862				
	4s CMD	20	22.82818				
	1s CMD	20	23.34766				
	0.334s NO_CMD	20	23.92542				
	6s CMD	20	24.89159				
	1s NO_CMD	20	31.88001	31.88001			
	2s NO_CMD	20		42.85059			
	4s NO_CMD	20			62.87477		
	6s NO_CMD	20			75.1902		
	Sig.		0.136	0.35	0.189		
Duncan	0s CMD	20	18.94799				
	0s NO_CMD	20	21.30394				
	0.334s CMD	20	22.05241	22.05241			
	2s CMD	20	22.14862	22.14862			
	4s CMD	20	22.82818	22.82818			
	1s CMD	20	23.34766	23.34766			
	0.334s NO_CMD	20	23.92542	23.92542			
	6s CMD	20	24.89159	24.89159			
	1s NO_CMD	20		31.88001			
	2s NO_CMD	20			42.85059		
	4s NO_CMD	20				62.87477	
	6s NO_CMD	20					75.1902
	Sig.		0.259	0.054	1	1	1

Table B.4: Post-Hoc tests for Task Completion Time due to Time Delay over combined Display Methods, Target Separations, and Target Sizes. Means within subsets are not statistically significant.

Variable	Treatment	N
Display	CMD	30
	NO_CMD	30
Delay_s	.000 s	10
	.334 s	10
	1.000 s	10
	2.000 s	10
	4.000 s	10
	6.000 s	10

Source	Sum Squares	df	Mean Square	F	Sig.
Corrected Model	3483.733a	11	316.703	.860	.584
Intercept	46816.267	1	46816.267	127.117	.000
Display	2018.400	1	2018.400	5.480	.023
Delay_s	286.933	5	57.387	.156	.977
Display * Delay_s	1178.400	5	235.680	.640	.670
Error	17678.000	48	368.292		
Total	67978.000	60			
Corrected Total	21161.733	59			

Table B.5: General Linear Model table for NASA Task Load Index between Time Delay and Display Method over combined Subjects.

B.2 Long Term Learning Error

Variable	Treatment	N
Repetition	1	12
	2	12

Source	Sum Squares	df	Mean Square	F	Sig.
Corrected Model	39.157	1	39.157	.340	.566
Intercept	21868.38	1	21868.308	189.797	.000
Repetition	39.157	1	2018.400	.340	.566
Error	2534.822	22	115.219		
Total	24442.287	24			
Corrected Total	2573.979	23			

Table B.6: ANOVA table for Task Completion Time between Repetitions over combined Time Delays and Displays.

Appendix C: Institutional Review Board Consent Form

This appendix contains the approved consent form, with identifying information removed. This was the form used to obtain informed consent from each test subject. It includes information about the study, the risks involved, protection of identifying information, the right to withdraw at any time, and a statement of consent.

University of Maryland College Park

Initials _____ *Date* _____

Project Title	Human Factors Evaluation of Operator Interfaces for Teleoperation of a Dexterous Manipulator
Purpose of the Study	This research is being conducted by Dr. David Akin and Kevin Davis at the University of Maryland, College Park. We are inviting you to participate in this research project because you are a healthy adult. The purpose of this research project is to evaluate human factors attributes of a telerobotic workstation for a dexterous manipulator. Robot operators will use a variety of display techniques and input devices to control a dexterous satellite servicing robot with time delay. The goal of this research is to decrease operator workload and task completion time, as well as identify and minimize bandwidth requirements between a workstation and robot.
Procedures	The procedures involve remote operation of a robotic manipulator to perform a simulated satellite servicing task. The researchers will first explain the task. You will sit at a workstation with a desktop computer and set of hand controllers. The researchers will configure the workstation for a test run and explain the workstation configuration and interfaces. The test run will begin, and you will perform the same task a set number of times. After the test run is complete, you will be given a break, and the researchers will reconfigure the workstation for another test run. You will perform a set number of test runs during each test session. You will be asked to participate in three test sessions over a period of several months. Each test session should last approximately two hours. The research will be conducted at the Space Systems Laboratory at the University of Maryland, College Park. Lab facilities are located in room 1309 of the Jeong H. Kim Engineering Building and at the Neutral Buoyancy Research Facility. You will be given a survey with questions relevant to using a computer workstation. For example: Are you left or right handed? (Pick one) Left,

	Right, Ambidextrous
Potential Risks and Discomforts	There may be some risk of discomfort from participating in this research study. The tasks you perform will likely increase your mental workload, which may lead to feelings of fatigue. Workload levels are similar to playing a video game.
Potential Benefits	There are no direct benefits from participating in this research. We hope that, in the future, other people might benefit from this study through improved understanding of techniques to improve task performance during teleoperation. We also hope to better define the bandwidth requirements that impact the design of satellite servicing spacecraft.
Confidentiality	Any potential loss of confidentiality will be minimized by storing data in a locked office or on a password protected computer. Your name will not be included on the surveys and other collected data; a code will be placed on the surveys and other collected data; through the use of an identification key, the researcher will be able to link our survey to your identity; and only the researchers will have access to the identification key. If we write a report or article about this research project, your identity will be protected to the maximum extent possible. Your information may be shared with representatives of the University of Maryland, College Park or governmental authorities if you or someone else is in danger or if we are required to do so by law.
Right to Withdraw and Questions	Your participation in this research is completely voluntary. You may choose not to take part at all. If you decide to participate in this research, you may stop participating at any time. If you decide not to participate in this study or if you stop participating at any time, you will not be penalized or lose any benefits to which you otherwise qualify. If you decide to stop taking part in the study, if you have questions, concerns, or complaints, or if you need to report an injury related to the research, please contact the investigator: **REDACTED**

Participant Rights	If you have questions about your rights as a research participant or wish to report a research-related injury, please contact: **REDACTED** This research has been reviewed according to the University of Maryland, College Park IRB procedures for research involving human subjects.	
Statement of Consent	Your signature indicates that you are at least 18 years of age; you have read this consent form or have had it read to you; your questions have been answered to your satisfaction and you voluntarily agree to participate in this research study. You will receive a copy of this signed consent form. If you agree to participate, please sign your name below.	
Signature and Date	**NAME OF PARTICIPANT** **[Please Print]**	
Signature and Date	**SIGNATURE OF PARTICIPANT**	
	DATE	

Bibliography

[1] J.C. Lane, *Human Factors Optimization of Virtual Environment Attributes for a Space Telerobotic Control Station*, Ph.D. Thesis, University of Maryland, MD (2000).

[2] J.C. Lane, C.R. Carignan, D.L. Akin, "Advanced Operator Interface Design for Complex Space Telerobots," Autonomous Robots **11**, 1:49-58 (2001).

[3] NASA Goddard Spaceflight Center, "On-Orbit Satellite Servicing Study Project Report," (October 2010).

[4] "History | The Orocos Project." Internet: http://www.orocos.org/content/history/, [March 19, 2014]

[5] N.J. D'Amore, *Development of a Reusable Top-Level Control Architecture for a Robotic Manipulator*, M.S. Thesis, University of Maryland, MD (2010).

[6] A. Ellsberry, *Development and Evaluation of a Flexible Distributed Robot Control Architecture*, M.S. Thesis, University of Maryland, MD (2010).

[7] "netem | The Linux Foundation." Internet: http://www.linxfoundation.org/collaborate/workgroups/networking/netem/, [April 16, 2014].

[8] "VideoLAN | VideoLAN Organization." Internet: http://www.videolan.org/videolan/, [April 16, 2014].

[9] "VideoLAN | VideoLAN Organization." Internet: http://www.videolan.org/vlc/features.html, [April 16, 2014].

[10] "About | OpenCV." Internet: http://opencv.org/about.html, [April 16, 2014].

[11] Fitts, P. M., "The information capacity of the human motor system in controlling the amplitude of movement," Journal of Experimental Psychology, 47, 381391 (1954).

[12] Craig, John J., "Introduction to Robotics: Mechanics and Control." 3rd ed., Upper Saddle, New Jersey: 2005.

[13] Rayman, R., Croome, K., Galbraith, N. McClure, R., Morady, R., Peterson, ..., Primak, S., "Robotic telesurgery: a real-world comparison of ground- and satellite-based Internet performance," The International Journal of Medical Robotics and Computer Assisted Surgery, 3: 111116 (2007)

[14] Imaida, T. Yokokohji, Y., Doi, T., Oda, M., Yoshikawa, T., "'Ground-Space Bilateral Teleoperation of ETS-VII Robot Arm by Direct Bilateral Coupling Under 7-s Time Delay Conditions," *IEEE Transactions on Robotics and Automation*, vol.20, no.3, 2004, pp499-511.

[15] "Tasks | Robotic Refueling Mission." Internet: http://ssco.gsfc.nasa.gov/rrm_tasks.html, [April 19, 2014].

[16] "Features | OpenSceneGraph." Internet: http://www.openscenegraph.org/index.php/about/features, [April 19, 2014].

[17] Wang, R., Qian, X., "OpenSceneGraph 3.0: Beginniner's Guide." Packt Publishing, Birmingham, UK: (2010)

[18] "Qt Project." Internet: http://qt-project.org/ [April 11, 2014].

[19] "Signals and Slots | Qt Project." Internet: http://qt-project.org/doc/qt-4.8/signalsandslots.html, [April 11, 2014].

[20] Conway, L., Volz, R., Walker, M., "Tele-Autonomous Systems: Methods and Architectures for Intermingling Autonomous and Telerobotic Technology", *Proceedings of the IEEE International Conference on Robotics and Automation*, 4:1121-1130 (1987)

[21] Sheridan, T.B., "Telerobotics, Automation, and Human Supervisory Control." Cambridge: MIT Press, 1992.

[22] Sheridan, T.B., "Space Teleoperation Through Time Delay: Review and Prognosis," *IEEE Transactions on Robotics and Automation*, vol 9, no. 5, 1993, pp 592-606.

[23] Sheridan, T.B, Ferrel, W.R., "Remote Manipulative Control with Transmission Delay," *IEEE Transactions on Human Factors in Electronics*, 6, 2432 (1963)

[24] Mar, L.E., "Human Control Performance in Operation of a Time-Delayed Master-Slave Telemanipulator." M.S. Thesis, MIT, MA (1985).

[25] Thompson, D.A., "The Development of a Six Degree of Freedom Robot Evaluation Test," Proceedings of 13th Annual Conference Manual Control, MIT, Cambridge, MA (1977).

[26] Noyes, M., "Superposition of graphics on low bit rate video as an Aid in Teleoperation." M.S. Thesis, MIT, MA (1982).

[27] Ferrel, W.R., Sheridan, T.B, "Remote Manipulation with Transmission Delay," *IEEE Transactions on Human Factors in Electronics*, 6, 2529 (1965)

[28] Held, R., Efstathiou, A., Greene, M., "Adaptation to Displaced and Delayed Visual Feedback from the Hand," *Journal of Experimental Psychology* vol. 2, no. 6, 1966, pp 887-891.

[29] Conway, L. Volz, R., Walker, M., "Tele-Autonomous Systems: Methods and Architectures for Intermingling Autonomous and Telerobotic Technology." Proceedings of the IEEE International Conference on Robotics and Autonmation, 6(9). 1990.

[30] Funda, J., "Human Factors Optimization of Virtual Environment Attributes for a Space Telerobotic Control Station," Ph.D. Thesis, University of Pennsylvania, PA (1991).

[31] Backes, P.G., "Supervised Autonomy for Space Telerobotics." Jet Propulsion Laboratory, CA, (1993).

[32] Ziebolz, H., Paynter, H.M., "Possibilities of a Two-Time Scale Computing System for Control and Simulation of Dynamic Systems," Proceedings of the National Electronics Conference, 9. (1953).