1-1-2012

Development Of A Graphical User Interface For Control Of A Robotic Manipulatior With Sample Acquisition Capability

Karan Desai
Ryerson University

DEVELOPMENT OF A GRAPHICAL USER INTERFACE FOR CONTROL OF A ROBOTIC MANIPULATOR WITH SAMPLE ACQUISITION CAPABILITY

By

Karan Desai, B.Eng
Aerospace Engineering,
Ryerson University, 2010

A thesis presented to Ryerson University

in partial fulfillment of the
requirements for the degree of
Masters of Applied Science
in the Program of
Aerospace Engineering

Toronto, Ontario, Canada, 2012

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Development of a Graphical User Interface for Control of a Robotic Manipulator with Sample Acquisition Capability

Karan Desai

Masters of Applied Science, Ryerson University, Toronto, 2012

Abstract

Design of a graphical user interface (GUI) is a delicate task requiring knowledge of human cognitive behaviour, design strategies and programming skills. In this thesis work, a GUI has been developed for control of a robotic arm that is capable of sample retrieval and collection. This thesis work creates a bridge between technical and psychological aspects of interface design by integrating the concepts of compatibility of GUI with users, consistency in design, visual hierarchy and page layout.

The developed GUI is able to support control of the robotic manipulator autonomously and manual operation using a joystick. Combinations of control functions have been defined and implemented to alleviate the operator's efforts. The developed GUI is capable of task planning in offline mode. Implemented intelligent server/client architecture enables efficient remote control of the robotic arm. The presented interface can also be used for multiple systems with minimal changes. To verify the effectiveness of the developed GUI, experiments have been conducted using a robotic arm comprised of three rotary joints and a scoop.

Acknowledgements

I would like to thank God for giving me the endurance and perseverance to complete this work.

I would also like to express my deepest gratitude to Dr. Guangjun Liu, who has supported me throughout the course of my academic career through encouragement, sound advice, good teaching, good company, and plenty of great initiatives.

I also wish to express my deepest gratitude to my family whose countless sacrifices have facilitated the completion of the thesis.

I would like to thank all of my colleagues, especially Dr. Yugang Liu, for providing well-needed guidance. It is a pleasure to thank my friends for their support during the preparation of this thesis. I am very grateful to everyone who edited and reviewed my work. This thesis would not have been possible without their help and support.

Karan Desai

Table of Contents

List of Tables

List of Figures

Nomenclature

AID – Astronaut Interface Device

CLI – Command Line Interfaces

CNC – Computer Numerical Control

DH – Denavit – Hartenberg

DOF – Degree of Freedom

GUI – Graphical User Interfaces

GUIDE – Graphical User Interface Development

HMI – Human Machine Interaction

IP – Internet Protocol

MER – Mars Exploration Rovers

OS – Operating System

RoSE – Rover Sequence Editor

ROV – Remotely Operated Vehicles

RSVP – The Rover Sequencing and Visualization Program

SCARA – Selective Compliance Assembly Robot Arm

SCL – Systems and Control Laboratory

TCP – Transmission Control Protocol

UAV – Unmanned Arial Vehicles

UDP – User Datagram Protocol

Chapter 1: Introduction

1.1 Introduction

Robotic arms have proven to be indispensable within industrial factories for lifting heavy objects, moving with high speeds and repeating complex performance with unerring precision [13]. Recently, robotic arms have also demonstrated enormous application potential in planetary exploration [20]. One of the most significant planetary missions involves sample collection for analysis and possible return to earth [20]. This thesis presents a graphical user interface (GUI) to address the human-robot interaction issue associated with control of a robotic manipulator that is capable of sample acquisition.

With the advancement of robotics technology, robots are getting closer to human beings and the human-robot interaction has become an important concern. As one of the most critical elements, human-robot interface determines the quality of interactions between human and robots [28]. While developing human-robot interfaces, the developers need to be aware of the technical knowledge of the end users. Furthermore, compatibilities of the designed interface with the robot and the operator are crucial and should get enough attention. The evolution of the user interface design of the device for personal and entertainment use has experienced a giant leap. For example, the revolutionary user interfaces of *Apple*'s *iPhone* have successfully integrated the latest technology with specialized software [30]. In order to take the user interface design of robotic manipulators to the same level, substantial research is yet needed [16].

1.2 A Brief History of the Human-Computer Interface

The need for people to communicate with each other has existed ever since humans first walked upon this planet. The lower and most common level of communication modes shared by humans are movements and gestures. Movements and gestures are language independent. The next higher level of communication mode is spoken language. Most people can speak one language, some two or more. The third level is the written language. While most people can speak, not all can write. Through its first decades, a computer's ability to deal with human communication was inversely related to what was easy for people to do. The computer demanded rigid, typed input through a keyboard (command line interface). Most people responded slowly using this device. Finally, in the 1970s research at the Xerox's Palo Alto Research Center found an alternative to command line interface. The Xerox system, Altus and STAR introduced the mouse, pointing, and selecting as the primary human-computer communication method, which was later popularized by Apple [6].

1.3 Graphical User Interface (GUI)

The human-robot interaction is difficult mostly because the human must be familiar with the detailed software and hardware architecture of the system. A user interface design is often significantly based on the technical properties of the system rather than the operators and the objectives of the system. This issue, along with inadequately designed GUI, leads to disappointed users and interfaces that are complicated.

A *user interface* is a collection of techniques and mechanisms to interact with a system. In a *graphical* interface, the primary interaction mechanism is a pointing device of some kind. This device is electronic equivalent to human

hand [19]. Shown in Figure 1.1 is the simplest explanation of the interaction between human and computer using user interface.

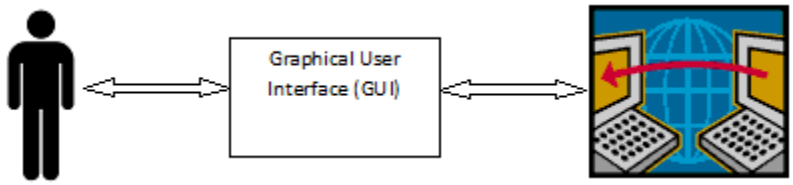

Figure 1.1: Interaction of user interface between user and computer

Most individuals have used GUI in one form or the other, for example: Windows or Apple operating systems, cell phone operating systems such as Android and iOS.

Tele-operated systems require intranet connection to transmit and receive data. Figure 1.2 shows a block diagram of a tele-operated system. The user selects the input using a keyboard, mouse or touch screen. Once the input is defined, the system goes through a hierarchy of control sequences and performs the action defined by the user. As soon as the action is completed, the user interface gives a visual or auditory indication to the user.

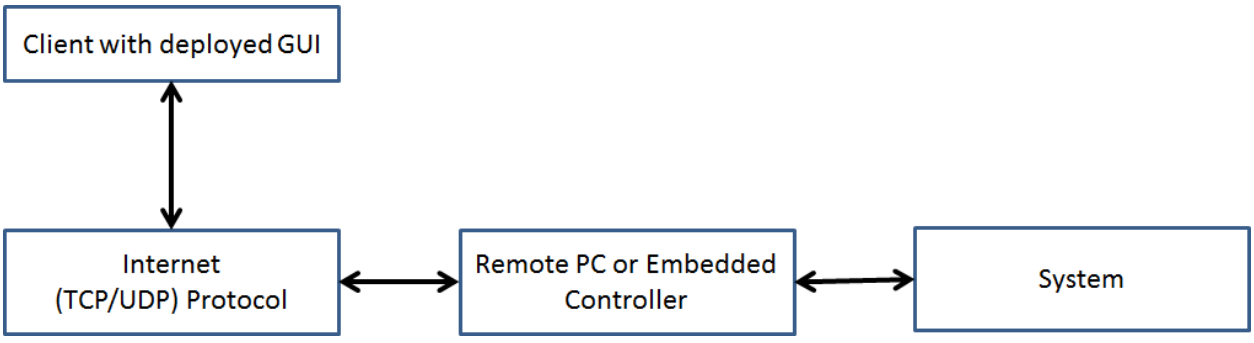

Figure 1.2: Block diagram of remotely controlled system

Human Machine Interface (HMI) is the interface where interaction between human and machine occurs. The goal of the interaction is to operate the machines effectively and feedback from the machines helps the operator in making correct decisions.

A *graphical user interface* (GUI) is a *human-computer interface* (i.e., a way for humans to interact with computers) that uses *windows*, *icons* and *menus* that can be manipulated by a mouse. A major advantage of a graphical user interface is that they make an operation more intuitive, thus are easier to learn and use. GUI is the most common form of user interface used in current industry for operation of robots.

Graphical user interface was not very popular until recently, one of the reasons is that the GUI requires considerably high computing power that was not cheap until recently. Now, GUI has become the new norm when it comes to interacting with machines. From the user's point of view, an interface is a software that allows them to perform tasks in their own way. Hence, it must be easy to learn and use.

Command Line interfaces: *command line interfaces* (CLIs), use only text and are accessed solely by a keyboard. The most familiar example of a CLI is MS-DOS.

There are many different types of HMIs, depending on the purpose of the system, the work environment and the cost. The most widely used are graphical user interface and command line interface. Other than GUI and CLI, below is a list and brief description of HMIs used for mobile and industry robots.

1.3.1 Touch User Interface

This is a type of GUI that uses touchscreen or touchpad for the input and output of the device. Feedback is given by haptic-feedback method to confirm the entry of a command. This type of user interface is commonly used in smartphones, photocopiers, or systems that do not require very precise inputs.

1.3.2 Gesture Interface

This type of user interface is new and research is underway to make it user friendly and effective. A gesture interface accepts input in-form of hand gestures and mouse gestures. Gesture interface is very helpful for people with disabilities because it recognizes gestures used for everyday interactions.

1.3.3 Voice Interface

Commonly known as speech recognition technology, voice interface accepts input and generates outputs by creating voice prompts. The user input is made by "speaking" to the machine. This type of interface is very effective, as it does not require physical interaction with the system. It is a relatively new field and more research is required because the diverse range of speech accents from humans all around the world make it harder for the machine to recognize the input.

Designing such interfaces is a challenge and it requires a great deal of work to make the interface logical, functional, accessible and pleasant to use. Some engineers specialize in developing human machine interfaces and changing the ways in which people interact with machines. Poorly designed graphical user interface can be very frustrating for the operator, resulting in errors. When industry robots have poorly designed user interface, errors are bound to occur, and the result of this could be catastrophic.

A well-designed interface can eliminate the need of learning complex theories, remembering commands and implementing complicated algorithms. All of the complex work is done behind the interface by the developers so that the users are provided with pleasant experience.

In order to make the final product meet the user's expectations and requirements, three necessary phases constitute the process of intelligent interface design. They are Analysis, Design and Construction, as shown in Figure 1.3.

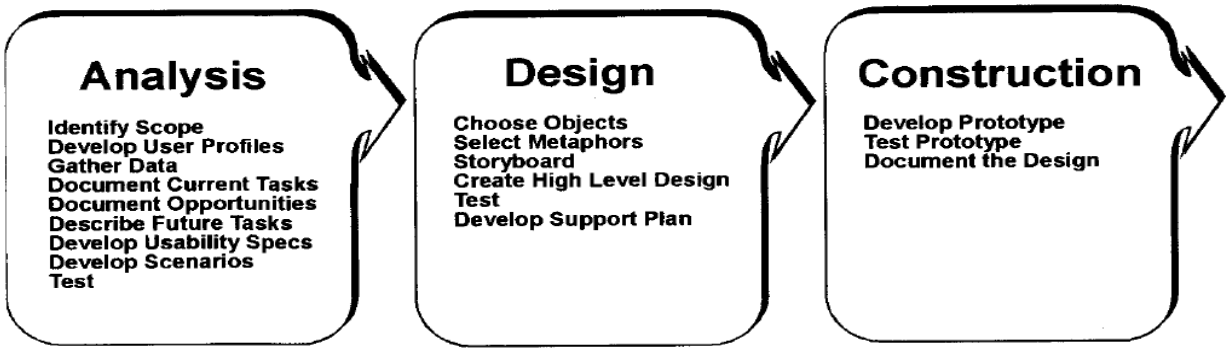

Figure 1.3: The phases of intelligent interface design [31]

1.4 Thesis Objective and Contributions

The objective of this research is to investigate principles of graphical user interface design and develop a graphical user interface for control of a robotic arm that is capable of sample retrieval and collection. Based on a thorough literature investigation, a GUI that can support autonomous and manual control of the robotic manipulator is developed. The developed interface is intuitive, user friendly, easy and enjoyable to navigate. In order to alleviate commonly used functions such as scooping, resetting joints and homing joints, shortcuts are developed and implemented.

Unexpected events such as connection failure may send robotic manipulator in uncontrolled motion. The developed interface operates the robotic manipulator using an internet connection; hence, it is necessary to develop an intelligent server/client architecture that is capable of handling unexpected events. Lastly, the developed user interface must be able to operate robotic manipulator manually using joystick and schedule multiple

tasks for later execution. It should also be able to operate other robotic manipulators with minimal changes and be operating system independent.

1.5 Thesis Outline

Chapter 2: Literature Review and Background

This chapter includes review of literatures related to this thesis work. Furthermore, integration of graphical user interface and sensors is explained.

Chapter 3: Kinematic Analysis and Hardware

This chapter explains basic concepts of robot kinematics. A forward and inverse kinematic analysis is derived. Lastly, hardware system used during the experiment is presented.

Chapter 4: Graphical User Interface Design Principles

In this chapter, guidelines of effective graphical user interface design are presented. These guidelines have been used throughout the designed interface.

Chapter 5: SCL – Robotic Mast Graphical User Interface

This chapter explains the developed and implemented "SCL – Robotic Mast Graphical User Interface."

Chapter 6: Conclusion

This chapter concludes the thesis by summarizing the developed user interface and discussing possible future work.

Chapter 2: Literature Review and Background

2.1 Related Work

Extensive efforts have been reported on the design and development of the graphical user interface. Remote operation and manipulation of robots has previously been implemented to perform pre-set tasks, often in unsafe, inaccessible and remote environments [2]. Igarashi *et al.* presented how individuals perceive information and established an approach to develop a GUI that efficiently presents critical information [8]. In tele-operation research, one way of presenting feedback is by using forced haptic device [3]. Anderson and Spong presented a way to make the operator aware of remote conditions using forced feedback [1]. Vajnberger *et al.* developed a GUI using visual feedback to operate a five-DOF robot arm, which was implemented in MATLAB, making it platform specific and dependent, i.e., the software had to be installed to use the interface [27]. Universal web interfaces for robot control allows multiple users to control a robot over the web [14], which is platform independent, allowing connectivity to different frameworks. Goldstain *et al.* did a study on the systematic determination of the interface design for web-based robot teaching [7].

2.2 Astronaut Interface Device GUI

Since it is not suitable for the suited astronauts to use mouse and a keyboard to control the robots, a GUI called AID is especially designed to control the robots by suited astronauts. In order to increase readability, the AID display program is set up in window manner, where multiple display screens are viewed one at a time using selection buttons. Once the robot is selected by the operator, all the commands are sent to that specific robot. "The current display page for AID includes an overhead "radar" view showing

the locations of all robots, a procedure tracking page, a predictive consumables gauge, an image display page and a robot movement control page. [10]" Because the AID uses standard windows software, all of the AID GUI applications can be developed using a standard Windows XP computer. The AID GUI is divided in different screens, thus, confusion is avoided while reading the information as each sub-display will give information about specific entities.

2.2.1 AIDMap GUI

The AIDMap GUI graphically displays the location of all the robots and stable locations (such as waypoints, craters etc.). Objects are defined as dots with their name on the side if available. If the displayed object is a robot and has a valid heading, then the heading indicator is also displayed, shown in Figure 2.1. Center of the map can be a robot or a fixed location, and is selected by the astronauts. AIDMap GUI offers two options in a map indicator - heading up map and north up map. The Heading Up map mode is found more useful when the astronauts are moving. As robots move, their position is updated on the map. Once the target object is selected, distance and bearing is displayed as texts in GUI.

Figure 2.1: AIDMap GUI: Display of robots and known locations [10]

2.2.2 Task Checklist GUI

This part of the GUI displays task to be completed along with the estimated time required to complete the task. A task item automatically expands when it is the current task and collapses when it is not.

2.2.3 Consumables GUI

The gauge display (right side of Figure 2.1), shows the present and future status of the consumable for the agent. If an agent has an active job, amount of needed consumables for the job is shown by the green bar while the white bar shows the estimated supplies needed for the all the future jobs. There are two configurable gauge limits, soft and hard, which by default are 20% and 10%, respectively.

2.2.4 Camera GUI

The Camera GUI shows an image from the selected cameras. This provides the user with the ability to view items of interest encountered by the agent and improves situational awareness for commanding robots. Using a dropdown menu, users can select a camera from the list of available cameras. The Pan/Tilt button is activated if the selected camera has such feature.

2.2.5 Robot Move GUIs

The AID is designed to work with multiple robots; this makes it difficult for the developer to design a GUI such that each robot can be controlled with one GUI. In order to maintain the simplicity of the GUI, each robot is responsible for interpreting the command. This may lead to breaking down a

single command in multiple movements or ignoring the commands all together. In order to make the input easier, robot's movement by the GUI is separated in two different windows. The first interface allows a user to enter simple command such as drive, turn or rotate in place. The commands can be sent by simply tapping on the screen. Second interface allows input of more complicated commands.

The AID has successfully shown that it is possible to control multiple robots using a single application. Since the AID uses common GUI for multiple robots, it saves time and money for both training agents and the development of GUIs. Design of the AID GUI is very interesting as it shows sub-display for each activity while showing important information (such as consumable) all the time. Simplicity of the GUI makes it user friendly and effective.

This particular study served as the base reference for implementing and developing the GUI presented in this thesis. When it comes to space applications, redundancy is very important. The AID GUI lacks this feature. Though one can control and command each robot, there is no way of confirming if the sent command is correct. It was also concluded that there should be a method to review all the commands sent to the system. Having such a feature will provide easy access to the command history in the case of a failure.

2.3 Graphical User Interface for Remote Control of a Robotic Arm

The graphical user interface shown in Figure 2.2 was developed by Vajnberger *et al* to remotely operate a five DOF robotic arm. The Robotic arm controlled by the interface does not have sensors built in to report the position of the arm. This interface has an incredible visual feedback provided

to users beside the control panel. The setup of controls and feedback makes it an ideal user interface to operate the robotic arm remotely.

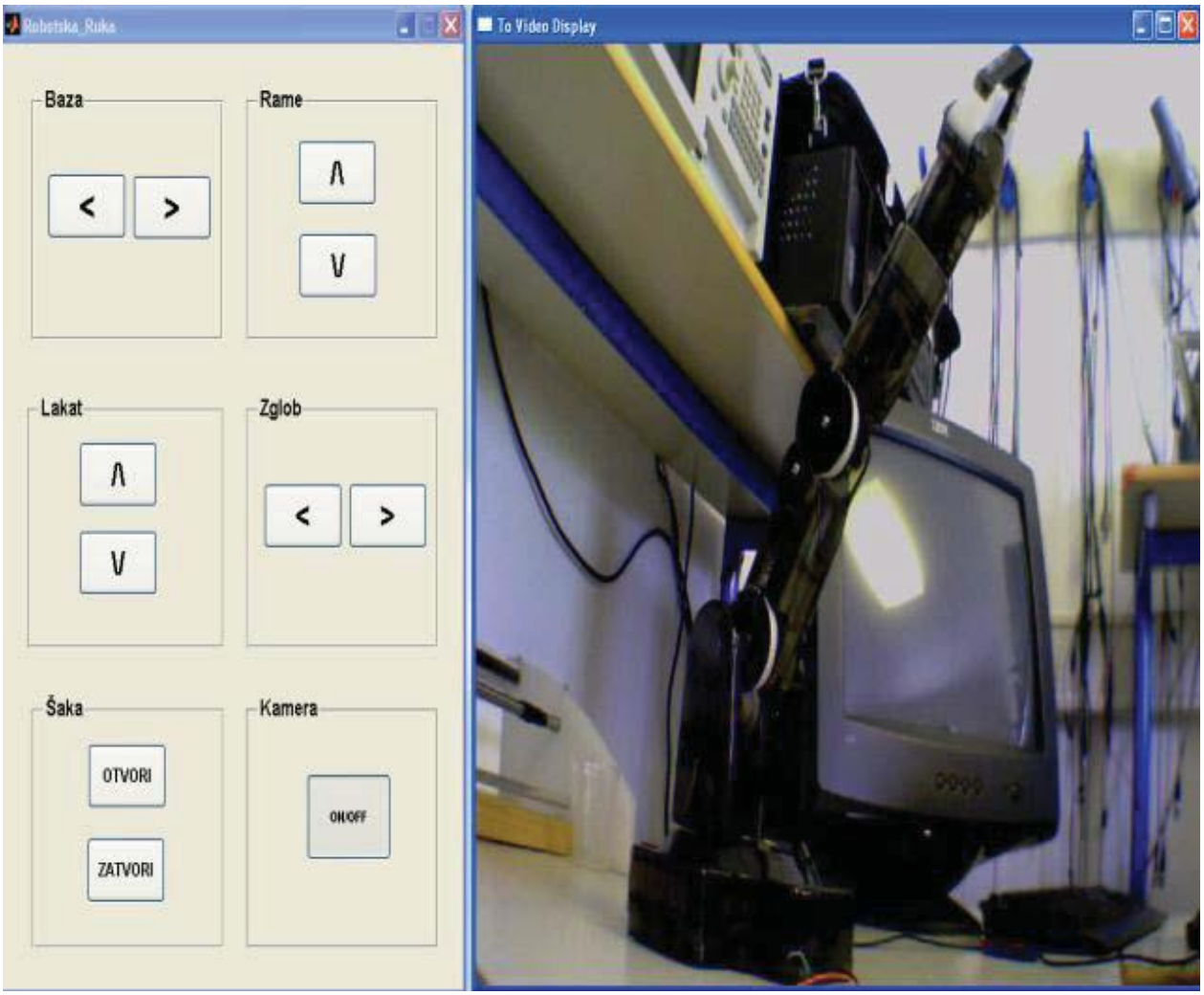

Figure 2.2: Graphical User Interface by Vajnberger *et al* [27]

The interface shown in Figure 2.2 was designed and developed using GUIDE, available MATLAB toolbox [27]. Since MATLAB was used to develop the interface, an additional toolbox called TCP/UDP/IP was installed to provide communication between the server and client computers. Due to the lack of available toolbox in MATLAB to provide remote visual feedback,a VNC server was installed. The camera is connected to the server computer and using the VNC server one can gain access to the server from a remote desktop. In order to access the visual information acquired by the camera, a VNC server

was used. Like every remote controlled robot arm, this arm also has two different communication systems. One of them used between the server and client, as described above. The second one provides lower level communication between the robotic arm and corresponding computer. The used communication is the serial communication RS-232. Serial communication is the most common low-level protocol for communicating between two or more devices [29]. In order to operate the arm using the GUI, user chooses the degree of freedom to manipulate the direction of the movement. By simply clicking on the arrows shown in the interface, users can change the position of the arm as per their requirement. Besides the visual feedback, every joint is implemented with contact micro-sensor. When DOF reaches its final position, the micro-sensor becomes active, and further movement is stopped avoiding damage to the arm.

Vajnberger *et al*'s interface presents a simple but effective way to control a robotic arm but it has few limitations that cannot be over looked. First and foremost the interface is developed using MATLAB and requires installation of additional software. This makes the interface system specific and requires additional work before the interface can be used. The interface presents options to control each joint individually but there is no option for controlling all the joints at the same time. It has limited number of available functions to use. Lastly, this GUI can only be used with the system it is developed for, such as, five degree of freedom robotic arm. The interface requires redevelopment to control multiple systems.

A final recommendation was to develop an interface that is easy to use, allows manual operation of a robotic arm using a joystick and has numerous functions to control a robotic arm.

2.4 Graphical User Interface for Sequence Editor for Spirit (MER)

The author will now discuss the user interface developed and used for the Spirit rover. Over the years, JPL Robotics has created many successful user interfaces to control robotic systems. "These robots have ranged from multi-arm manipulators for microgravity deployment or terrestrial surgical applications, to legged prototype-rovers for rough terrains, to multi-wheeled rovers including flight systems such as the Mars Pathfinder Sojourner Rover and the Mars Exploration Rovers (MERs), Spirit and Opportunity. [28]" The Rover Sequencing and Visualization Program (RSVP) is the name of the development program for the graphical user interface of MER. Linux OS was not the first choice for this development project, but was used afterwards as it uses fewer resources. There are two parts of RSVP, The Rover Sequence Editor (RoSE) and HyperDrive. Each of these individual applications can run in standalone mode, but they are more powerful when used together.

RoSE is a Java application compiled using IBM's much faster compiler, Jikes. RoSE provides efficient GUI support to all the mission commands and interfaces with the institutional tool SEQGEN for sequence validation [9]. RoSE is used to create all command sequences sent to the Mars rovers for every day of the MER mission [28]. Shown in Figure 2.3 is the RoSE interface. This technique simplifies the users' operation to locate all the available commands and makes it simpler for the developers to add new commands. This interface was created for the users who are familiar with the rovers and its system.

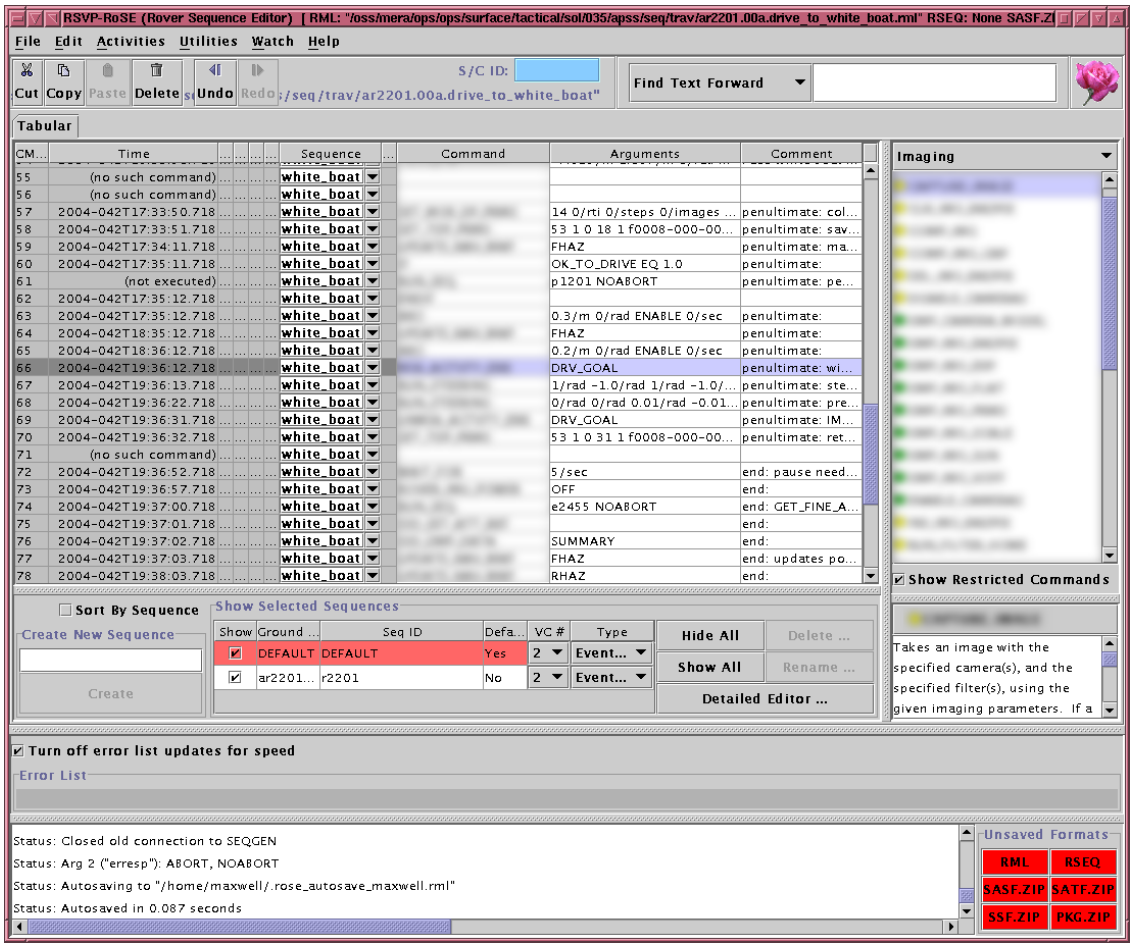

Figure 2.3: RoSE Command Editor (Blurring due to ITAR Restrictions) [9]

The level of interaction required by the operator is determined by the mission profile. Identifying optimal balance for decision making between operators and robots require understanding of the mission objectives, operating environment and human behavior. In space-based mission, reliability is crucial and there is zero error tolerance. This principle was used to develop the task management feature of the interface presented in this thesis. RoSE only allows sequence generation; there is no option to operate the system in real time. In order to enable users who do not have the prior knowledge of the system, it was concluded that the same interface should be used for both autonomous control and sequence planning.

2.5 MotoSim EG

MotoSim EG is software used for the offline teaching and task execution of robotic manipulators. This software and its functionalities are very similar to the RoSE. They both are sequence generators for robots and robotic manipulators, respectively. The ability to show a simulation of the task that the manipulator is going to perform makes MotoSim unique. By doing this, users can verify if the commands are in correct order and if the manipulator will perform as expected before the operation starts. Figure 2.4 shows the MotoSim EG interface with selected robotic manipulator.

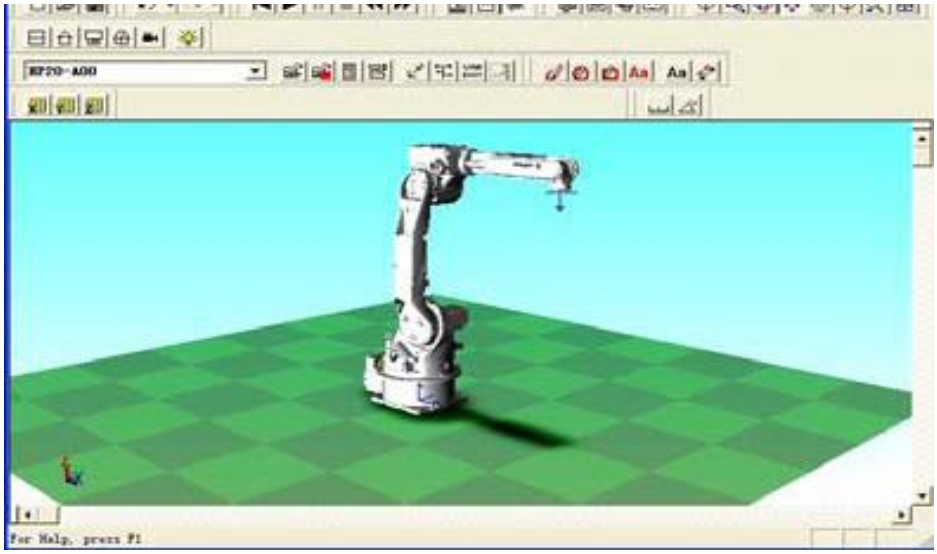

Figure 2.4: Interface of MotoSim EG [32]

In order to schedule any task, simple or complex, users need to type commands. For example, "MOV" has to be entered by the operator in order to move the robot to a particular configuration [32]. Anyone using this software would require prior experience with the software. Since users need to remember the command to schedule a task, it can lead to errors while typing the command. In order to avoid such situations, it is established that the users should have the choice of typing commands or selecting them

when scheduling tasks. This prevents users from making a mistake without compromising the speed and accuracy.

2.6 Background

People's interpretation of the word 'robot' can vary depending on their background, but most commonly the answer would be, "Robots are machines that look like human". Ironically, they are the hardest to design. Robots can be classified by their application, their work environment or by their design. Robots on the Earth can be further differentiated in other types such as industrial robots, household robots, service robots and military robots. Robots in space are remotely operated vehicles, autonomous or tele-operated, in-orbit service robots and remote manipulator system, like the Canadarm.

2.7 Types of Robots

Humans have been using robots for the past few decades in different forms and the use of robots has been increasing exponentially. They are being used in manufacturing industries, military, transportation, medical application, and agriculture.

The job of a typical robotic manipulator is to perform tasks that are difficult, dangerous, or dull. They perform the same task repeatedly with the same precision, whereas humans cannot perform these repetitive tasks with required precision. Dangerous tasks such as lifting heavy object or handling reactive chemicals can also be performed using industry robot without endangering a human life. Due to the speed, accuracy and cost effectiveness of industry robots, they are mostly used in the assembly plants. These types of robots include cartesian robots, cylindrical robots, polar robots, articulated

robots, parallel robots and SCARA (Selective Compliance Assembly Robot Arm) robots.

2.8 Graphical User Interface and Sensors

A user interface of the robotic manipulators works in the same way as the user interface of computer programs. Figure 2.5 shows the generalized design algorithm for a user interface. The user interface of a robotic manipulator is on a remote machine, which interacts with the robotic manipulator using radio or data link.

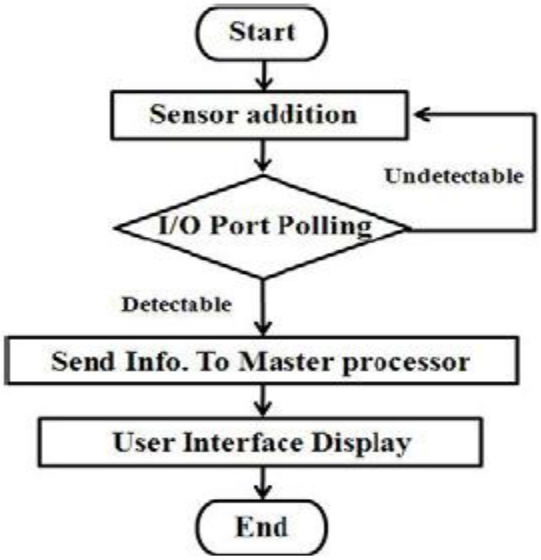

Figure 2.5: Simple user-interface design algorithm [23]

Sensors are very important part of any robotic system. They are the "eyes" and "ears" of the mobile robots. There are many different types of sensors such as laser range finder, shock sensor, temperature sensor, pressure sensor and GPS. All of these sensors are essential for the operation of a robot. When operated in real time, these sensors provide the condition of robots to the operator so the operator can decide the proper steps to control it. If the robot is autonomous, then the data from these sensors are sent to

an onboard computer. Onboard computer is pre-programmed with intelligent algorithms to control the rover with the available data.

Figure 2.6 shows the data flow block diagram of a CanSat, which is equipped with GPS, onboard microprocessor, temperature and pressure sensor.

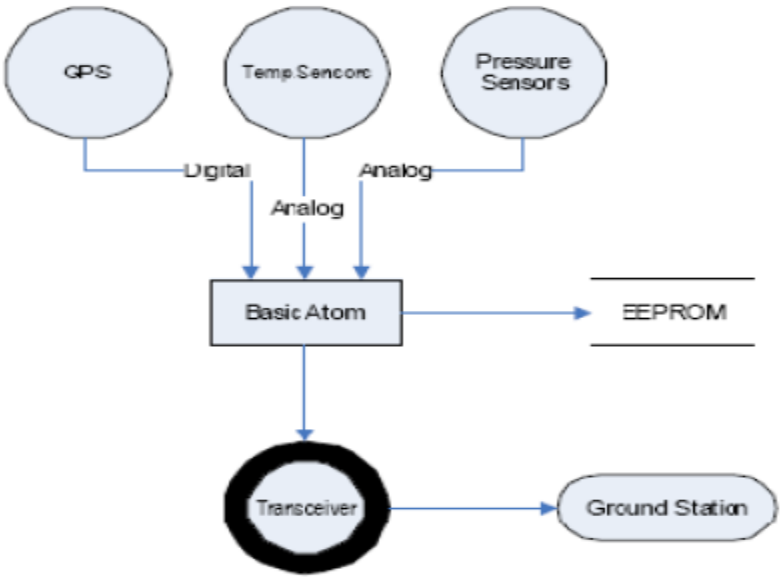

Figure 2.6: CanSat data flow block diagram [4]

The GUI is located on a ground station, which communicates with the communication system of a CanSat using radio link. Once the command is sent, system performs the task and sends back the feedback that is seen on the GUI. Typically, this is how every GUI works for rovers or mobile robots. Communication between robots may not be in real time; in such matter, data is uploaded on the robot when communication is established.

Information collected by these sensors is also sent to the ground station or control center if the robot is operated in real time. The user interface or human machine interface displays this information to the operator, which allows the operator to make necessary changes. Even the most autonomous robots require a user interface, as it gives the way for human to interact

with the robots. Occasionally a user interface is used only for the observation purpose of the hostile environment rather than the operation of the system. Most robots have similar user interface consisting of few buttons and predefined functions.

Chapter 3: Hardware System and Kinematics

The following section briefly explains hardware system used in the experiment for testing the developed user interface.

3.1 Robotic Arm

The robotic arm shown in Figure 3.1 is comprised of three rotary modules and a scoop. The rotary modules are known as Universal Rotary Actuator PRL and built by Schunk GmbH & Co. KG. Modules PRL 60, PRL 80 and PRL 80 are used for wrist, elbow and shoulder joint, respectively. The individual PRL modules have flexible mounting possibilities to an individual lightweight arm using connecting elements [21]. The end-effector of this arm is a scoop, which is specially designed for sample scooping. These *PRL* modules are connected with customized robotic links, as shown in Figure 3.1. The PRL rotary actuator is electrically actuated by the fully integrated control and power electronics.

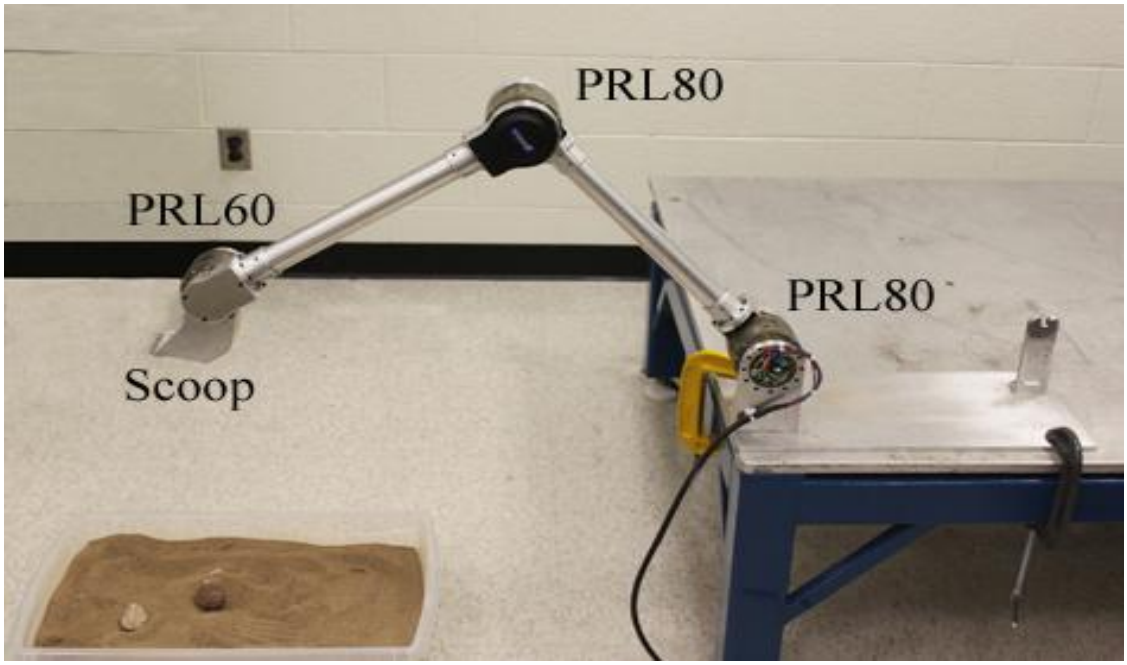

Figure 3.1: A robotic arm for sample retrieval

Each module has a built-in encoder that can provide position, velocity and acceleration. In case of the power failure, integrated magnetic brakes can hold the position of the manipulator. Modules allow varied range of communication methods including Profibus DP, CAN-Bus or RS-232 [21]. The PRL modules are connected electronically to a CAN-Bus interface, which is controlled by an embedded computer running Linux.

3.1.1 Embedded Computer - PC/104

PC/104 is a standard of the embedded computers defined and controlled by the PC/104 Consortium in February 1992 [18]. The PC/104 Consortium defines both the form factor and computer bus of an embedded computer. PC/104 is popular for small computing modules typically used in industrial control systems and vehicles. It is specially used where the applications depend on a reliable data acquisition despite an often-extreme environment. PC/104 Systems are small and have very low power requirement, making them the ideal candidate for robotics applications. A typical system includes common functions like a CPU, motherboard, analog-to-digital converter and digital input/output module. Since PC/104, as shown in Figure 3.2, is a stripped down version of a PC with a different form factor, program development tools used for PCs can be used for the PC/104 systems as well. This greatly reduces the learning curve for the programmers and hardware designers.

The embedded computer system is connected to the robotic arm using a CAN-Bus for data communication. An on-board Ethernet port is connected to the Ryerson University's Systems and Controls Laboratory's network using an Ethernet cable. This allows control of the robotic manipulator over the internet connection.

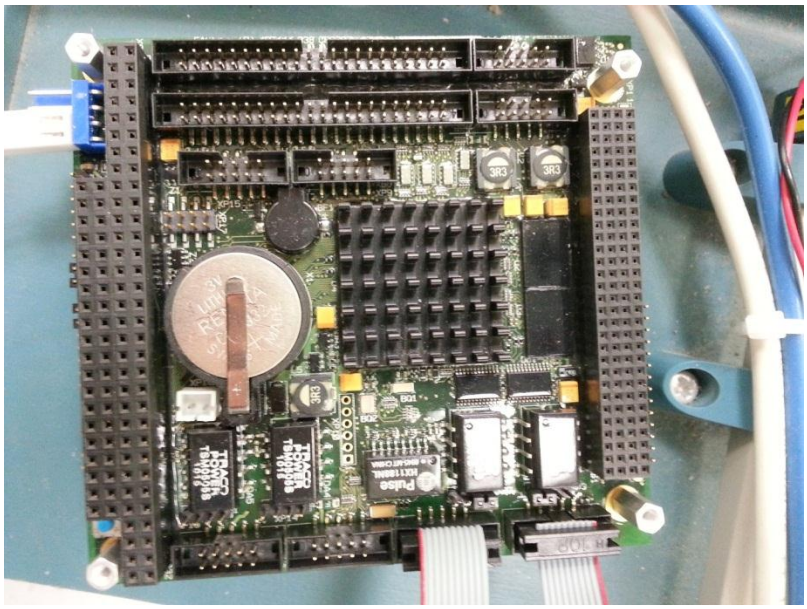

Figure 3.2: Embedded computer (PC/104)

3.1.2 Joystick

A joystick, also known as the control column, is a main control device of many modern aircrafts. A joystick is an input device consisting of a stick that swivels on a base and reports its direction to the connected device. In recent times, the implementation of joysticks has become very common in industrial applications such as; cranes, robotic arms, mining trucks. Furthermore, most Unmanned Aerial Vehicles (UAVs) and Remotely Operated Vehicles (ROVs) consist of at least one joystick to control either the vehicle itself or onboard cameras.

Most modern joysticks have two-dimensional movement, similar to a mouse. A joystick is generally configured such that moving the stick forward or backward corresponds to a movement along the Y-axis and moving it sideways corresponds to movement along the X-axis. Additionally, joysticks also have one or more buttons; they are simple push button switches. Most modern joysticks use a USB interface for connection to PCs.

A Joystick used in this experiment is a Logitech Attack™ 3 Joystick, shown in Figure 3.3. It is "An easy-to-use joystick with a full array of customizable controls that gives you durability and ambidextrous control—right out of the box. [15]" It has full X-Y axis control and eleven customizable buttons. Force feedback on this device is unavailable. Like most joysticks, it also uses a USB interface for PC connection.

Figure 3.3: Joystick

The developed GUI is deployed on the client computer, as shown in Figure 3.4. The operator can send command through the GUI to control the robotic manipulator for sample retrieval and collection.

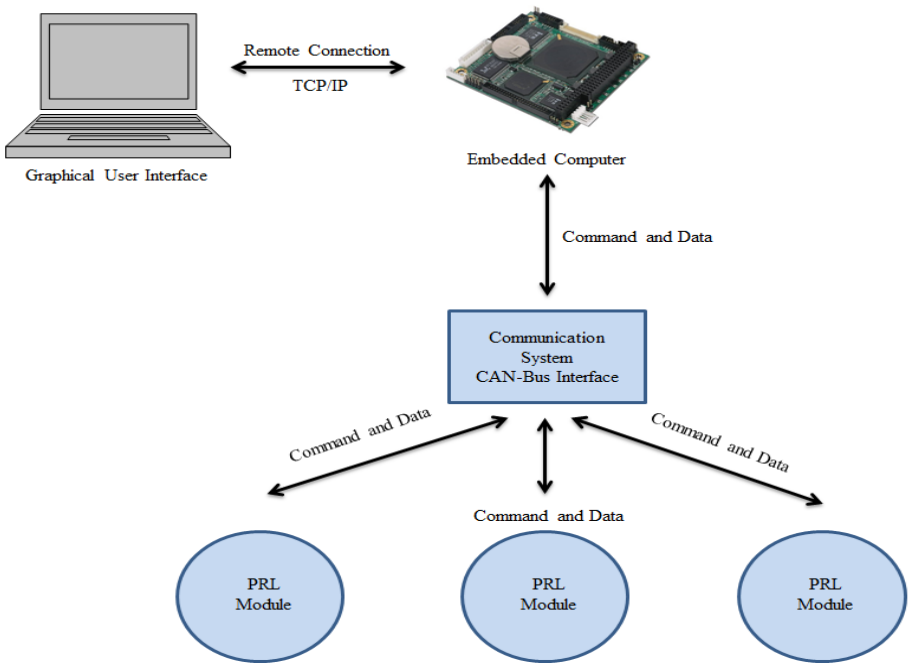

Figure 3.4: The hardware schematic

3.2 Kinematics

Robotic manipulator is typically comprised of series of links connected by joints. Simple joints, such as revolute and prismatic, of robotic manipulators have one degree of freedom [24]. A *revolute joint*, also known as pin joint, is a one DOF kinematic pair, which provides rotation function. A *prismatic joint*, also known as slider, is a one DOF kinematic pair, which provides linear sliding movement between two bodies.

There are two parts to the kinematics analysis, *forward* and *inverse.* In forward kinematics, given the angles of joints, one is interested in finding the position and the orientation of the end-effector. In inverse kinematics, the position and orientation of the end-effector is known and one is interested in finding the joint angles for each joints. The inverse kinematics is more useful when programming a robot or designing a user interface. Users want to manipulate the end-effector since most tasks such as scooping, grasping and painting are performed by the end-effector.

3.3 Denavit – Hartenberg (DH) Convention

Denavit and Hartenberg proposed a matrix method for the forward kinematics analysis [11]. First, the DH convention parameters will be expressed and the forward kinematics analyses will be done accordingly.

The DH convention is mainly used in robot manipulators that are comprised of an open kinematic chain, in which each joint has one degree of freedom. In DH convention, each homogenous transformation matrix H_i is represented as a product of four quantities associated with each link. The four parameters a_i, $\propto_i$, d_i, and Θ_i are known as link length, link twist, link offset, and joint angle, respectively. For a revolute joint, Θ_i is the revolute variable and for a prismatic joint, d_i is the prismatic variable. Using these parameters, the orientation matrix H_{i-1}^{i} is given by equation 3.1.

$$\begin{bmatrix} \cos(\theta)_i & -\sin(\theta)_i\cos(\propto)_i & \sin(\theta)_i\sin(\propto)_i & \propto_i \cos(\theta)_i \\ \sin(\theta)_i & \cos(\theta)_i\cos(\propto)_i & -\cos(\theta)_i\sin(\propto)_i & a_i\sin(\theta)_i \\ 0 & \sin(\propto)_i & \cos(\propto)_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad 3.1$$

The DH convention is implemented through following steps; Figure 3.5 indicates the frame assignment. Here, a_i is the distance from axis Z_{i-1} to Z_i measured along the axis X_i. Angle $\propto_i$ is the angle from axis Z_{i-1} to Z_i measured about the axis X_i. Distance d_i is the distance from axis X_{i-1} to X_i measured along the axis Z_{i-1}. Lastly, angle Θ_i is the angle from axis X_{i-1} to X_i measured along the axis Z_{i-1}.

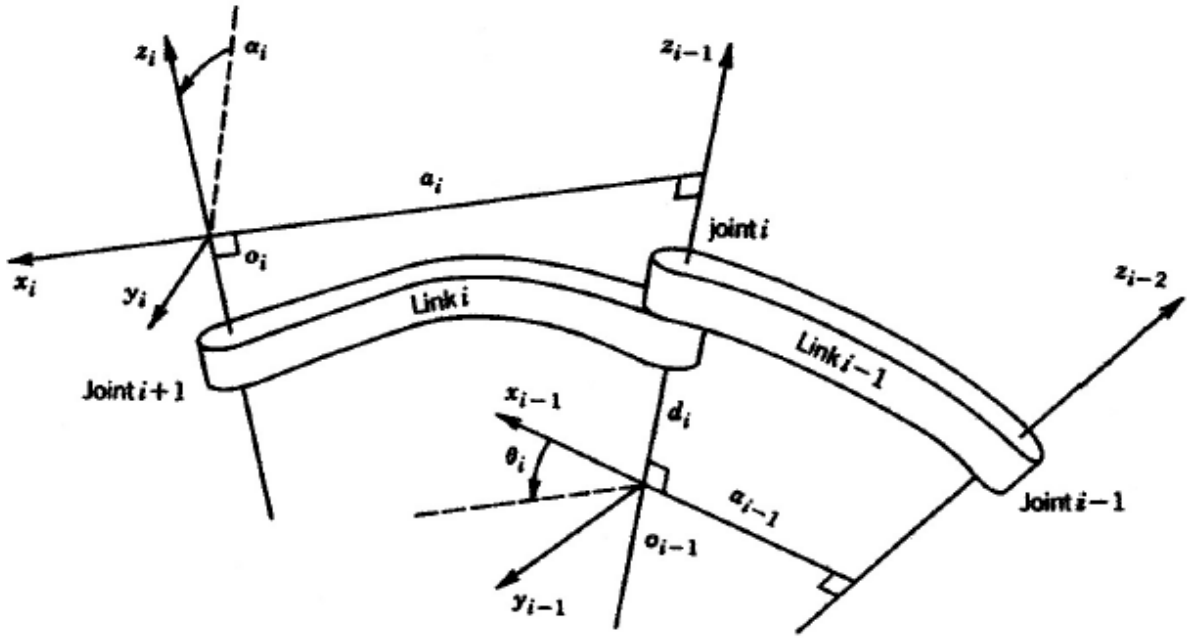

Figure 3.5: Frame assignment for the DH convention [24]

1) Number the links and joints starting from the base. The joints are numbered 1 to *i* and links are number 0 to *i*.
2) Assign link's co-ordinate system for each joints according to the following rules:
    a. Z_{i-1} axis is chosen along the axis of motion of the joint *i*.
    b. Axis X_i is set so it is perpendicular to and intersects Z_{i-1} axis. The origin O_i is assigned at the intersection of axis X_i and Z_i. There are three possible instances when assigning X_i axis.
        i. When axis Z_{i-1} and Z_i are not coplanar, there exists only one possible line for axis X_i, which is the shortest line from axis Z_{i-1} and Z_i.
        ii. When axis Z_{i-1} and Z_i are parallel to each other, there are infinite numbers of solution for the assignment of axis X_i. Generally, it is simplest to choose axis X_i such that it passes through origin O_{i-1} making d_i zero. In this scenario, $\propto_i$ is always zero.

iii. When axis Z_{i-1} and Z_i intersect, axis X_i is normal to the plane. Positive direction of axis X_i is arbitrary for such cases. Parameter a_i is always zero in such scenarios.

c. Axis Y_i is chosen to satisfy the right-handed co-ordinate system.

d. To assign co-ordinate system to the tool, axis Z_n is chosen in the *approach* direction, axis Y_n is chosen in the *slide* direction and axis X_n is chosen in direction normal to the other axis.

3.4 DH Parameter of Robotic Manipulator and Forward Kinematics Analysis

To find the DH parameter of robot, the wire frame model must be drawn, shown in Figure 3.6.

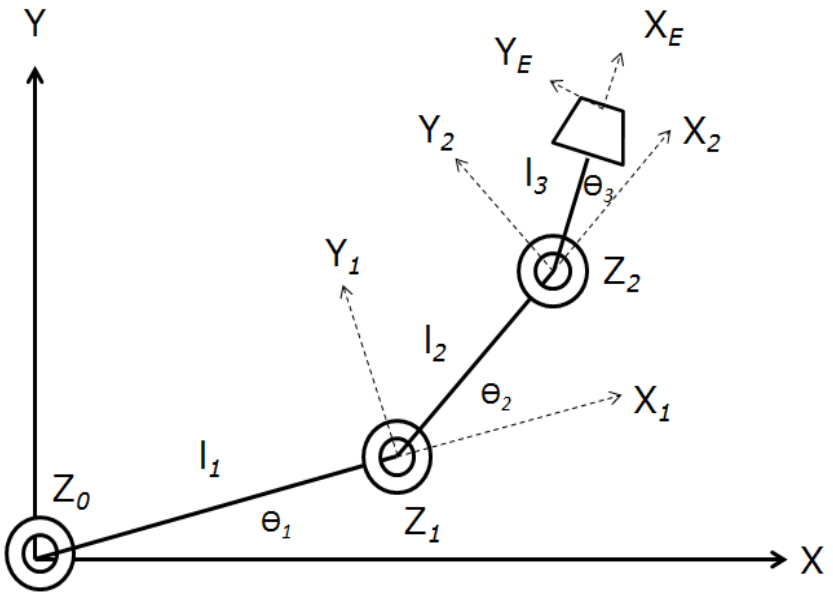

Figure 3.6: Model of robotic mast

The DH parameters for the manipulator are defined in Table 3.1 and Table 3.2 contains the length of links and their description.

Table 3.1: DH parameters for the robotic mast

i	$\propto_i$	a_i	d_i	Θ_i
1	0	l_1	0	Θ_1 (Joint Variable)
2	0	l_2	0	Θ_2 (Joint Variable)
3	0	l_3	0	Θ_3 (Joint Variable)

Table 3.2: Link lengths and description

Parameters	Value	Remark
l_1	0.460 m	Length of the upper arm (the link connecting Shoulder and Elbow)
l_2	0.440 m	Length of the fore arm (the link connecting the Elbow and Wrist)
l_3	0.110 m	Length of the scoop (from the tip to the centerline of the Wrist)

$$T_1^0 = \begin{bmatrix} \cos(\theta)_1 & -\sin(\theta)_1 & 0 & \ell_1 \\ \sin(\theta)_1 & \cos(\theta)_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 3.2$$

$$T_2^1 = \begin{bmatrix} \cos(\theta)_2 & -\sin(\theta)_2 & 0 & \ell_2 \\ \sin(\theta)_2 & \cos(\theta)_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 3.3$$

$$T_3^2 = \begin{bmatrix} \cos(\theta)_3 & -\sin(\theta)_3 & 0 & \ell_3 \\ \sin(\theta)_3 & \cos(\theta)_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 3.4$$

Using equations 3.2, 3.3, and 3.4 transformation matrix from initial frame zero to final frame *n* can be found. The transformation matrix T_3^0 is given by equation 3.6.

$$T_3^0 = T_1^0 T_2^1 T_3^2 \tag{3.5}$$

$$T_3^0 = \begin{bmatrix} \cos(\theta_1+\theta_2+\theta_3) & -\sin(\theta_1+\theta_2+\theta_3) & 0 & P_x \\ \sin(\theta_1+\theta_2+\theta_3) & \cos(\theta_1+\theta_2+\theta_3) & 0 & P_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

where, $P_x = \ell_1 \cos(\theta)_1 + \ell_2 \cos(\theta_1+\theta_2) + \ell_3 \cos(\theta_1+\theta_2+\theta_3)$

$$P_y = \ell_1 \sin(\theta)_1 + \ell_2 \sin(\theta_1+\theta_2) + \ell_3 \sin(\theta_1+\theta_2+\theta_3)$$

Orientation, $\emptyset$, of the end-effector is given by equation 3.7 and position of the end-effector in global frame is given by P_x and P_y.

$$\emptyset = \theta_1 + \theta_2 + \theta_3 \tag{3.7}$$

3.5 Inverse Kinematics Analysis: Geometric Approach

The inverse kinematics analysis is performed to find the joint angles from a given pose of the end-effector. For the common kinematics arrangements, such as manipulator in this thesis, the variable q_1, q_2, q_3 corresponding to O_0 can be found using a geometric approach. For many robots, the orientation is set by a wrist at the end of the manipulator. For simplicity, convert position of tip point with respect to the base, to position of the wrist point with respect to O_0. Then the analysis is performed using the wrist point position. The general approach to solve for joint variable q_i using a geometric approach is by projecting the manipulator onto x_{i-1} and y_{i-1} plane

and solving a simple trigonometry problem. For example, to solve for θ_1 project the manipulator onto the x_0 – y_0 plane and use trigonometry to find θ_1.

Given X_E, Y_E and orientation, ∅, the joint variable θ_2 can be found using the method below.

$$x_2 = X_E - l_3 \cos(\emptyset) \qquad 3.8$$

$$y_2 = Y_E - l_3 \sin(\emptyset) \qquad 3.9$$

Therefore,
$$\cos(\theta_2) = \frac{x_2^2 + y_2^2 - l_1^2 - l_2^1}{2l_1 l_2} \qquad 3.10$$

$$\sin(\theta_2) = \pm\sqrt{1 - (\cos(\theta_2))^2} \qquad 3.11$$

The $\pm$ of the square root generates two solutions for θ_2, they correspond to the elbow-up and elbow-down, respectively, as shown in Figure 3.7.

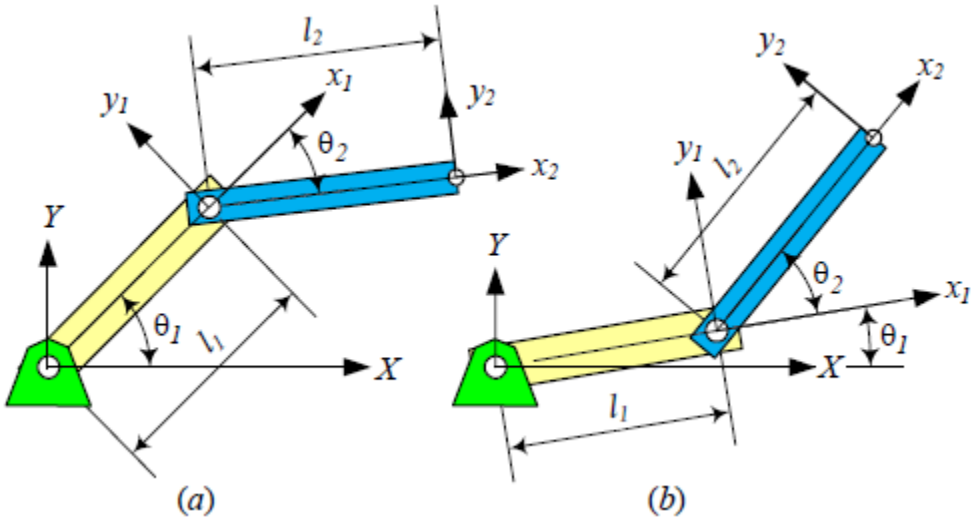

Figure 3.7: a) elbow-up, b) elbow-down configuration [11]

Therefore, θ_2 can be found using equation 3.12,

$$\theta_2 = \text{atan2}(\sin(\theta_2), \cos(\theta_2)) \qquad 3.12$$

Similar process is used to find the joint variable θ_1. Shown below is the calculation for finding θ_1.

$$\theta_1 = \text{atan2}(y_2, x_2) - \varphi \quad (3.13)$$

where,

$$\varphi = \text{atan2}(\sin(K),\ \cos(K)) \quad (3.14)$$

$$\cos(K) = \frac{x_2^2 + y_2^2 + l_1^2 - l_2^2}{2\sqrt{(x_2^2 + y_2^2)}\, l_1} \quad (3.15)$$

$$\sin(K) = \pm\sqrt{1 - (\cos(K))^2} \quad (3.16)$$

Now that the values of θ_1 and θ_2 is known, the joint variable θ_1 can be easily calculated the using equation 3.17.

$$\theta_3 = \emptyset - \theta_1 - \theta_2 \quad (3.17)$$

In the next chapter, guidelines of an effective graphical user interface design are presented. These guidelines have been used throughout the developed interface.

Chapter 4: Graphical User Interface Design Principles

The purpose of this thesis is to design a graphical user interface for a robotic manipulator. It is essential to understand how a graphical user interface works and the process used to design it. This section will focus on general guidelines used for designing and developing a graphical user interface.

4.1 Guideline to GUI design

With the ever-changing technology, there are no set rules that can help design an interface. There are still some basic principles for all the good interfaces. Unfortunately, they are exhibited by very few programs. The author will discuss some of the most acknowledged user interface design guidelines. Subsequently, they will be compared against each other. Most of the guidelines discuss about designing an interface for software or websites. Focus of this research is to develop a user interface for a robotic arm. During the operation of a robotic arm, an operator can get hurt if incorrect commands are executed. Simple system crash on the computer may have adverse effect on the robotic arm such as sending the arm in out of control motion.

4.2 Interface Design Guide by Donald Norman

First, the author will discuss the interface design principles defined by Donald Norman in early nineties. In his book, The Design of Everyday Things, he explains about how to take a conceptual model to a physical design. Throughout the book, Norman strives to explain that the visibility, feedback, constraints and consistency are essential for an effective user interface design [17]. These principles are discussed below in detail.

4.2.1 Visibility

There are two aspects to making objects visible: what a user can do and response to the action performed. In terms of what a user can do, the more visible the functions are, easier it is to use them. Users cannot do what they cannot see, also known as the concept of *what you see is what you get* (WYSIWYG). While it is important to make all the functions visible, one should not make a cluster of functions. They should be organized in such a way that they are easy to navigate. Most people who have used wording software can relate to the example. Figure 4.1 shows the user interface of Microsoft (MS) Word 1995 with all the available functions And Figure 4.2 shows the interface of Microsoft Word 2012. By quick visual comparison, several differences between two user interfaces designed for the same software can be noticed.

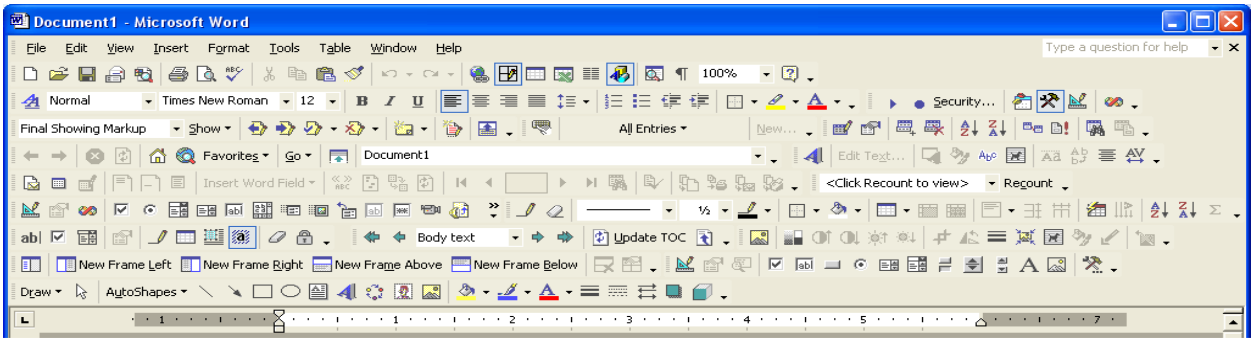

Figure 4.1: Microsoft Word 1995 with all the functions[1]

The interface of MS Word 2012 is lot more visually appealing than of MS Word 1995. In MS Word 2012 important functions are visible on the main screen and ribbon style menus displays additional available functions, whereas MS Word 1995 makes a cluster by displaying all the available functions.

[1] http://www.codinghorror.com/blog/2006/02/sometimes-a-word-is-worth-a-thousand-icons.html

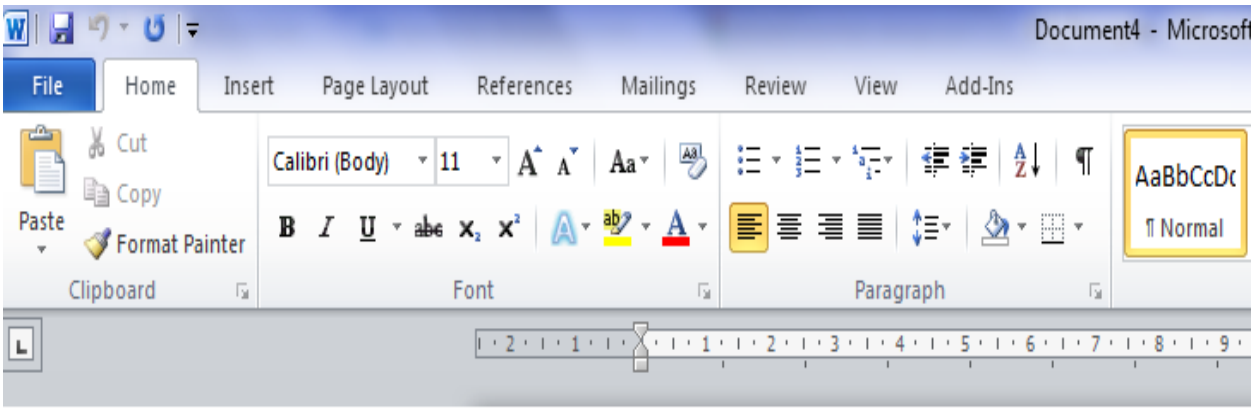

Figure 4.2: MS Word 2012 user interface

4.2.2 Constraints

The design concept of the constraints refers to determining ways of restricting some of the actions a user can perform. For example, sliders used to adjust volume on a computer. It can only be moved between the lower limit of zero to upper limit of hundred. By constraining, developers can make sure that there is no mis-operation. According to Norman, there are multiple ways of achieving that: physical constraints, semantic constraints, cultural constraints and logical constraints [17]. Physical constraints are physical limitations that constraint possible actions. It is not possible to move the cursor outside of the screen; that is a physical constraint. Logical constraints use reasoning or logic, for example number sequences or alphabetical order. Semantic constraints rely on answer of why we do what we do. Lastly, cultural constraints rely on accepted cultural conventions, such as red means danger or stop.

4.2.3 Feedback

"Give each action an immediate and obvious effect. [17]" Feedback is very important when designing an effective user interface. Feedback is necessary in two cases, letting users know of their actions and in the case of errors

letting users know the type of error. Feedback is about sending information back to the users. One cannot start the next task unless it is made clear that the current task has been completed. Feedback is also necessary in case of errors. We, as humans, are prone to making errors. It is the responsibility of the interface designer to warn users when an error is made.

4.2.4 Consistency in Design

The system should be consistent in its structure and commands to minimize the memory problems faced while performing the task. His research shows that the lack of consistency leads to errors. Consistency means using the similar elements to achieve similar tasks. A consistent interface is the one that follows rules such as using the same input method. Throughout the interface, texts used should be same except in the case of warning users of an error. The use of graphics and colors are important and useful if kept consistent. Everyone knows red color is used to warn when something is wrong. Now if the developed interface uses red color to indicate that the system has no error. That may lead to confusion, which may result in errors made by the user.

Knowing how to design an interface is just as important as knowing, what the users want to use the software for? Simplifying the structure of the tasks leads to less use of long-term memory. This results in fewer errors made by users. Tasks should be simple in structures, requiring less planning or problem solving. Complex tasks can be restructured into a series of simpler tasks eliminating the risk of error. As mentioned earlier, this can be achieved by making things as visible as possible. When clicked on the button labelled "add" on calculator, it should do the addition of two numbers not the opposite. Hence, one needs to get the mapping right. Use the power of constraints as much as possible; it is the easiest way of forcing users to do

the right thing. Always expect that the users will eventually make a mistake so design for error. Last but not least, when all else fails, standardize the task; this leaves users with no choice but to do what is being shown.

4.3 Designing Interface: Patterns for Effective Interaction Design

Jenifer Tidwell discusses numerous practices currently used by interface designers to make an effective GUI. In her book, Designing Interface: Patterns for Effective Interaction Design, she discusses about different layouts used to make an interface look appealing. It is of utmost important to focus on the users even before starting the interface design. Tidwell also explains different means of collecting information from users that will lead to fewer errors.

4.3.1 Focus on Users

"A Means to an End: Everyone who uses a tool – software or otherwise – has a reason for using it. [26]" For instance, performing a search, buying a product and in the case of this thesis: operating a robotic arm. Easiest way of learning what the users want to do is by asking "why". For examples, why does a person use an email client? Answer is obvious "to read an email". When doing a basic research about users' need, one should try to learn following:

- What they would like to achieve by using the software?
- Specific task they would like to perform
- Prior knowledge of using similar software
- Language and words they use to describe what they are doing

These can be achieved by various ways such as surveys, case studies and observations. When designing a user interface, one should also keep in mind

what inspires and motivates the users to learn. In the end, develop a software that can be used "out of box".

4.3.2 Organizing the Page Layout

In simple words, a page layout means laying information on a page that makes the page attractive and appealing to users. There is more to page layout than just making the page attractive. An experienced designer can arrange the page in such manner that not only it makes the page appealing but also draws users' attention to the most crucial part of the page. There are three important aspects of page layout: visual hierarchy, visual flow and use of dynamic displays.

4.3.3 How to List Things

Most search results, articles, forum posts, documents and more are in one way or the other shown in the form of a list. Every webpage or moderately complex interface includes lists. By looking at a list, the user should be able to get a general idea of what is the purpose of the list. At times, simple but carefully chosen words can fulfill that purpose or sometimes it requires more than just words. The list should also have a sorting order, for example sort alphabetically or by time. Grouping the items inside the list is an effective way of displaying similar types of information. By implementing that, users do not have to browse the entire list to find similar commands or options. Now that the list is there, what should users do with the list? Generally, when users see a list, their natural instinct is to click on their choice for further work.

According to Tidwell, a list can be displayed in two-panel selector, where one panel shows the list of items and other shows the content of selected item [26]. This is commonly used in picture gallery of every computer. Another

effective way is to list all the items or information. When users select the item listed, the interface should open that list to show additional related information.

4.3.4 Doing Things: Actions and Commands

Designing an interface that looks good and is easy to navigate is only one part of making an interface. One other aspect is getting the interface to fulfill its purpose by allowing users to perform correct actions. More often than not, a designed interface contains buttons. They are placed directly onto the interface and by simply clicking on it; the intended action can be performed. Menu bars are also common example for putting more commands in less space. Certain actions and commands such as keyboard shortcut "Ctrl-S" have been used for saving the file. Those traditional commands should be kept in mind and should be used for the same purpose they are intended for, altering them can lead to confusion among the users. Additional ways of displaying available actions are pop-up menus, drop-down menus and action panels. One interesting feature to note is *Hover tools.* It is used to show more information about the task or actions. When users take the mouse pointer over the button or input field, more options are displayed. This is typically used to display what the purpose of the function is and how to utilize the function.

4.3.5 Getting Data from Users

At some point, users have to input relative data to perform the action. For example, to add two numbers on a calculator, the user needs to enter two numbers so that the addition can be performed. It is hard for users to enter data if they do not know what they are being asked. The interface must be very specific about what users are being asked to enter. When possible, list

all the choices available for the users so that they do not have to remember. This leads to fewer errors and happy users. Drop downs, combo boxes and lists are a great way to ensure this. When entering data, users are bound to make mistakes. If an error is made, the interface must politely indicate the error and allow them to return to the last stage.

There are multiple ways of getting input from users such as check box, radio buttons and text field. Choice of which form to use depends on the available space. Some controls take up more space than the others do. It is also dependant on the how experienced users are with computers. For users, new to the computer, using a text field for data input may lead to more errors. In that case, one should use radio buttons so that all the choices are stated and visible. Available technologies also play an important role in selecting which form one uses to get the user's input. Writing a GUI in HTML provides limited options whereas open source GUI toolkits provide richer sets of option.

4.4 Galitz's View on Design Guidelines

4.4.1 Aesthetically Pleasing

Visually pleasing design is attractive to the eye and makes a system accessible and inviting. It also conveys messages clearly to users. A visually pleasing design can be provided by creating meaningful contrast between screen elements. One should create groups and align screen elements to make the interface more pleasing.

4.4.2 Clarity and Compatibility

The interface must be clean in visual appearance and should be understandable. It should relate to the user's real world concepts and functions. The design must be compatible with the needs of the user and

client. It is very important to understand the users' need to create an efficient interface.

4.4.3 Configurability and Consistency

The interface should be designed such that the end users can change the layout according to their need. Easy configuration and reconfiguration of a system enhances a sense of control, encourages an active role in understanding and allows users' personal preferences. Design consistency is very common in all the systems, be it a simple interface such as calculator or complex interface such as Windows 7. Consistency is very important as it reduces the learning curve by allowing the skills learned in one situation to be transferred to another.

4.4.4 Directness and Predictability

The tasks should be performed directly and intuitively. In order to avoid mistakes, the interface should provide direct and intuitive ways to accomplish tasks. It can be achieved by simply putting tabs for each important function. Tasks, displays and movement through the system should be anticipatable through the user's previous knowledge. Another feature seen in Linux is *tab* completion, which is predicting the full command by pressing a *tab* key once first three letters of the command are written.

4.4.5 Simplicity

Complex systems are often not fully utilized or used incorrectly. Complexity confuses the user that leads to erroneous input. Simple interface is easy to learn. By designing a simple interface, one can make it efficient and avoid mistakes resulting in less time and energy consumption.

4.5 Johnson's User Interface Design Rules

Following user interface design guidelines is not as simple as following cooking recipes. Designers are often given final goal of the user interface. Most design rules are general, leaving them open to any interpretation designers like. Occasionally, more than one rule will apply to the same situation and lots of trade-off is necessary. Lastly, many recent developers of the user interface lack background in cognitive psychology. Jeff Johnson explains some design rules in a way that is easy to understand by someone who does not have psychological background.

4.5.1 Focus on Users and Tasks First

Everything humans perceive is based on what they *expect* to perceive [12]. Perception is based on experience, past and future; same applies for a user interface. It is important to know background of the users when focusing on them. Users will expect and perceive information in a user interface based on their past. In Figure 4.3, based on the first three pages, users expect "Next" button to be in the same location. Hence, it is important to know users before designing the interface.

Perception is also biased by users' goals [12]. Users tend to *ignore* everything except what they are looking for. For example, when browsing website with specific goal in mind, users quickly skim through the screen and only read texts related to their goals. Proximity indicates similarity therefore keep similar functions together.

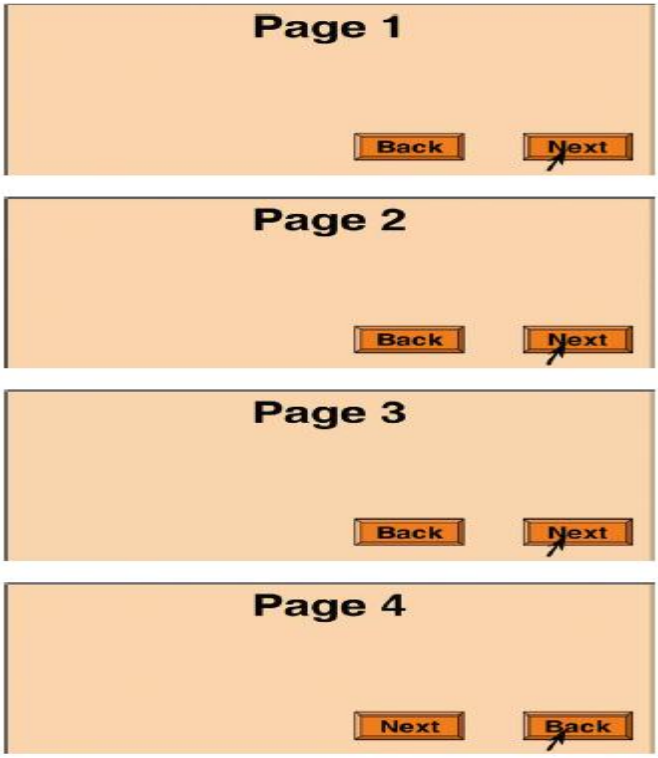

Figure 4.3: "Next" button in Page 4 is perceived to in same location as other three pages [12]

4.5.2 Conform the Users' View of the Task

Throughout the interface, the developer should strive for naturalness. As explained earlier, it is natural for users to read from left to right, "Ctrl + C" means copying the selected text; keep everything that way. Humans in general seek visual structures wherever they look [12]. Hence, visual hierarchy is important. It lets people focus easily on the relevant information. Break the information in to distinct sections and break large sections in subsections. Reading big chunk of texts is boring; break them into related and distinct sections. This makes the interface easy to navigate and interesting. Present higher level and lower level functions as a hierarchy. Higher-level functions should be more direct and visible to users. Users of the software do not need to know programming details of the software.

4.5.3 Design for the Common Case

Once all the requirements of the user interface are known, determine functions that will be used often. Design the interface around the commonly used functions, it makes the interface easy to use and navigate. When using texts in the interface keep them simple and avoid uncommon or unfamiliar vocabulary. When describing a function, use layman's term so that it is easy to grasp. It is best to avoid noisy background and tiny texts. Using color in the user interface can greatly improve the look and functionality of the interface. Red color indicates danger or warning so avoid using it unless it is for warning. Similarly, green, orange and yellow colors also have a pre-determined function.

4.5.4 Do Not Complicate Users' Task

Attention span of a human brain [12] is limited when engaged in more than one task. During the operation of the software, users want to finish the work as early as possible. In order to make the user interface effective and easy to use, avoid giving users extra problem to solve when they are using the software. Sometimes it is impossible to proceed further without solving a complex problem. In such cases, divide the problem in smaller steps and do as much work as possible "behind the screen".

4.5.5 Design for Responsiveness

Using software, which does not provide feedback, is confusing. Without feedback, it is difficult to know if the action performed was correct or not. To avoid the confusion, acknowledge user actions instantly. Provide users with as much feedback as possible. In case the action performed is wrong, warn the user about it and show them how it can be corrected. Allow users to exit

the operation whenever they want and let users be in control of the operation whenever possible.

4.6 Common Mistakes

Since an intelligent interface allows users to perform tasks in their own ways, it must be easy to learn and user friendly. Despite their popularity, there are still some basic principles for all the good interfaces that only are exhibited by a few programs. Common mistakes that are easily overlooked by the designer can be summarized by the following three aspects:

1. Forgetting the user

   Developers often base the design of an interface on what they know instead of what the users know. This problem is very common in the field of engineering but is more widespread in the interface design. This is because the inadequately designed interface immediately makes users feel that they are incapable of operating the system. Even if state of the art system is developed, if users cannot use the system effectively, it is a waste.

2. Exposing too many features at the top level

   Designer may think that by putting all the features at top level, they are providing users with more functions. Unfortunately, by doing so the designer prevents users from selecting desired function and makes the interface look disturbing and disorganized. A well designed GUI presents higher level and most used functions at the top level with the additional functions hidden in a menu or drop down panel.

3. Taking control away from the users

   Users are most happy when they are in the control of the system, not vice-versa. When the control is taken away, users feel frustrated and get distracted. They are more reluctant to use the system again after such instances. Instead of taking away the control from users when mistake is made, display a warning and a way to fix the error.

4.7 User Interface Design Guidelines

Designing an interface that is easy to navigate, effective, and easy to use is a trade-off between many different factors. It is nearly impossible to follow each and every guidelines stated above. One can notice, all the guidelines in this section explain how the interface should be for software. They do not mention necessary steps that one has to take when designing the interface for a machines or a robotic arm. Table 4.1 below compares notable user interface guidelines.

From Table 4.1, it is clear that focusing on users is of utmost important. It only makes sense because in the end, they will be the one using the interface. Each design guidelines explained in the section above is crucial for designing an effective interface. Additional guidelines are needed to design an interface for a robotic arm that is intuitive, user friendly and safe. The author will now explain interface design rules and how they can be used to design the user interface for robotic arm.

Table 4.1: Comparison of user-interface design guidelines

Rule	Norman [17]	Tidwell [26]	Galitz [6]	Johnson [12]	Shneiderman And Plaisant [22]	Stone et al. [25]
Visual Hierarchy		✓	✓	✓		
Visibility	✓					✓

Constraints (Physical, logical)	✓					
Feedback and Acknowledgement	✓	✓	✓	✓	✓	✓
Focusing on users	✓	✓	✓	✓	✓	✓
Simplicity	✓	✓				✓
Easy Reversal of Actions				✓	✓	✓
Consistency	✓	✓	✓	✓	✓	✓
Preventing Errors by Design	✓					✓
Organizing the Page Layout (Structure)		✓	✓	✓		✓
Universal Design				✓	✓	

4.7.1 Focus on Users

Throughout this section, it has been stated and restated that the focusing on users is the key to a successful user interface. A user centered design starts with the people who will use the system. The aim is to design the graphical user interface to fit people and their tasks. Developers of the graphical user interface should keep in mind the answers to the following questions.

- Who are the end-users?

There are different types of users, such as direct users, remote users, support users etc. Direct users use the system hands-on to perform their own task. The end users for the graphical user interface designed in this

thesis are engineers that have prior knowledge of the system as well as the hardware.

- What characteristics and knowledge do the users have?

These engineers have enough knowledge to understand the function and application of the robotic arm.

- What will the system mean to them?

This graphical user interface is used for the educational and research purposes in the lab.

- What are the usability requirements?

Agreeing what tasks users will perform is a key prerequisite to the graphical user interface design. In order to understand how the new system and its graphical user interface will assist users, the initial aim is to understand what the user will do and how they will do it. A graphical user interface designer should also evaluate how the tasks will be performed with the developed GUI.

4.7.2 Joint Space Control and Task Space Control

There has been no mention of how users should allow the control of a robot. Most literatures explain the design of user interface for software used in computer; therefore, they do not run into this problem. Anyone with background in robotics knows that the robots can be controlled in joint space or task space. Controlling a robotic arm in joint space is simple and straightforward. A user simply enters the local joint parameter; and the selected joint moves to the desired position. Design of an interface for such task does not require much work. Most rules described above can be used for this task.

Controlling the robotic arm in task space requires lot of calculations and conditions. Most users would not want to do the calculations when using the interface and frankly, it is hard to perform such complex calculations. Designing a user interface for such a task is complicated. The developer has to calculate all the system parameters in the program so that the users can easily control the robot in task space. When developing a user interface for task space control, two parameters have to be verified before the operation. First parameter is user input, which is verified in all interfaces designed for a robotic arm operation. Second and more important, reachability of the entered task space coordinates in joint space. For example, task space limit of a robotic arm is between -90m to 90m in X and Y direction and 0° to 180°. A user enters task space parameters within the defined limit and developed interface accepts the entered parameters. After inverse kinematics calculations are performed, it is realized the parameters cannot be reached in joint space. In this case, sending the calculated parameters for execution without rechecking them could potentially harm the system. Hence, the need of designing a user interface that is redundant and deciding whether to allow joint space control or task space control.

4.7.3 Visual Hierarchy

Figure 4.4, shows the simple page layout for collecting information, layout on the left is disorganized whereas layout on right is properly organized.

The concept of visual hierarchy plays an important part in all forms of the graphic design. Visual hierarchy focuses on the most important part of the page layout and relationships among the other displayed data. There are multiple ways a designer can draw attention to the main part of the page. For example, short but large texts, such as news headlines. In general, by making the density of important texts higher than the rest can make them

standout. Background colors, position and size of the texts are another ways of making vital information stand out. By grouping the same information, one can distinguish between different items without having to make multiple windows.

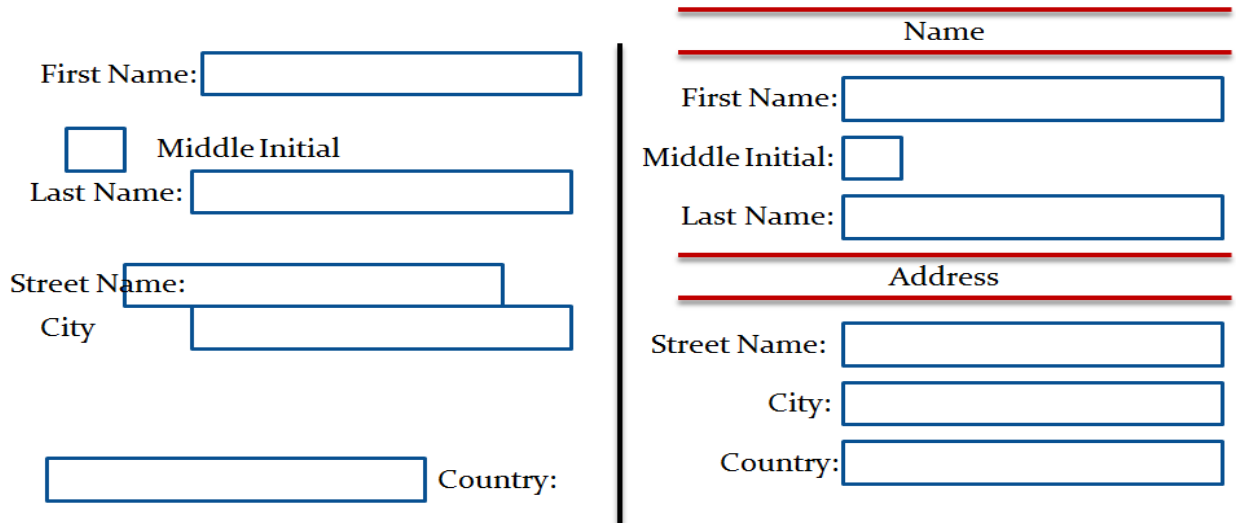

Figure 4.4: Disorganized vs. organized page layout

Visual flow deals with the way human eyes tend to follow when reading or looking for something. Humans are hardwired to read from left to right. Suddenly if a page is displayed where information is presented from right to left, it is hard to look for the information. That is why visual flow is necessary when creating an excellent user interface. Figure 4.5 exhibits the concept of visual flow and predictability in a user interface.

Last but not least, is the use of dynamic displays. Current technology has allowed the display of information in more way than one. This is a good thing if used with caution, if not it can lead to an inefficient user interface. For example, putting the most important part of the user interface into the largest subsection of window; and secondary tools are arranged around that window.

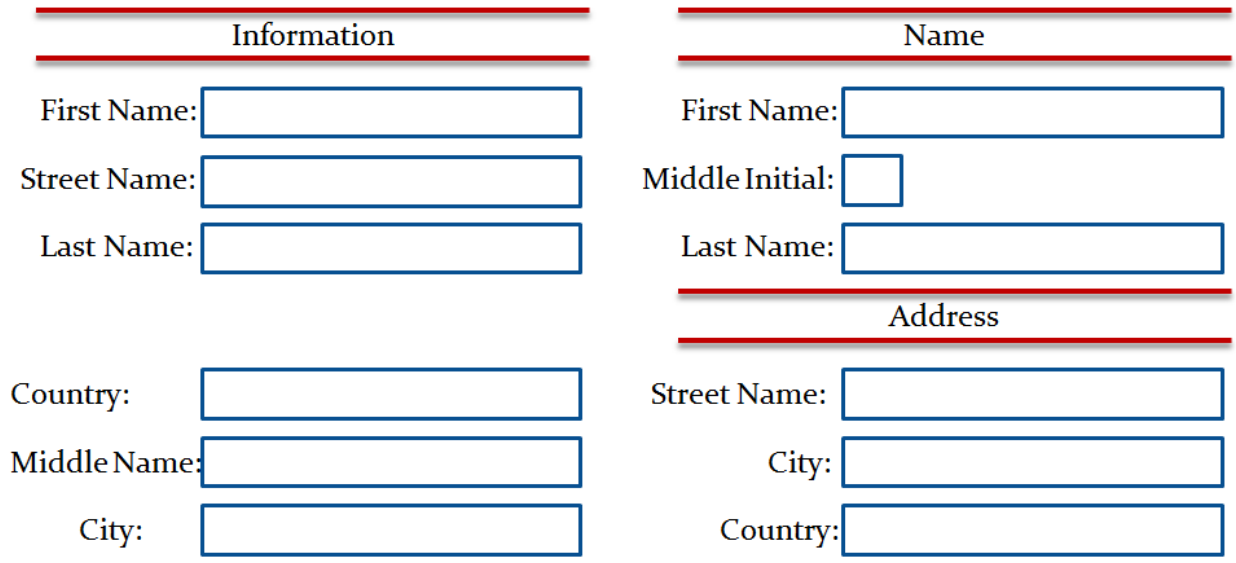

Figure 4.5: Visual flow and predictability

4.7.4 Remote Operation or Local Operation

More and more robots operated now are tele-operated robots. It is an excellent way to work without putting human lives at risk. A user interface developed for any robotic arm requires more work than the interface developed for other software. However, a user interface for a remote operation requires extra work than the interface usually developed for a robotic arm. Before examining remote operation, let us focus on the local operation of a robotic arm.

Figure 4.6 shows the user interface for Lynx robotic arm for local operation. As one can see, there is no feedback provided. Each sliding bar controls different joints and user physically looks at the robotic arm for feedback. This is usually the case with local operation and need of feedback is minimal. In most cases, during the local operation, users operating the arm are situated near the arm. Users also have immediate access to an emergency stopping mechanism. If anything goes wrong with the software sending the arm in undesired motion, the user can stop the robotic arm with a physical

emergency button. However, during the remote operation, this is simply not the case.

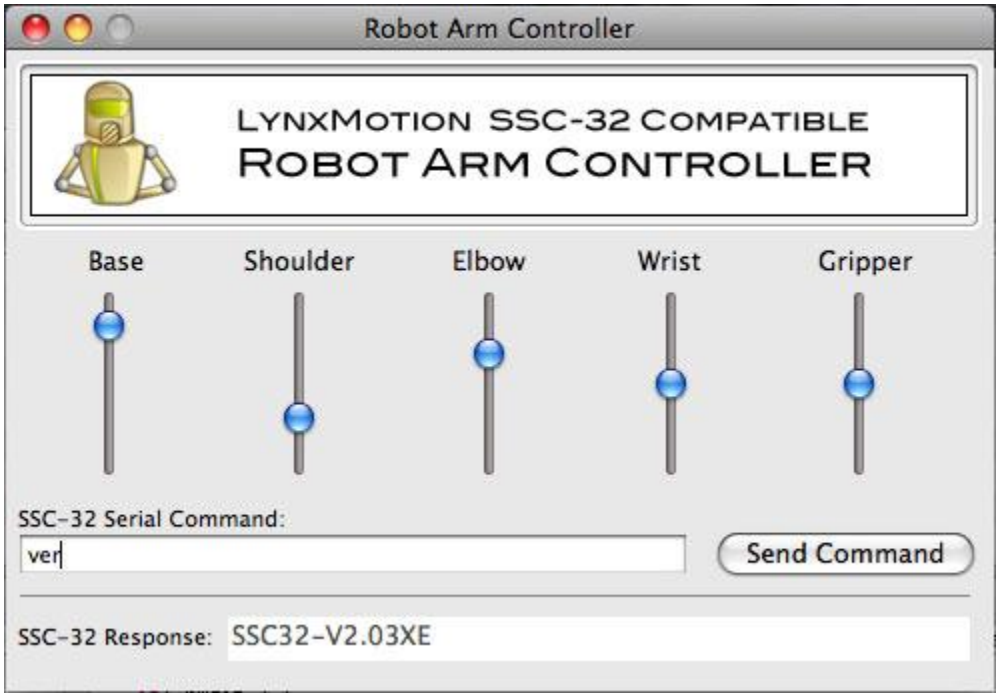

Figure 4.6: Lynx Robot Arm Controller[2]

In remote operation, users are not located near the robotic arm. In such cases, need of feedback is very important. Easiest yet most effective way of providing feedback in remote operation is the use of camera that can monitor and display the robotic arm. The question rises, what to do if the software crashes and sends the arm in an uncontrolled motion? This is where, the additional work has to be done when developing a user interfaces for tele-operation. Apart from providing necessary feedback, developer should take all the precautionary steps leading to unwanted operation. Designed interface for the tele-operation should be redundant. Each input, before sending it to a robot arm, should be checked for its validity. Lastly, emergency halt software should be deployed on a computer that controls the arm. This emergency software should be triggered when a connection

[2] http://www.nopdesign.com/images/robotarm.jpg

between the host and client is disturbed. As soon as the emergency software is triggered, it should put the robot arm in emergency halt mode stopping all the operations. This way highest level of safety of the robotic arm as well the environment it is operated in, can be achieved.

Next chapter will present in-depth discussion of the developed and implemented “SCL – Robotic Mast Graphical User Interface”.

Chapter 5: SCL – Robotic Mast Graphical User Interface

5.1 Software Structure

The developed user interface consists of multiple parts: main screen, quick tabs, task scheduling window and pop-up windows to input parameters. Deployment of the graphical user interface is done in two main parts: server side and client side. Client contains the graphical user interface developed for the control of the robotic mast. Server is deployed on the remote computer, and client is deployed on the operator's computer. For communication between the server and client, TCP/IP network protocol is used. Figure 5.1 shows the top-level software architecture of the server and client.

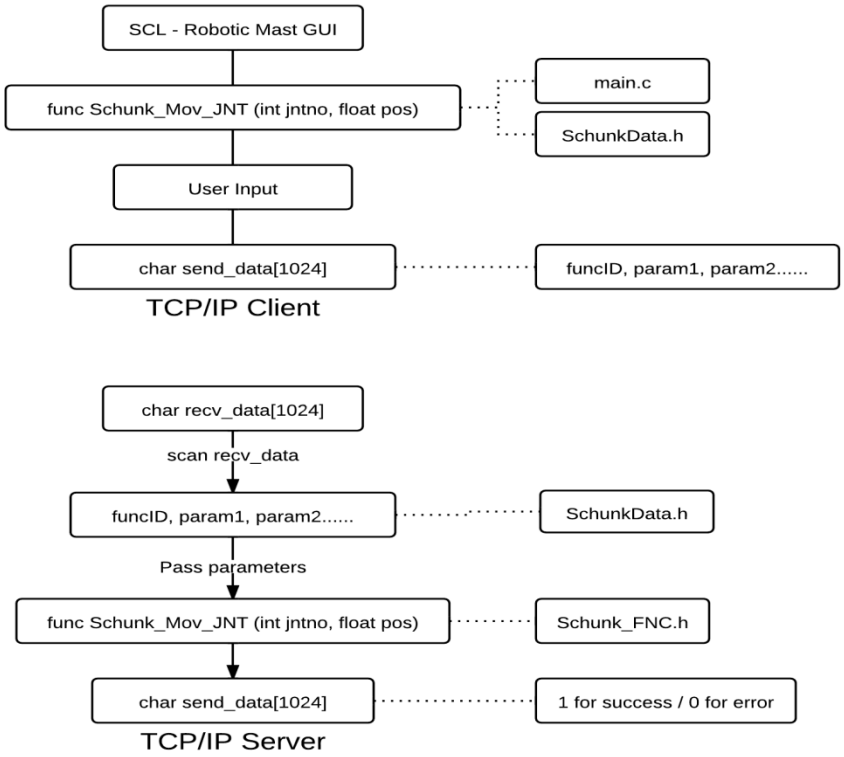

Figure 5.1: Generalized Software architecture

5.1.1 TCP/IP Server

Figure 5.2 shows the process flow chart of the deployed server.

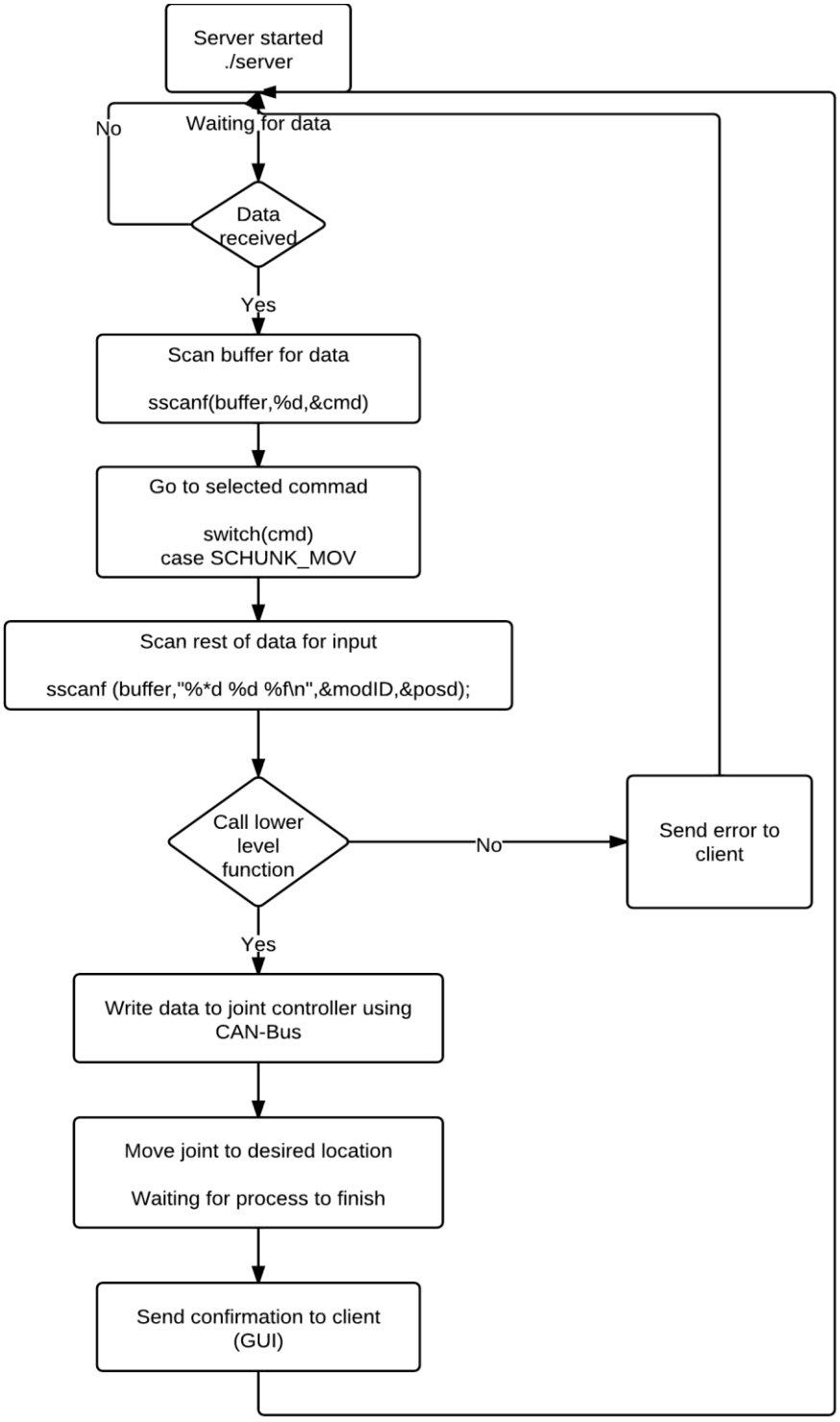

Figure 5.2: Server process flow

One of the prime requirements was the ability to operate the robotic manipulator remotely using the developed GUI. For that, it is necessary to develop a server that can handle the lower level control for the joints and communicate with the client. As shows in Figure 5.1, server includes these files: server.c, functions.h, param.h and func_cmd_no.h. File server.c is the main file, containing TCP/IP communication protocol code as well as necessary functions to control the robotic manipulator. File functions.c contains lower level control codes and func_cmd_no.c has command numbers respect to the function name. The Server runs on a remote computer because in the case of communication breakdown, server can halt the manipulator for safety.

5.1.2 TCP/IP Client

By the nature of TCP/IP protocol, server has to connect with client to receive and send data. Client is the main part of the interface development in this thesis. Program files for SCL – Robotic Mast Graphical User Interface are deployed on client. File main.c contains all the necessary information to start the user interface as well as to communicate with the server. The user interface is developed in C language with the support of GTK+; layout of the interface is designed using "*Glade – A User Interface Designer*". File taskplan.h contains necessary functions for task planning capability of the interface and inputcheck.h validates user input before proceeding further. Next section will describe the developed user interface in-depth and the use of it with the robotic manipulator available in the laboratory. Figure 5.3 shows simplified algorithm for the developed GUI.

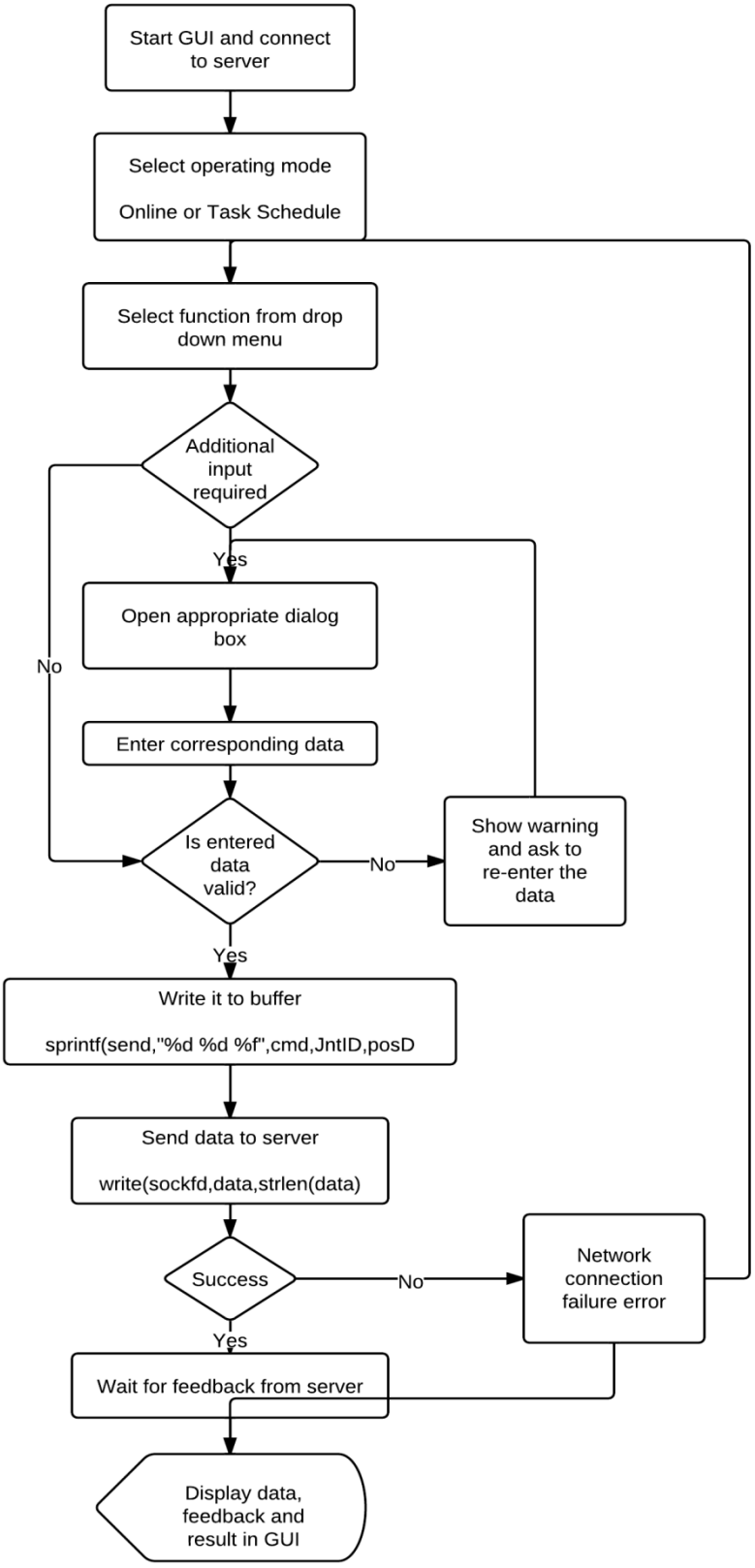

Figure 5.3: Client data process flowchart

5.2 SCL – Robotic Mast GUI

Initially, the user interface was designed using MATLAB GUIDE for the SCL – Robotic Mast. It provided a convenient way to design and implement an interface for the control of a robotic mast. However, it was soon realized that the lower level control for the robotic mast in the lab was incompatible with MATLAB. Due to this problem, another development platform was chosen. The interface is developed in the C language with libglade and GTK+ add-on, making the interface OS independent.

The built-in encoders in *Schunk* modules can provide position of each joint. This information is used as a feedback for the presented commands. The joints' positions are not displayed in real time, but they are available at any time upon request during the operation. The developed GUI is general and can be easily extended for control of the other manipulators with minor modifications.

5.2.1 Design and Operation

As soon as the GUI is started, *SCL – Network Input* shown in Figure 5.4, dialog box appears. This dialog box is used to obtain an IP address and a port number for the network connection. Once appropriate information is provided, a TCP/IP connection is established with the running server on the embedded computer. If the client fails to connect to the server, an error message is displayed in the *Main* window to remind the user of the communication error.

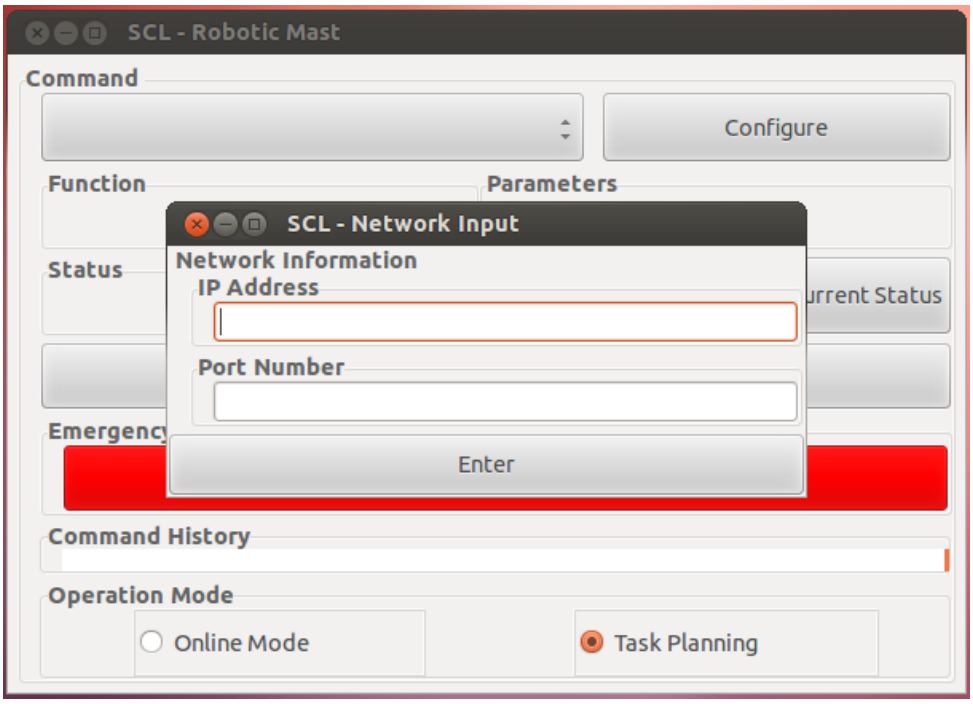

Figure 5.4: Network input dialog box

The *Main* window of the developed GUI features a drop down *Command* menu where all the functions are located, such as setting and inquiring the joints' positions, as shown in Figure 5.5. The position of each joint can be requested during the operation by simply clicking on the *Current Status* button available in the *Main* window.

The *Quick Tabs* window provides easy access to most of the commonly used functions, such as scooping samples, resetting joints and homing joints, as shown in Figure 5.7. Functions in this window do not require any parameters and they are always available as long as the TCP/IP connection is established. It should be noted that both the *Main* window and the *Quick Tabs* window are movable and can be positioned anywhere on the screen independently. In order to avoid mis-operation, each command has to be confirmed by the user before being sent to the server. When a command is selected from the dropdown *Command* menu, the command name and

essential parameters are displayed in a pop-up box, as shown in Figure 5.6, which can be resized without losing its functionality.

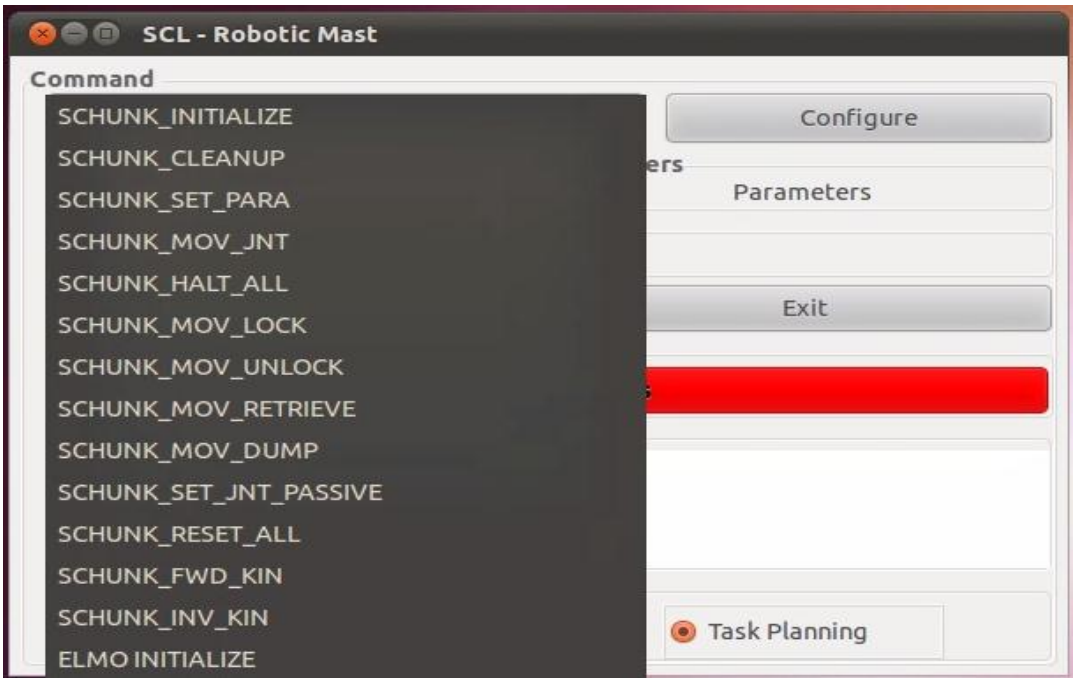

Figure 5.5: The drop down Command menu

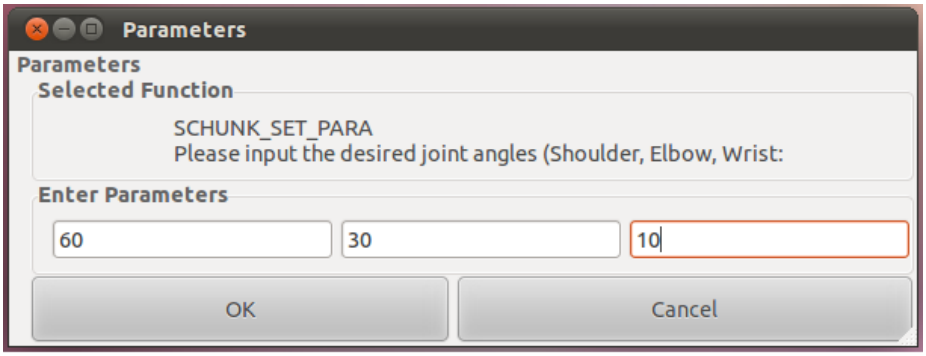

Figure 5.6: The Parameter Input box

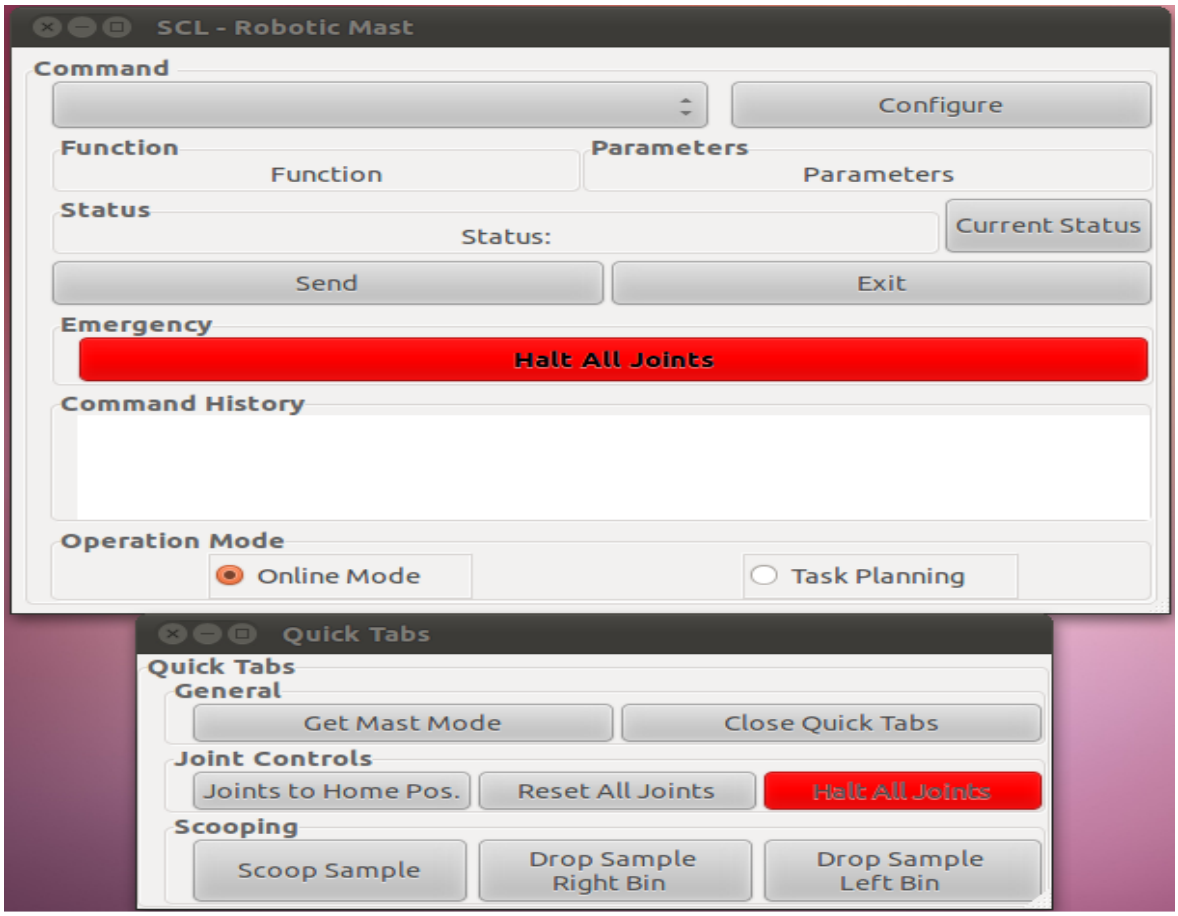

Figure 5.7: The Main screen and Quick Tabs

The *Parameter Input* box is a pop-up style dialogue box that appears after clicking the *configure* button. In this window, only the information that is related to the selected function is displayed. All the user inputs are validated to make sure they are within the physical limits. If wrong parameters are provided, a *Warning* window will pop out, as shown in Figure 5.8, and all the other windows become inactive. In such a situation, no command will be sent even if the *Send* button is pressed by accident.

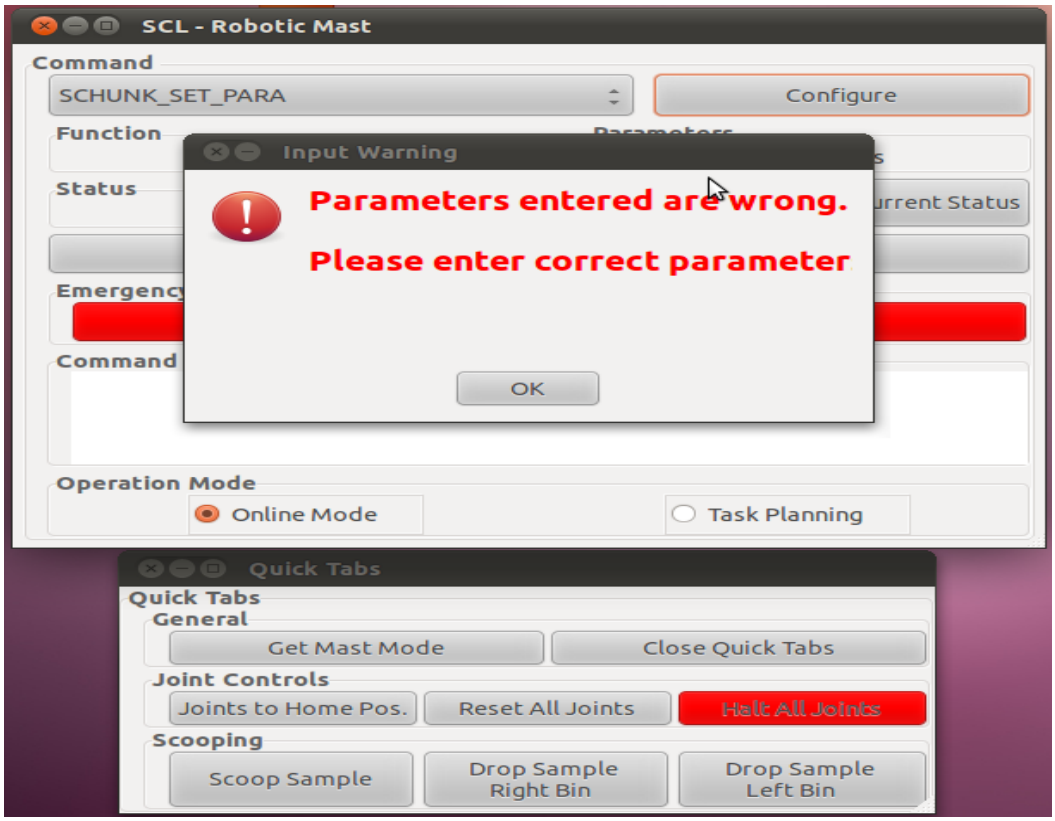

Figure 5.8: The Warning window

The selected function and the corresponding user inputted parameters are displayed in the *Main* window. The command will not be sent to the server until the *Send* button is pressed. One exception is the *Halt All Joints* command, which does not need any confirmation. Once pressed, the *Halt All* command is sent to the server immediately, which will terminate the motion of all the joints and engage the built-in magnetic brakes of the *Schunk* rotary modules. The commands that were sent to the server are listed in the *Command History* tab, which is read-only and can be exported into a text file. When an error occurs during the sample retrieval process, the server will halt the manipulator and report the error to the client, which will be displayed in the *Main* window of the developed GUI. As long as the manipulator is halted, the motion functions of the GUI will be deactivated, and the modules have to be reset before performing the next task.

The developed interface can be generalized to control the other systems with minor modifications. Description on implementing the developed GUI to control the other robotic manipulators is presented here. Figure 5.1 presents the software architecture of the developed GUI. In Figure 5.1, the *Server* is implemented with all the lower level functions and the *Schunk_FNC.h* module is designed for low-level control of the robotic manipulator. The *Schunk_FNC.h* module contains name of all the functions and the necessary input variables. The function *Schunk_Mov_JNT (int jntno, float pos)* function contains necessary lower level code to operate the manipulator. By rewriting the lower level function of another system in *Schunk_FNC*.h file, the same interface can be used to control the new system. From this point of view, the developed GUI is reusable and can be extended to control any robotic manipulators with minor modification.

5.2.2 Operation using a Joystick

One of the requirements of the developed interface was to allow manual control of the robotic manipulator using a joystick. Design of this assembly is trickier because data communication between a joystick and connected PC is much faster than the operating speed of the joints and CAN-bus communication. As discussed by Norman implementing physical constraint can help avoid mis-operation of the system. Upper and lower limit of the joystick is associated with the upper and lower limit of the joints, respectively. Homing position of the joystick corresponds to zero position of the each joint. The joystick can be used anytime once program has been started and successfully connected to the server. In order to evade unintended operation of the manipulator, users must press and hold a trigger button associated with each joint, Table 5.1, while operating the joystick.

Table 5.1: Interfacing joystick button with manipulator joints

Joystick Button ID	Joint ID
Button No. 2	6 – Shoulder
Button No. 3	7 – Elbow
Button No. 4	8 – Wrist

5.2.3 Task Scheduling

Any task using a manipulator can be performed in two ways: online mode and offline mode. Currently, online mode is being used by the majority of the users [32]. With consideration of the low efficiency of the online mode, author introduces task scheduling into developed GUI. The flowchart for task scheduling is shown in Figure 5.10.

Figure 5.9 shows the task scheduling dialogue box. Once the *Execute* button is pressed, interface selects the first task in the stack and sends it to the server. Upon receiving the confirmation of the task completion, the following task is send, and so on. The order of tasks to be executed is determined by the operator through the GUI, instead of autonomously by the low-level controller. The system administrators are also allowed to plan tasks by simply entering the *command ID* and *parameters* in the entry box.

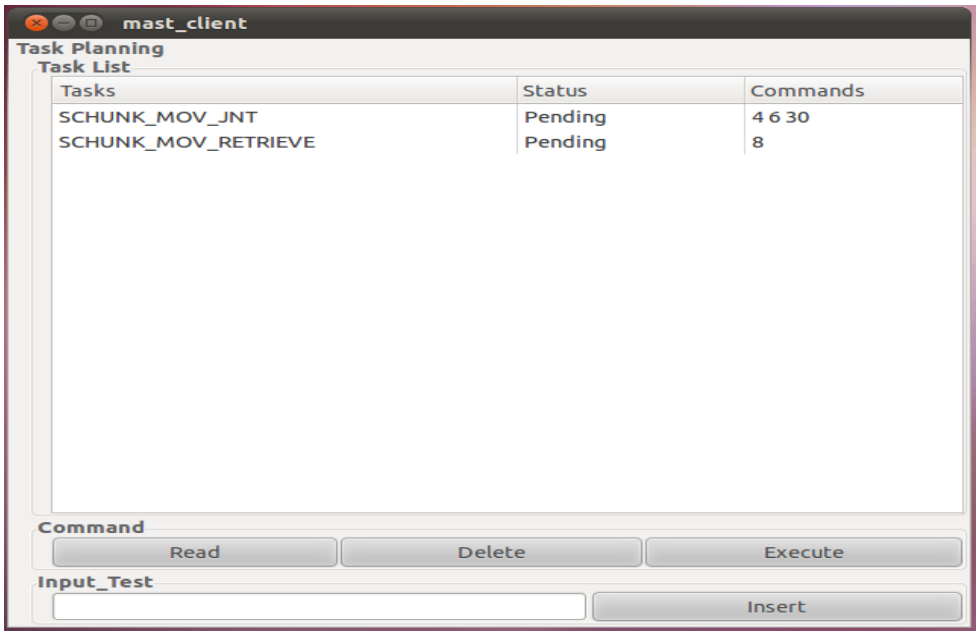

Figure 5.9: Task schedule dialogue box

With task scheduling, the users can use the same interface presented above to plan the multiple tasks in *offline* mode. In *offline* mode, the interface will connect to the server and the task starts to execute, as soon as the user confirms one command. In the event of connection failure, the user will be warned and sever will finish the task before halting the manipulator.

Developed interface will also be used for testing of new control algorithms that requires extensive data from joints' sensors. As soon as a task is started, developed task scheduling algorithm will start collecting position data for each of the joints. The collected data is saved in new text file on the client computer for easy access. In order to make sure no data is lost or over written when new operation starts, new text file with unique identity is created. Figure 5.11 shows partial data collected during the operation.

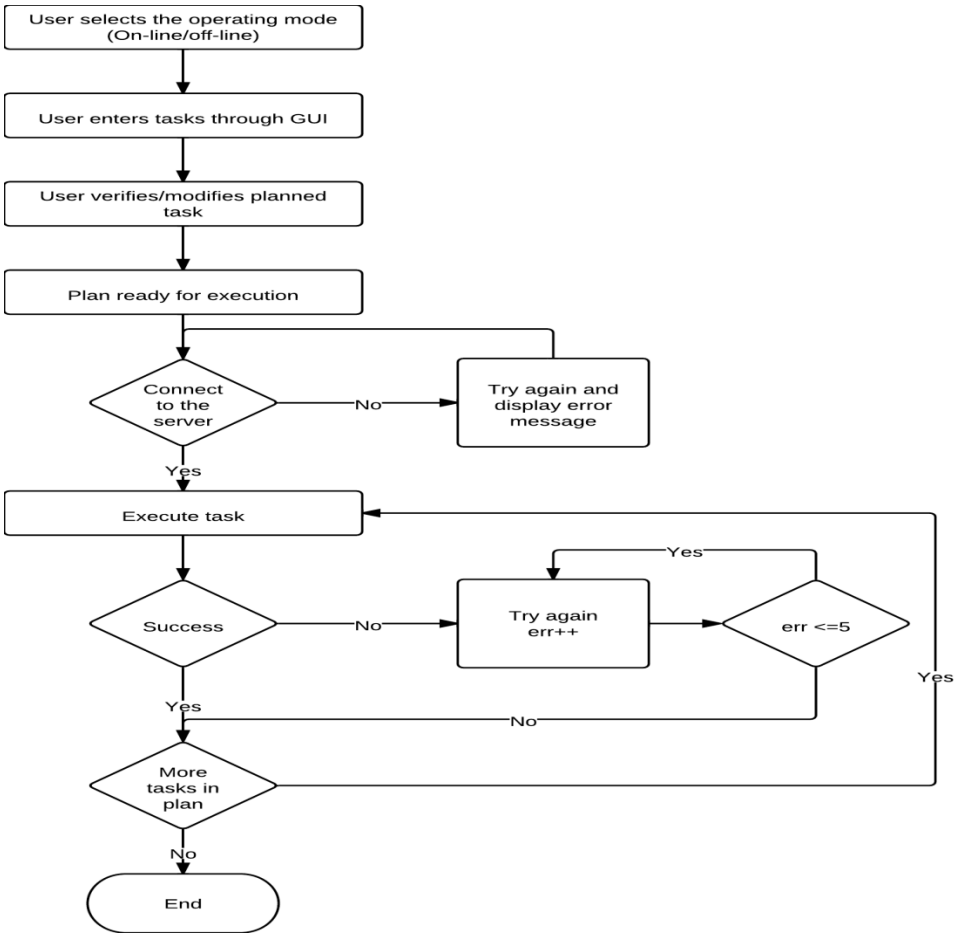

Figure 5.10: Task scheduling and execution algorithm

```
28.676741 51.363918 -19.900801
28.908264 51.157177 -19.900801
29.109457 50.967068 -19.900801
29.330219 50.796200 -19.900801
29.481850 50.645546 -19.900801
29.625656 50.505657 -19.900801
29.769459 50.376526 -19.900801
29.855871 50.268265 -19.900801
29.992178 49.975113 -19.902601

DONE
30.013699
30.014023
30.014351
30.014351
30.014351
30.014351
30.014351
```

Figure 5.11: Sample position data

5.2.4 Function Mapping and Process Flow

As explained before, each command has its own unique dialog box. Some commands directly call the lower level function to perform the needed actions. Whereas, certain commands are combination of one or more lower level functions. Below, each command and its associated lower level function(s), process and the action performed by the command will be explained, Table 5.2 to Table 5.9. After that, the process flowchart of the most used commands will be presented for better understanding of the programing structure.

5.2.4.1 SCHUNK_INITIALIZE

Table 5.2: Function mapping for SCHUNK_INITIALIZE

Function Name	**Schunk_Initialize**
Function ID	1
Input parameters	None
Lower level command used	int schunk_Init() int CAN_Init() int Schunk_Get_Jnt_State () int Schunk_Set_Jnt_Vel() int Schunk_Set_Jnt_Acc() int Schunk_Mov_Reset_Jnt() {If needed}
Purpose of the function	• Initialization of the CAN-BUS communication between server (PC/104) and Schunk joints • Checking the error code of each joint • If any joint has an error such as activation of halt command,

	unable to set velocity or acceleration than reset the joint.
Return	**0** for success **1** for error

5.2.4.2 SCHUNK_CLEANUP

Table 5.3: Function mapping for SCHUNK_CLEANUP

Function Name	**Schunk_Cleanup**
Function ID	2
Input parameters	None
Lower level command(s) used	int Schunk_Cleanup() int CAN_Cleanup() int Schunk_Mov_Halt_All() {If needed}
Purpose of the function	• Closing the CAN-BUS communication before clearing any error in the joint for safe shutdown of the robot arm • Checking the error code of each joint • If any joint has error such as activation of halt command, unable to set velocity or acceleration than reset the joint.
Return	**0** for success **1** for error

5.2.4.3 SCHUNK_MOV_JNT

Table 5.4: Function mapping for SCHUNK_MOV_JNT

Function Name	Schunk_Mov_Jnt

Function ID	4
Input parameters	• Joint ID (int jntid) • Desired angle (float posd)
Lower level command(s) used	int Schunk_Mov_Jnt_Pos() int CAN_Init()
Purpose of the function	• To move individual joint to desired location • After performing inverse kinematics of the robot arm, use this command to go to desired position in task space.
Return	**0** for success **1** for error

Figure 5.12 shows the process flowchart for function SCHUNK_MOV_JNT. Process flow in the developed GUI for all the commands with the exception of SCHUNK_INV_KIN is relatively same.

5.2.4.4 SCHUNK_SET_PARAM

Table 5.5: Function mapping for SCHUNK_SET_PARAM

Function Name	Schunk_Set_Param
Function ID	3
Input parameters	Desired position angle of all the joints (float shoulder, float elbow, float wrist)
Lower level command(s) used	int Schunk_Mov_All_Joints() int Schunk_Mov_Jnt_Pos() int Schunk_Wait_Jnt()
Purpose of the function	• Moving all the joints together with the use of single function • Create shortcuts for complicated

	tasks such as sending robot arm to home position
Return	**0** for success **1** for error

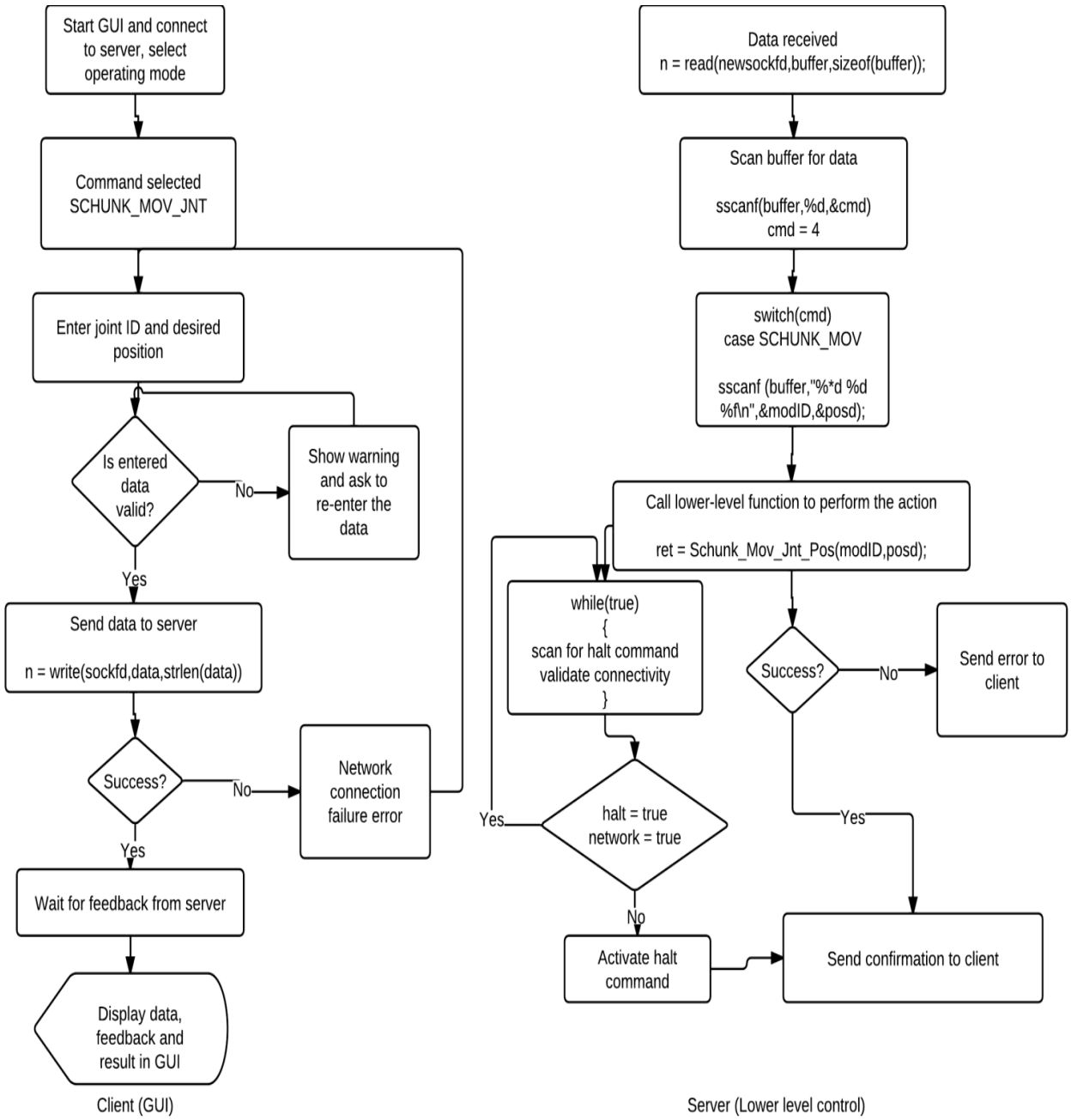

Figure 5.12: Process flow of SCHUNK_MOV_JNT

5.2.4.5 SCHUNK_HALT_ALL

Table 5.6: Function mapping for SCHUNK_HALT_ALL

Function Name	Schunk_Halt_All
Function ID	5
Input parameters	None
Lower level command(s) used	int Schunk_Mov_Halt_All() int CAN_Init()
Purpose of the function	• Halting the robot arm in case of emergency • Same function is used to halt robot when connection to the server is lost, no action is require by the user in case of lost connection
Return	**0** for success **1** for error

5.2.4.6 SCHUNK_MOV_RETRIEVE

Table 5.7: Function mapping for SCHUNK_MOV_RETRIEVE

Function Name	Schunk_Mov_Retrieve
Function ID	8
Input parameters	None
Lower level command(s) used	int Schunk_Sample_Retrieve() int Schunk_Mov_All_Joints ()
Purpose of the function	• Scooping of the sample (critical task) • Quicktabs box also includes shortcut for this function
Return	**0** for success **1** for error

Function Schunk_Mov_Dump uses the same structure and lower level commands as the function Schunk_Mov_Retrieve. The purpose of this function is to dump the scooped samples in the pre-designated area. Shortcut for Schunk_Mov_Dump is also presented in the Quick Tabs window for easy access.

5.2.4.7 SCHUNK_INV_KIN

Table 5.8: Function mapping for SCHUNK_INV_KIN

Function Name	Schunk_Inv_Kin
Function ID	13
Input parameters	Desired position in task space (float x, float y, float theta)
Lower level command(s) used	int Schunk_Mov_All_Joints() int Schunk_Mov_Jnt_Pos() int Schunk_Wait_Jnt()
Purpose of the function	• Performing inverse kinematics of the robot from the given task space input • Sending robot arm to the given task space position if everything valid • Warning the user if the position is not reachable in joint space
Return	**0** for success **1** for error

Figure 5.13 shows the process flow for SCHUNK_KIN_INV command, which is used for task space control of the manipulator. Difference between this command and rest is the retrieval of *inv_kin* function. Function *inv_kin* performs inverse kinematics as explained in Chapter 3. Inverse kinematics

computations depend on the configurations of the manipulator. This function is written in separate header file *inv_kin.h*. By rewriting the code for new system, developed interface can be used with the new system.

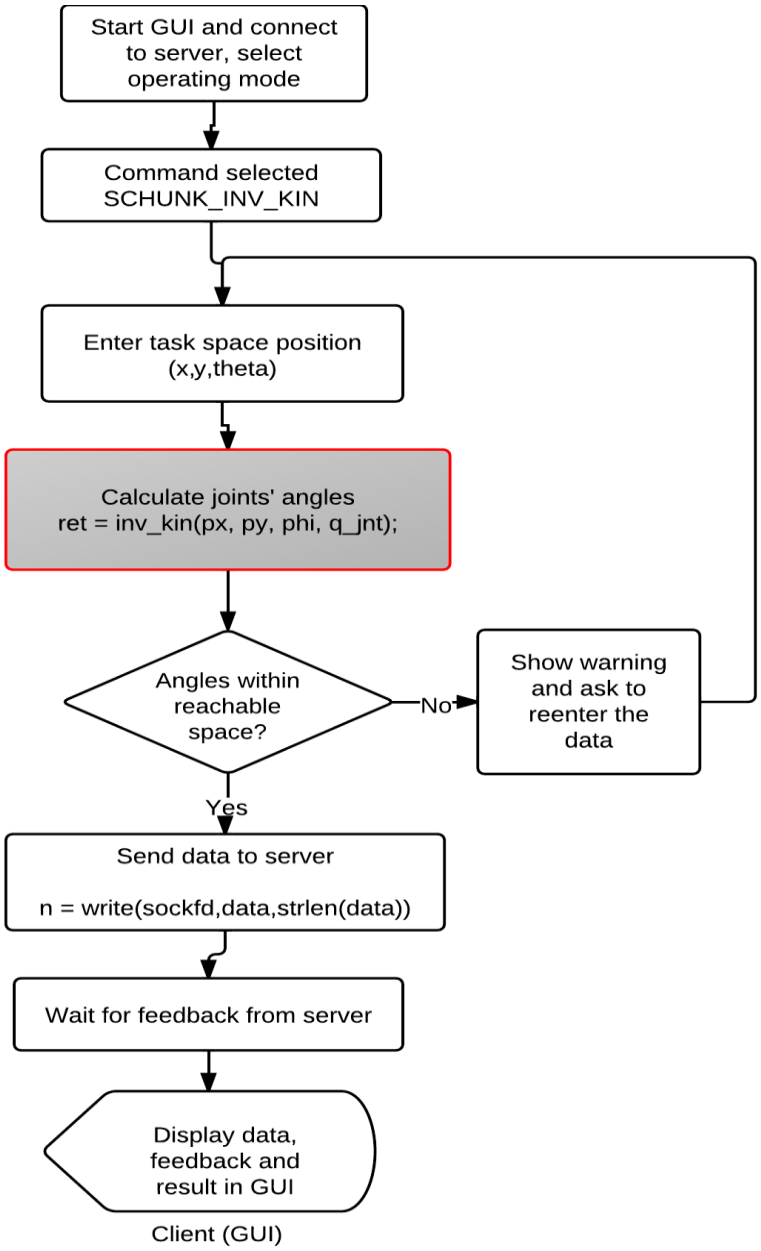

Figure 5.13: Process flow for task space control

5.2.4.8 SCHUNK_GET_JNT_POS

Table 5.9: Function mapping for SCHUNK_GET_JNT_POS

Function Name	Schunk_Get_Jnt_Pos
Function ID	16
Input parameters	Joint ID (int jntid)
Lower level command(s) used	int CAN_Init()
Purpose of the function	• To inquire about the position of the joint in joint space
Return	**0** for success **1** for error

5.3 Experiments

The robotic manipulator used in this experiment consists of three rotary *Schunk PRL* modules (*PRL 60*, *PRL 80* and *PRL 80* for the wrist, elbow and shoulder joints, respectively) and a scoop. These *PRL* modules are connected with customized robotic links. The PRL modules are connected electronically to a CAN-Bus interface, which is controlled by an embedded computer running Linux.

Each command has been tested individually as well as using the task-scheduling feature. Initially, each joints were moved to several different positions using the interface. During this test, all the joints were controlled using joint space functionality of the interface. After the successfully completion of this test, the joints were controlled in task space. Afterwards, all three joints were operated together multiple times to ensure the reliability of the software. During this particular test, network connection was intentionally interrupted to assess the emergency halting software. For the duration of this test, behaviour of the interface was as expected.

After the successfully testing the joint movement commands, more complex task of sample scooping and dumping was experimented using *Quick Tabs* as well as drop down menu. Snapshots for the sample retrieval experiments are shown in Figure 5.14.

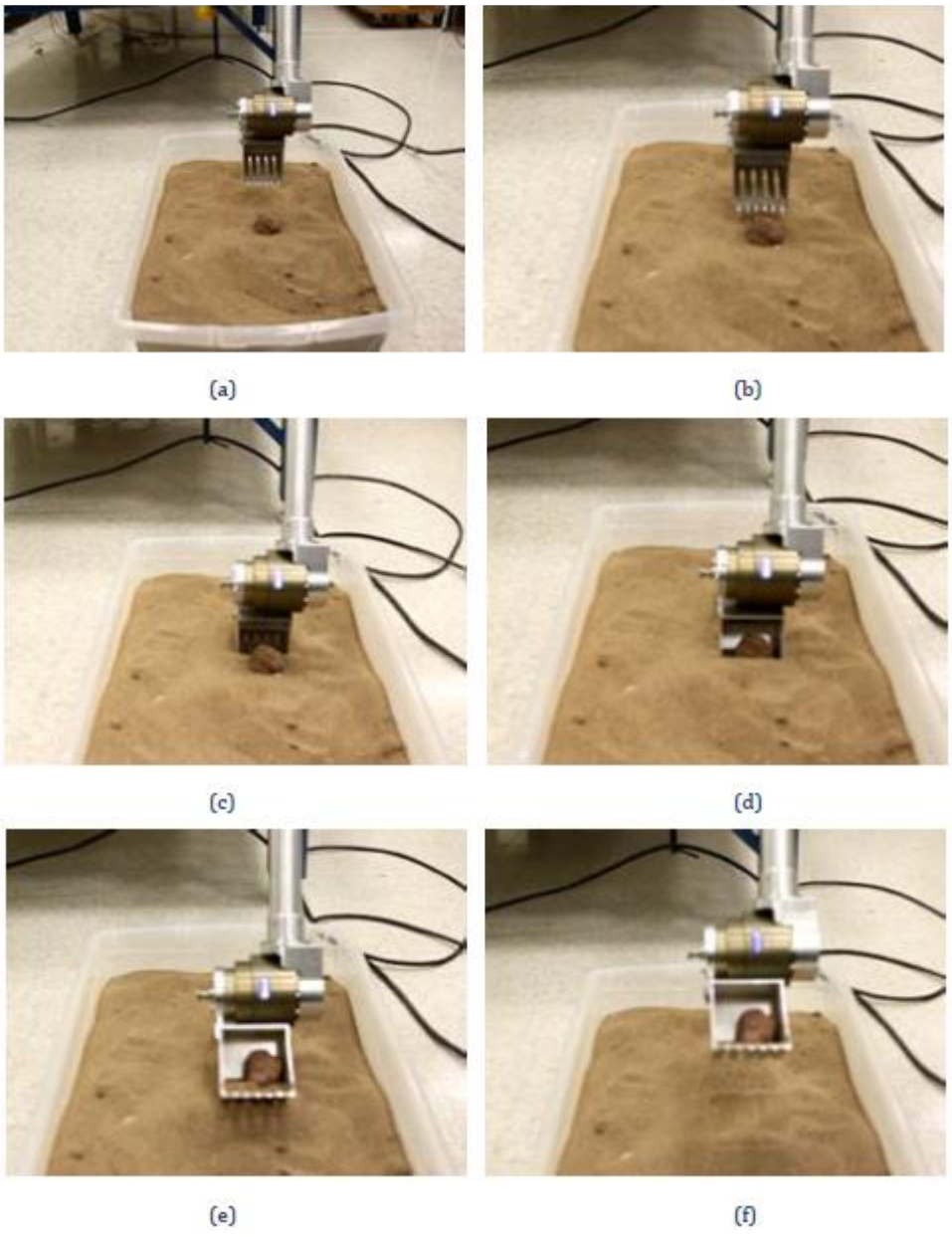

Figure 5.14: Snapshots for sample retrieval

Finally, manual operation of the manipulator using a joystick was verified. Software structure of the developed interface allows control the other

systems with minor modifications. Lower level code and a parameter input box was modified to test the interface with the robotic manipulator made of six rotary joints. Controls of individual joints were verified, scooping and dumping functions were unavailable due to different configuration of the manipulator. Videos of all the experiments are included in the DVD attached with this thesis.

Overall, performance of the developed interface exceeded all the expectations.

5.4 Comparative Study

In this subsection, the developed GUI is compared with several robotic tele-operation interfaces reported in the literature. As summarized in Chapter 1, the major requirements of the interface are to allow comfortable operation of robotic arm and to enable efficient task scheduling. Some of the interfaces reported in the literature have several good features, such as enabling online operation, displaying all the available functions together, allowing remote operation, and so on. However, task scheduling was not paid enough attention. For example, the GUI for Astronaut Interface Device [10]and the GUI developed by Vajnberger *et al* for remote control of a 5-DOF robotic arm, allow only one task to be performed at a time. RoSE and MotoSim EG [32] integrate all the necessary functions for operation, but they need input commands to schedule a task. For example, "MOV" has to be entered by the operator to move to robot to a particular configuration. With the developed GUI, the user can perform one task at a time or schedule multiple tasks using the same interface. The comparison is summarized in Table 5.10.

Table 5.10: A Comparison of Reported GUIs

Features	AID [10]	RoSE [9]	Vajnberger *et al* [27]	MotoSim [32]	Proposed GUI
Task scheduling		✓		✓	✓
Online operation	✓		✓		✓
Display all available functions together		✓	✓	✓	✓
Ability to use GUI to schedule tasks					✓
Sub-display for each function	✓				✓
Remote operation	✓	✓	✓	✓	✓
Use command line to schedule tasks		✓		✓	✓

Next chapter concludes the thesis with several remarks on future work.

Chapter 6: Conclusion

6.1 Conclusion

In this thesis work, a graphical user interface has been developed for the control of a robotic manipulator capable of sample acquisition. The developed GUI is easy to use and the operator does not need to have any prior technical knowledge of the system. Furthermore, this thesis simplifies the process of a graphical user interface design for the developers lacking the knowledge of human cognitive behaviour by focusing on users and their requirements as well as consistency in design. This thesis also infuses the concept of simplicity, feedback, and an appropriate page layout for GUI.

In chapter 2, literature related to this thesis is extensively reviewed to develop a GUI that is capable of controlling the robotic manipulator autonomously and manual operation using a joystick. Furthermore, it was determined to develop a method to review all the commands sent to the system for execution. Having such a feature will provide an easy access to the command history in case of a failure. In chapter 3, inverse kinematics analysis of the manipulator is presented. This analysis was required since the developed interface has a functionality of operating the manipulator in task space.

Chapter 4 presents essential guidelines for a user interface design. In order to develop an effective GUI, it is of utmost important to focus on users and not to take away the control from them. The designed interface should also be visually pleasing and should exhibit visual hierarchy. Use of texts, colors and input methods should be consistent throughout the GUI. For robotic

manipulator, users should have the option of controlling the manipulator in both joint space and task space. Lastly, emergency halt software should be deployed on a computer that controls the manipulator. This emergency software should automatically trigger when a connection between the host and client is disturbed.

Details of the designed user interface are explained in chapter 5. Combinations of control functions have been defined to alleviate the operator's efforts in robotic arm manipulation. The developed GUI is easy to use, and the operator does not need to have any prior technical knowledge of the system. Since the GUI checks and validates all the user inputs before sending a command to the server, there is a minimal risk of mis-operation. The developed GUI is capable of monitoring and collecting position data of each joint for further analysis. Task scheduling is also implemented in the developed GUI allowing a user to plan and execute multiple tasks. Furthermore, read-only command history is displayed in the *Main* Window to help the diagnosis of an unexpected behavior of the robot manipulator if necessary. Implemented intelligent server/client architecture is able to handle unexpected events such as, connection failure, and able to halt the manipulator to safety in case of emergency. In order to verify the effectiveness of the presented GUI, it has been successfully tested on two different robotic manipulators available in the laboratory.

In the end, the new interface promises to give a unique and comfortable experience to the user when operating the manipulator.

6.2 Future Work

The following are suggestions for future work on SCL – Robotic Mast Graphical User Interface software:

- During task scheduling, adding an animation that enables the user to visualize the task before it is executed will improve the functionality of the interface.
- Currently, only position data of the joints are displayed and collected; adding the ability to collect velocity and acceleration data can be helpful and can enrich the interface.
- To improve the intuitive feedback of the developed GUI, a vision camera system should be introduced.
- Addition of path planning and fault detection algorithm for the manipulator can also enhance the developed GUI.

References

[1] Anderson, R. J., & Spong, M. W. (May 1989). Bilateral control of teleoperators with time delay. *IEEE Trans. on Automatic Control*, vol.34, pp. 494-501.

[2] Bukchin, J., Luquer, R., & Shtub, A. (March 2002). Learning in tele-operations. *IIE Transactions (March 2002)*, pg. 245-252 .

[3] Cho, C., Song, J., Kim, M., & Hwang, C. (April 2004). Energy-based control of a passive haptic device. *IEEE International Conference on Robotics and Automation*, (pp. pp. 292-297). New Orleans, LA.

[4] Cresnik, P. (2010). *CANSAT Construction Documentation and Guide.* Toronto: Ryerson University.

[5] Desai, K., Liu, Y., & Liu, G. (2012). A Graphical User Interface for Tele-operated Robotic Sample Acquisition. *IEEE International Conference On Mechatronics and Automation*, (pp. pg. 1202-1207). Chengdu, China.

[6] Galitz, W. O. (1997). *Essential Guide to User Interface Design.* New York, Toronto: Wiley Computer Pub.

[7] Goldstain, O., Ben-Gal, I., & Bukchin, Y. (October 2011). Evaluation of telerobotic interface components for teaching robot operation. *IEEE Trans. on Learning. Technologies*, vol. 4, pp. 365-376.

[8] H. Igarashi, I., Takeya, A., Kubo, Y., Suzuki, S., Harashima, F., & Kakikura, M. (November 2006). Human adaptive GUI design for teleoperation system. *31st Annual Conf. IEEE Industrial Electronics Society*, (p. pp. 6).

[9] Hartman, F., & Maxwell, S. (September 2004). Driving the mars rovers. *Linux Journal*.

[10] Hirsh, R., Simon, C., Tyree, K., Ngo, T., Mittman, D., Utz, H., et al. (September 2008). Astronaut interface device (AID). *AIAA SPACE 2008 Conference and Exposition*, (pp. AIAA-2008-7910). San Diego, California.

[11] Jazar, R. N. (2010). *Theory of Applied Robotics.* Springer.

[12] Johnson, J. (2010). *Designing with the Mind in Mind.* Morgan Kaufmann.

[13] Kemp, C. C., Edsinger, A., & Torres-Jara, E. (2007). Challenges for robot manipulation in human environments. *IEEE Robotics and Automation Magazine*, *14*, pp. pp.20-29.

[14] Koch, J., Reichardt, M., & Berns, K. (September 2008). Universal web interfaces for robot control frameworks. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. pp.2336-2341).

[15] Logitech. (2012). *Logitech Attack™ 3 Joystick*. Retrieved July 10, 2012

[16] Luostarinen, R., Manner, J., Ma¨a¨tta¨, J., & Ja¨rvinen, R. (November 2010). User-centered design of graphical user interfaces. *IEEE Military Communications Conference*, (pp. pp. 50-55).

[17] Norman, D. A. (1990). *The design of everyday things.* New York: Doubleday Books.

[18] PC/104 Consortium. (2012). *PC/104 Consortium - History*. Retrieved July 10, 2012

[19] Redmond-Pyle, D., & Moore, A. (1995). *Graphical User Interface Design and Evaluation.* Trowbridge: Prentice Hall.

[20] Schenker, P. S. (2006). Advances in rover technology for space exploration. *IEEE Aerospace Conference*, (p. pp.26). Big Sky, MT.

[21] SCHUNK Gmbh & Co. (n.d.). *SCHUNK. rotary modules PRL*. Retrieved January 23, 2012

[22] Shneiderman, B., & Plaisant, C. (2009). *Designing he User Interface: Strategies for Effective Human–computer Interaction.* Addison-Wesley.

[23] Song, T., Park, J., Chung, S., Kwon, K., & Jeon, J. (July 2008). Intelligent User Interface for Human-Robot Interaction. *INDIN 2008. 6th IEEE International Conference*, (pp. pp.1463-1468).

[24] Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2005). *Robot Modeling and Control.* Wiley.

[25]Stone, D., Jarrett, C., Woodroffe, M., & Minocha, S. (2005). *User Interface Design and Evaluation.* San Francisco: Morgan Kaufmann Publishers.

[26] Tidwell, J. (2011). *Designing Interfaces.* Sebastopol, CA: O'Reilly.

[27] Vajnberger, V., Terzimehic, T., Silajdzic, S., & Osmic, N. (n.d.). Remote control of robot arm with five DOF. *MIPRO, 34th International Convention*, (pp. pp. 1707-1711).

[28] Volpe, R. (n.d.). *Robotics - Application*. Retrieved February 25, 2012, from http://www-robotics.jpl.nasa.gov/applications/applicationArea.cfm?App=11

[29] Wang, Y., Yan, K., Sun, G., & Lou, P. (n.d.). *Serial Communication in DNC Information Systems.* Nanjing, China: Moxa Technologies.

[30] Want, R. (2010). iPhone: Smarter than the average phone. *IEEE Pervasive Computing*, vol. 9, no. 3, pp. 6-9.

[31] Weinschenk, S., Jamar, P., & Yeo, S. C. (1997). *GUI design essentials.* Wiley.

[32] Yu, H., Shan, J., & Zhu, X. (2011). Off-line programming and remote control for a palletizing robot. *IEEE International Conference on Computer Science and Automation Engineering*, (pp. pp. 586-589).

A Appendix

Figure A.1, shows the developed interface in early stages of the thesis. This interface was designed in MATLAB.

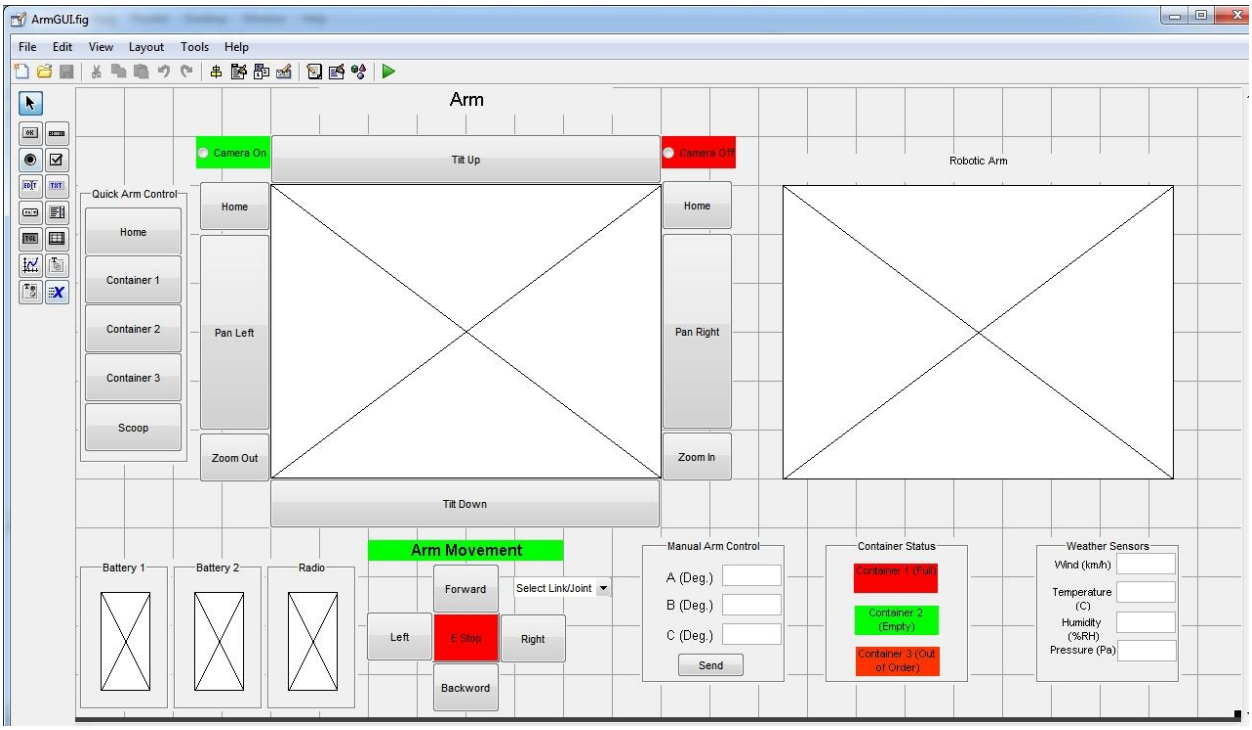

Figure A.1: Robotic Arm Control Screen

One button is dedicated to scooping; scooping can also be done manually. Status box shows the status of the container. Three angles A, B and C are for manual control of the arm that is discussed in the next section.

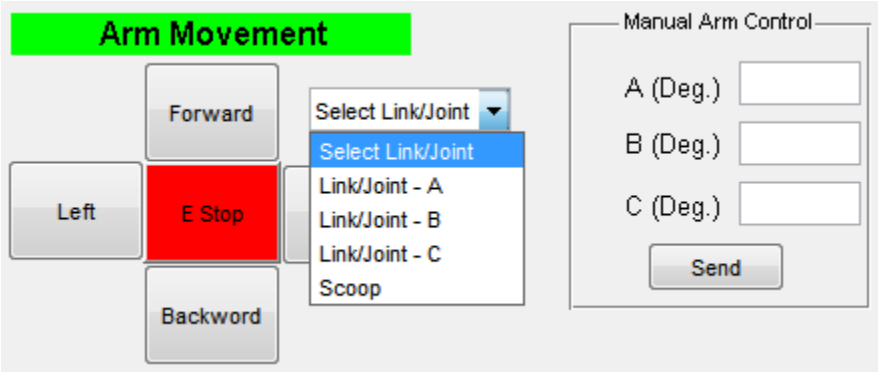

Figure A.2: Arm movement options

Figure A.2 shows two ways the arm can be controlled. One is using push buttons and another by entering specific angles for each joint.

A.I atan2(y,x)

For any real number arguments *x* and *y* not both equal to zero,atan2(*y*, *x*) is the angle in radians between the positive *x*-axis of a plane and the point given by the coordinates (*x*, *y*) on it. The angle is positive for counter-clockwise angles (upper half-plane,*y* > 0), and negative for clockwise angles (lower half-plane, *y* < 0).

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \text{undefined} & y = 0, x = 0 \end{cases}$$

A.II Types of Robots

i) Cartesian Robot

A Cartesian robot, also known as Cartesian coordinate robot shown in Figure A.3, has three links that coincides with Cartesian coordinate system; hence, each link is perpendicular to one another. These types of robots are mostly used in Computer Numerical Control (CNC) machine.

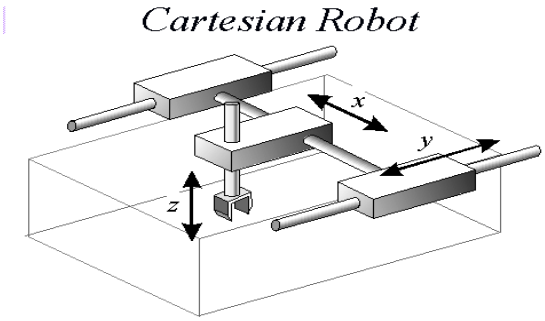

Figure A.3: Cartesian Robot[3]

ii) Cylindrical Robot

Cylindrical robots, as shown in Figure A.4, are the rarest now a days due to their stationary position. As the name suggests cylindrical robot is the robot whose axes form a cylindrical coordinate system. Generally, they are used for spot wielding and handling machines tools.

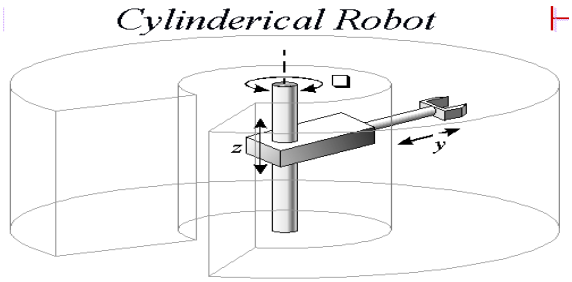

Figure A.4: Cylindrical Robot[4]

iii)Polar Robot

Polar robot is similar to cylindrical robot with only difference being, polar robot's the ability to rotate in two different directions along its main axis. Lastly, the joint moves in translation so that it forms polar coordinates.

[3] http://prime.jsc.nasa.gov/ROV/images/cartesian.GIF
[4] http://prime.jsc.nasa.gov/ROV/images/cylindrical2.GIF

iv) Articulated Robot

An articulated robot, shown in Figure A.5, is a robot consisting of rotary joints. They range from simple two-jointed system to complex eight or more interacting joints. They are typically used in automobile industries for large assembly operation and for spray painting.

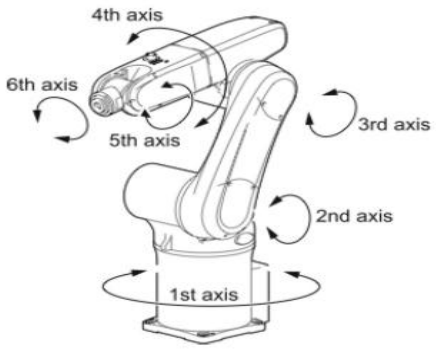

Figure A.5: Articulated Robot[5]

v) Parallel Robot

Parallel robot consists of a fixed base platform connected to an end-effector platform by means of a number of links. These links consist of an actuated prismatic joint, allowing only traction or compression movement and due to this a higher accuracy and lightweight system is achieved. Since the end-effector is connected to the base via multiple kinematic joints, movement in any two joints would cause entire system to move, making it a closed loop system. These types of robots are used for cockpit flight or automobile simulator.

[5] http://www.processonline.com.au/articles/36410-Packaging-automation-trends-using-small-assembly-robots-in-upstream-packaging-processes

vi) SCARA (Selective Compliance Assembly Robot Arm) Robot

Design of the SCARA robot is much like a shoulder and elbow held perfectly parallel to ground. Basic configuration of a SCARA is a four degree of freedom movement. A SCARA has full range of motion on XY direction and has rigid Z-axes. SCARA robots are known for their speed, large workspace, payload capacity, and excellent repeatability. SCARA robots are more expensive and controlling software requires inverse kinematics for interpolated movements. They are generally used perform precise jobs repeatedly, such as pick and place work, installing pin, application of sealant.

Above described robots are typically used in the manufacturing industry. Robots also perform important tasks outside of the manufacturing industries. Their tasks include but are not limited to hazardous duty service, CAM/CAD design and prototyping, medical applications, fighting fire and military warfare.

A.III mast_client.cpp

Code presented here contains necessary functions to run GUI, callback functions as well as TCP/IP client. In the main code, multiple headerfiles, specifically developed for this software, have been used. Brief description of the header files is presented here. Code from the headerfiles is also presented in the appendix.

inputcheck.h – Check and validate all the input entered by the user before sending it to server.

task_signals.h – Start the task scheduling interface and allow users to manipulate the order of the task. Code manual task scheduling is written in this file.

inv_kin.h – Inverse kinematics calculations for the manipulator, used in the experiement of this thesis, is coded in this file.

testjs.h – Necessary code for operation of joystick and interfacing of joystick between GUI and hardware is presented in this file.

```
/*******************************************************
* Name: mast_client.cpp                                *
*                                                      *
* Func: Graphical User Interface for Client Software    *
* Date: June 15, 2012                                  *
* Auth: Karan Desai                                    *
*                                                      *
*******************************************************/
// Current Arm Pos: 0, -80, -40

#include <stdio.h>
#include <iostream>
#include <string.h>
#include <time.h>
#include <fcntl.h>
#include "SchunkData.h"
#include "inputcheck.h"
#include "task_signals.h"
#include "inv_kin.h"
#include "testjs.h"

#include <gtk/gtk.h>
#include <stdlib.h>

#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <math.h>
```

```
//Function to check if the input is numeric or not

int isNumeric (const char * s)
{
   if (s == NULL || *s == '\0')
     return 0;
   char * p;
   strtod (s, &p);
   return *p == '\0';
}

void error(const char *msg)
{
   perror(msg);
   exit(0);
}

void signal_callback_handler(int signum)
{
            printf("Caught signal %d\n",signum);
            // Cleanup and close up stuff here

            // Terminate program
            exit(signum);
}

//------------Write data file in current directory
char cwd[200];
char dir[1024];
struct tm *current; //pointer to array holding the current time

//-----------End data file---------------------

//End of isNumeric function


//--------------------GUI Variables-----------

gchar buf[1024];
gchar param[1024];
gchar sendd[1024];
gchar data[1024];
gchar recvv[1024];
gchar optns[1024];
gchar cmdlist[1024];
gchar Buff[4096];
int comd, test[4], testnum[4];
```

```
int kk=0, ok, tpw, sd;
int ret;
float px,py,phi,q_jnt[3];
char *selectt;
const gchar *entry1, *entry2, *entry3, *entry4;
FILE *data_out;


   GtkBuilder     *main1, *parampop1, *parampop2, *parampop3, *parampop4, *quicktabs, *warning,
*configwarn, *taskplan, *netconn;
   GtkWidget      *mainwindow, *parampop1w, *parampop2w, *parampop3w,
*parampop4w,*mainhalt, *quicktabsw, *warningw, *configwarnw;
   GtkWidget              *taskplanw, *treewidget, *qthaltall, *netconnw;
         GtkEntry                 *parampop3entry1, *parampop3entry2, *parampop3entry3,
*parampop1entry1, *parampop2entry1, *parampop2entry2;
         GtkEntry                 *parampop4entry1, *parampop4entry2, *parampop4entry3,
*parampop4entry4, *netconn_ip, *netconn_port;
         GtkLabel                 *funclabelmain, *parampop4lab, *parampop3lab, *parampop1lab,
*parampop2lab, *mainparamlabel, *mainstatus;
         GtkTextView              *textviewm;
         GtkTextBuffer            *buffer1;
//       GtkTextTagTable                  *texttag;
         GtkTextIter              start, end;
         GtkTextIter              iter;
//       GtkTreeIter newrow;
//       GtkTreePath *path1;

gboolean task_plan=false;
gboolean online=false;
gboolean connectok = false;


//-------------------END GUI Variables-----------------
using namespace std;

 int i;
 int sockfd, n, k;

   char ip_addr[100];
//    sprintf (ip_addr,"localhost");


 //------------------------GTK+ GUI CALLBACK START----------------------


// Following function is developed for receiving network connection data, such as IP address and port
number from users.
```

```
extern "C"
void on_netconn_enter_clicked (GtkObject *object, gpointer user_data)
{

                int portno;
                entry1= gtk_entry_get_text (netconn_ip);
                entry2= gtk_entry_get_text (netconn_port);
                test[0] = atoi(entry2);
                testnum[0]=isNumeric(entry2);

        if ((testnum[0] != 1))
        {
                sprintf(ip_addr,"192.168.1.148");
                portno = 5000;
        }
        else
        {
                bzero(ip_addr,sizeof(ip_addr));
                sprintf(ip_addr,"%s",entry1);
                sprintf(param,"%s",entry2);
                sscanf(param,"%d",&portno);
        }
        gtk_widget_hide(netconnw);

 //-------------------------TCP/IP Connection code is written in this section---------

  struct sockaddr_in serv_addr;
  struct hostent *server;
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0)
     error("ERROR opening socket");

  server = gethostbyname(ip_addr);

  if (server == NULL)
  {
     fprintf(stderr,"ERROR, no such host\n");
     exit(0);
  }
  bzero((char *) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  bcopy((char *)server->h_addr,
      (char *)&serv_addr.sin_addr.s_addr,
      server->h_length);
  serv_addr.sin_port = htons(portno);

  if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
          error("ERROR connecting");
```

```
	signal(SIGINT, signal_callback_handler);

//-------TCP/IP End--------

}


// Start of the combo box function
extern "C"
void on_combobox1main_changed (GtkComboBox *combobox1main, gpointer user_data)
{
selectt = gtk_combo_box_get_active_text(combobox1main);
sprintf(buf,"%s",selectt);
gtk_label_set_text(funclabelmain,buf);

}

//Selection of On-line/Task plan mode

extern "C"
void on_onlinebut_clicked (GtkButton *button, gpointer user_data)
{

	if (gtk_toggle_button_get_active( GTK_TOGGLE_BUTTON( onlinebut )))
	{
	task_plan=false;
	online=true;
	printf ("Online Mode:\n");
	n = write(sockfd,"1000",4);
	gtk_widget_show(quicktabsw);
	}
}

extern "C"
void on_taskplanbut_clicked (GtkButton *button, gpointer user_data)
{
	if (gtk_toggle_button_get_active( GTK_TOGGLE_BUTTON( taskplanbut )))
	{
	task_plan=true;
	online=false;
	printf ("Task plan Mode:\n");
	n = write(sockfd,"1001",4);
	gtk_widget_show(taskplanw);
	}
}
```

```
//End online/task plan mode

extern "C"
void on_mainokbut_clicked (GtkButton *mainokbut, gpointer user_data)
{

ok = 1;
/*
SCHUNK_INITIALIZE       1
SCHUNK_CLEANUP          2
SCHUNK_SET_PARA         3
SCHUNK_MOV_JNT          4
SCHUNK_HALT_ALL         5
SCHUNK_MOV_LOCK         6
SCHUNK_MOV_UNLOCK       7
SCHUNK_MOV_RETRIEVE     8
SCHUNK_MOV_DUMP         9
SCHUNK_SET_JNT_PASSIVE  10
SCHUNK_RESET_ALL        11
SCHUNK_FWD_KIN          12
SCHUNK_INV_KIN          13
ELMO INITIALIZE         14
ELMO_CLEANUP            15
SCHUNK_WAIT_FINISH      16
SCHUNK_GET_MAST_CFG     17
SCHUNK_GET_JNT_POS      18
SCHUNK_JUDGE_LOCK       19
SCHUNK_TOUCH_DETECT     20
ELMO_READ_FORCESENSOR   21
QUIT                    100
*/

gchar compare1[50];
sprintf(compare1,"%s","SCHUNK_INITIALIZE");

gchar compare2[50];
sprintf(compare2,"%s","SCHUNK_CLEANUP");

gchar compare3[50];
sprintf(compare3,"%s","SCHUNK_SET_PARA");

gchar compare4[50];
sprintf(compare4,"%s","SCHUNK_MOV_JNT");

gchar compare5[50];
sprintf(compare5,"%s","SCHUNK_HALT_ALL");

gchar compare6[50];
```

```
sprintf(compare6,"%s","SCHUNK_MOV_LOCK");

gchar compare7[50];
sprintf(compare7,"%s","SCHUNK_MOV_UNLOCK");

gchar compare8[50];
sprintf(compare8,"%s","SCHUNK_MOV_RETRIEVE");

gchar compare9[50];
sprintf(compare9,"%s","SCHUNK_MOV_DUMP");

gchar compare10[50];
sprintf(compare10,"%s","SCHUNK_SET_JNT_PASSIVE");

gchar compare11[50];
sprintf(compare11,"%s","SCHUNK_RESET_ALL");

gchar compare12[50];
sprintf(compare12,"%s","SCHUNK_FWD_KIN");

gchar compare13[50];
sprintf(compare13,"%s","SCHUNK_INV_KIN");

gchar compare14[50];
sprintf(compare14,"%s","ELMO INITIALIZE");

gchar compare15[50];
sprintf(compare15,"%s","ELMO_CLEANUP");

gchar compare16[50];
sprintf(compare16,"%s","SCHUNK_WAIT_FINISH");

gchar compare17[50];
sprintf(compare17,"%s","SCHUNK_GET_MAST_CFG");

gchar compare18[50];
sprintf(compare18,"%s","SCHUNK_GET_JNT_POS");

gchar compare19[50];
sprintf(compare19,"%s","SCHUNK_JUDGE_LOCK");

gchar compare20[50];
sprintf(compare20,"%s","SCHUNK_TOUCH_DETECT");

gchar compare21[50];
sprintf(compare21,"%s","ELMO_READ_FORCESENSOR");

gchar compare22[50];
```

```
sprintf(compare22,"%s","QUIT");

if (strcmp(buf,compare1)==0) //schunk_init
{
comd =1;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare2)==0) //schunk_clean
{
comd =2;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare3)==0) //schunk_set_para
{
comd =3;
sprintf(optns,"%s \nPlease input the desired joint angles (Shoulder, Elbow, Wrist): ",buf);
gtk_widget_show(parampop3w);
gtk_label_set_text(parampop3lab,optns);
}

else if (strcmp(buf,compare4)==0) //schunk_mov_jnt
{
comd =4;
sprintf(optns,"%s \nPlease input the JntID (6-Shoulder, 7-Elbow, 8-Wrist):",buf);
gtk_widget_show(parampop2w);
gtk_label_set_text(parampop2lab,optns);
}

else if (strcmp(buf,compare5)==0) //HALT
{
comd =5;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare6)==0) //schunk_lock
{
comd =6;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare7)==0) //schunk_unlock
{
```

```
comd =7;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare8)==0) //schunk_retrieve
{
comd =8;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare9)==0) //schunk_dump
{
comd =9;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare10)==0) //schunk_set_jnt_passive
{
comd =10;
sprintf(param,"Func: Not Available");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare11)==0) //schunk_reset_all
{
comd =11;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare12)==0) //schunk_fwd_kin
{
comd =12;
sprintf(param,"Func: Not Available");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare13)==0) //schunk_inv_kin
{
comd =13;
sprintf(optns,"%s \nPlease input px, py and phi: ",buf);
gtk_widget_show(parampop3w);
gtk_label_set_text(parampop3lab,optns);

}
```

```
else if (strcmp(buf,compare14)==0) //elmo ini
{
comd =14;
sprintf(param,"Func: Not Available");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare15)==0) //elmo_cleanup
{
comd =15;
sprintf(param,"Func: Not Available");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare16)==0) //schunk_get_param
{
comd =16;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare17)==0) //schunk_get_mast_cfg
{
comd =17;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare18)==0) //schunk_get_jnt_pos
{
comd =18;
sprintf(optns,"%s \nPlease input the JntID (6-Shoulder, 7-Elbow, 8-Wrist):",buf);
gtk_widget_show(parampop1w);
gtk_label_set_text(parampop1lab,optns);
}

else if (strcmp(buf,compare19)==0) //schunk_touch_detect
{
comd =19;
sprintf(param,"Func: Not Available");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare20)==0) //elmo_read_forcesensor
{
comd =20;
sprintf(param,"Func: Not Available");
```

```
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare21)==0) // N/A
{
comd =21;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

else if (strcmp(buf,compare22)==0) //quit
{
comd =100;
sprintf(param," ");
gtk_label_set_text(mainparamlabel,param);
}

if ((comd == 3)||(comd == 4)||(comd == 18))
        {
        gtk_label_set_text(mainstatus, "Status: Configuration...");
        }
else
        {
        gtk_label_set_text(mainstatus, "Status: Ready...");
        }

}

//Parameter popup3 OK

extern "C"
void on_parampop3ok_clicked (GtkButton *button, gpointer user_data)
{

                entry1= gtk_entry_get_text (parampop3entry1);
                test[0] = atoi(entry1);
                testnum[0]=isNumeric(entry1);
                entry2= gtk_entry_get_text (parampop3entry2);
                test[1] = atoi(entry2);
                testnum[1]=isNumeric(entry2);
                entry3= gtk_entry_get_text (parampop3entry3);
                test[2] = atoi(entry3);
                testnum[2]=isNumeric(entry3);

        if ((testnum[0] == 1) && (testnum[1]==1) && (testnum[2]==1))
        {
                kk = user_input(comd,test);
                while (kk == -1)
```

```
                        {
                        gtk_widget_show(warningw);
                        gtk_widget_hide(parampop3w);
                        break;
                        }

                sprintf(param,"%s %s %s",entry1, entry2, entry3);
                gtk_widget_hide(parampop3w);
                gtk_label_set_text(mainparamlabel, param);
        }//end if
        else
        {
                gtk_widget_show(warningw);
                gtk_widget_hide(parampop3w);
        }// end else
        gtk_label_set_text(mainstatus, "Status: Ready...");
        gtk_entry_set_text(parampop3entry1, "");
        gtk_entry_set_text(parampop3entry2, "");
        gtk_entry_set_text(parampop3entry3, "");
}

//parameter popup1

extern "C"
void on_parampop1ok_clicked (GtkButton *button, gpointer user_data)
        {
                entry1= gtk_entry_get_text (parampop1entry1);
                test[0] = atoi(entry1);
                testnum[0]=(isNumeric(entry1));
        if (testnum[0] == 1)
        {
                kk = user_input(comd,test);
                while (kk == -1)
                        {
                        gtk_widget_show(warningw);
                        gtk_widget_hide(parampop1w);
                        break;
                        }

                sprintf(param,"%s",entry1);
                gtk_widget_hide(parampop1w);
                gtk_label_set_text(mainparamlabel, param);
        }//end if
        else
        {
                gtk_widget_show(warningw);
                gtk_widget_hide(parampop1w);
        }//end else
```

```
	gtk_label_set_text(mainstatus, "Status: Ready...");
	gtk_entry_set_text(parampop1entry1, "");

	}

//parampop2 ok
extern "C"
void on_parampop2ok_clicked (GtkButton *button, gpointer user_data)
	{
		entry1= gtk_entry_get_text (parampop2entry1);
		test[0] = atoi(entry1);
		testnum[0]=isNumeric(entry1);
		entry2= gtk_entry_get_text (parampop2entry2);
		test[1] = atoi(entry2);
		testnum[1]=isNumeric(entry2);
	if ((testnum[0] == 1) && (testnum[1]==1))
	{
		kk = user_input(comd,test);
		while (kk == -1)
			{
			gtk_widget_show(warningw);
			gtk_widget_hide(parampop2w);
			break;
			}

		sprintf(param,"%s %s",entry1, entry2);
		gtk_widget_hide(parampop2w);
		gtk_label_set_text(mainparamlabel, param);
	}//end if
	else
	{
		gtk_widget_show(warningw);
		gtk_widget_hide(parampop2w);
	}// end else
	gtk_label_set_text(mainstatus, "Status: Ready...");
	gtk_entry_set_text(parampop2entry1, "");
	gtk_entry_set_text(parampop2entry2, "");
	}

//parampop4 ok
extern "C"
void on_parampop4ok_clicked (GtkButton *button, gpointer user_data)
	{
		entry1= gtk_entry_get_text (parampop4entry1);
		test[0] = atoi(entry1);
		testnum[0]=isNumeric(entry1);
		entry2= gtk_entry_get_text (parampop4entry2);
		test[1] = atoi(entry2);
```

```
		testnum[1]=isNumeric(entry2);
		entry3= gtk_entry_get_text (parampop4entry3);
		test[2] = atoi(entry3);
		testnum[2]=isNumeric(entry3);
		entry4= gtk_entry_get_text (parampop4entry4);
		test[3] = atoi(entry4);
		testnum[3]=isNumeric(entry4);

	if ((testnum[0] == 1) && (testnum[1]==1) && (testnum[2]==1) && (testnum[3]==1))
	{
		kk = user_input(comd,test);

		while (kk == -1)
			{
			gtk_widget_show(warningw);
			gtk_widget_hide(parampop3w);
			break;
			}

		sprintf(param,"%s %s %s %s",entry1, entry2, entry3, entry4);
		gtk_widget_hide(parampop4w);
		gtk_label_set_text(mainparamlabel, param);
	}//end if
	else
	{
		gtk_widget_show(warningw);
		gtk_widget_hide(parampop4w);
	}//end else
	gtk_label_set_text(mainstatus, "Status: Ready...");
	gtk_entry_set_text(parampop4entry1, "");
	gtk_entry_set_text(parampop4entry2, "");
	gtk_entry_set_text(parampop4entry3, "");
	gtk_entry_set_text(parampop4entry4, "");

}

extern "C"
void on_warningok_clicked (GtkButton *button, gpointer user_data)
	{
	gtk_widget_hide(warningw);
	if ((comd==18))
		{
			gtk_widget_show(parampop1w);
		}
	else if ((comd==4))
		{
			gtk_widget_show(parampop2w);
		}
```

```
	else if ((comd==3) || (comd==13))
		{
			gtk_widget_show(parampop3w);
		}
	}


// Widget Hide Button
extern "C"
void on_parampop4canc_clicked (GtkButton *button, gpointer user_data)
{

gtk_widget_hide(parampop4w);

}

extern "C"
void on_parampop3canc_clicked (GtkButton *button, gpointer user_data)
{

gtk_widget_hide(parampop3w);

}

extern "C"
void on_parampop2canc_clicked (GtkButton *button, gpointer user_data)
{

gtk_widget_hide(parampop2w);

}

extern "C"
void on_parampop1canc_clicked (GtkButton *button, gpointer user_data)
{

gtk_widget_hide(parampop1w);

}
// End Widget Hide Button

//----------------------------Quick Tabs Buttons--------------------------
extern "C"
void on_qtbutcqt_clicked (GtkButton *button, gpointer user_data)

{
gtk_widget_hide(quicktabsw);
}
```

```
extern "C"
void on_qtbutmastmode_clicked (GtkButton *button, gpointer user_data)

{
	float pos[3];
	comd=17;
	sprintf(buf,"%s","MAST_GET_CONFIG");
	sprintf(param," ");
	gtk_label_set(mainparamlabel,param);
	gtk_label_set(funclabelmain,buf);
	sprintf(data,"%d",comd);

	n = write(sockfd,data,strlen(data));
	if (n < 0)
		error("ERROR writing to socket");
	bzero(data,sizeof(data));

	k = read(sockfd,Buff,sizeof(Buff));
	sscanf(Buff,"%*d %f %f %f",&pos[0],&pos[1],&pos[2]);
	bzero(Buff,sizeof(Buff));
	sprintf(recvv,"Shoulder: %f\n Elbow: %f\n,Wrist %f\n",pos[0],pos[1],pos[2]);
	gtk_label_set_text(mainstatus, recvv);
	bzero(recvv,sizeof(recvv));

	sprintf(cmdlist,"Command: %s, Parameters: %s",buf,param);
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	if (gtk_text_buffer_get_char_count(buffer1))
	{
		gtk_text_buffer_insert (buffer1, &iter, "\n",1);
	}
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);
}

extern "C"
void on_qthomeall_clicked (GtkButton *button, gpointer user_data)

{
	float homepos[3] = {0.0,-80.0,-40.0}; // Joint 6, 7, 8
	comd =3;
	sprintf(buf,"%s","MAST_HOME_ALL");
	sprintf(param,"%f %f %f",homepos[0],homepos[1],homepos[2]);
	gtk_label_set(mainparamlabel,param);
	gtk_label_set(funclabelmain,buf);
	sprintf(data,"%d %f %f %f",comd,homepos[0],homepos[1],homepos[2]);
	n = write(sockfd,data,strlen(data));
	if (n < 0)
```

```
		error("ERROR writing to socket");
	bzero(data,sizeof(data));
	k = read(sockfd,Buff,sizeof(Buff));
	printf("Buff: %s",Buff);
	bzero(Buff,sizeof(Buff));

	gtk_label_set_text(mainstatus, "Status: Sent...");

	sprintf(cmdlist,"Command: %s, Parameters: %s",buf,param);
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	if (gtk_text_buffer_get_char_count(buffer1))
	{
		gtk_text_buffer_insert (buffer1, &iter, "\n",1);
	}
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);
}


extern "C"
void on_qtresetall_clicked (GtkButton *button, gpointer user_data)

{
	comd =11;
	sprintf(buf,"%s","MAST_RESET_ALL");
	sprintf(param," ");
	gtk_label_set(mainparamlabel,param);
	gtk_label_set(funclabelmain,buf);
	sprintf(sendd,"%s",param);
	sprintf(data,"%d",comd);
	n = write(sockfd,data,strlen(data));
	if (n < 0)
		error("ERROR writing to socket");
	bzero(data,sizeof(data));

	k = read(sockfd,Buff,sizeof(Buff));
	bzero(Buff,sizeof(Buff));

	gtk_label_set_text(mainstatus, "Status: Sent...");

	sprintf(cmdlist,"Command: %s, Parameters: %s",buf,param);
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	if (gtk_text_buffer_get_char_count(buffer1))
	{
		gtk_text_buffer_insert (buffer1, &iter, "\n",1);
	}
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);
```

```
}

extern "C"
void on_qthaltall_clicked (GtkButton *button, gpointer user_data)

{
        comd =5;
        sprintf(buf,"%s","SCHUNK_HALT_ALL");
        sprintf(param," ");
        gtk_label_set(mainparamlabel,param);
        gtk_label_set(funclabelmain,buf);

        sprintf(sendd,"%d %s",comd,param);
    sprintf(data,"%s",sendd);
        n = write(sockfd,data,strlen(data));
        if (n < 0)
                error("ERROR writing to socket");
        bzero(data,sizeof(data));
        bzero(Buff,sizeof(Buff));
        gtk_label_set_text(mainstatus, "Status: Sent...");

        sprintf(cmdlist,"Command: %s, Parameters: %s",buf,param);
        gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
        if (gtk_text_buffer_get_char_count(buffer1))
        {
                gtk_text_buffer_insert (buffer1, &iter, "\n",1);
        }
        gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
        gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);

}

extern "C"
void on_qtscoop_clicked (GtkButton *button, gpointer user_data)

{
        comd =8;
        sprintf(buf,"%s","MAST_SAMPLE_RETRIEVE");
        sprintf(param," ");
        gtk_label_set(mainparamlabel,param);
        gtk_label_set(funclabelmain,buf);
        sprintf(sendd,"%s",param);
        sprintf(data,"%d",comd);
        n = write(sockfd,data,strlen(data));
        bzero(data,sizeof(data));
        if (n < 0)
                error("ERROR writing to socket");
        k = read(sockfd,Buff,sizeof(Buff));
```

```
	bzero(Buff,sizeof(Buff));

	gtk_label_set_text(mainstatus, "Status: Sent...");

	sprintf(cmdlist,"Command: %s, Parameters: %s",buf,param);
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	if (gtk_text_buffer_get_char_count(buffer1))
	{
		gtk_text_buffer_insert (buffer1, &iter, "\n",1);
	}
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);
}

extern "C"
void on_qtdroprb_clicked (GtkButton *button, gpointer user_data)

{
	comd =9;
	sprintf(buf,"%s","MAST_SAMPLE_DUMP - Right Bin");
	sprintf(param," ");
	gtk_label_set(mainparamlabel,param);
	gtk_label_set(funclabelmain,buf);
	sprintf(data,"%d",comd);
	n = write(sockfd,data,strlen(data));
	bzero(data,sizeof(data));
	if (n < 0)
		error("ERROR writing to socket");
	k = read(sockfd,Buff,sizeof(Buff));
	bzero(Buff,sizeof(Buff));

	gtk_label_set_text(mainstatus, "Status: Sent...");

	sprintf(cmdlist,"Command: %s, Parameters: %s",buf,param);
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	if (gtk_text_buffer_get_char_count(buffer1))
	{
		gtk_text_buffer_insert (buffer1, &iter, "\n",1);
	}
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);
}

extern "C"
void on_qtdroplb_clicked (GtkButton *button, gpointer user_data)

{
	comd =9;
```

```
	sprintf(buf,"%s","MAST_SAMPLE_DUMP - Left Bin");
	sprintf(param," ");
	gtk_label_set(mainparamlabel,param);
	gtk_label_set(funclabelmain,buf);
	sprintf(data,"%d",comd);
	n = write(sockfd,data,strlen(data));
	bzero(data,sizeof(data));
	if (n < 0)
		error("ERROR writing to socket");
	k = read(sockfd,Buff,sizeof(Buff));
	bzero(Buff,sizeof(Buff));

	gtk_label_set_text(mainstatus, "Status: Sent...");

	sprintf(cmdlist,"Command: %s, Parameters: %s",buf,param);
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	if (gtk_text_buffer_get_char_count(buffer1))
	{
		gtk_text_buffer_insert (buffer1, &iter, "\n",1);
	}
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);
}



//-----------------------------Quick Tabs Buttons End----------------------

extern "C"
void on_mainhalt_clicked (GtkButton *button, gpointer user_data)

{
	comd =5;
	sprintf(buf,"%s","SCHUNK_HALT_ALL");
	sprintf(param," ");
	gtk_label_set(mainparamlabel,param);
	gtk_label_set(funclabelmain,buf);

	sprintf(sendd,"%d %s",comd,param);
    sprintf(data,"%s",sendd);
	n = write(sockfd,data,strlen(data));
	bzero(data,sizeof(data));
	if (n < 0)
		error("ERROR writing to socket");

	bzero(sendd,sizeof(sendd));
	bzero(data,sizeof(data));
	bzero(Buff,sizeof(Buff));
```

```
	gtk_label_set_text(mainstatus, "Status: Sent...");

	sprintf(cmdlist,"Command: %s, Parameters: %s",buf,param);
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	if (gtk_text_buffer_get_char_count(buffer1))
	{
		gtk_text_buffer_insert (buffer1, &iter, "\n",1);
	}
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);
}

extern"C"
void on_configwarnok_clicked (GtkButton *button, gpointer user_data)
{
	gtk_widget_hide(configwarnw);
}

//----------------------Send Command-------------------------
extern "C"
void on_mainsendbut_clicked (GtkButton *button, gpointer user_data)
{
if (ok==0)
	{
	gtk_widget_show (configwarnw);
	}
while (param[0] != '\0')
{
sprintf(sendd,"%d %s",comd,param);
//---------------------Print Command History------------------
if (comd == 4)
	{
	int mode; float pos;
	sscanf(param, "%d %f", &mode,&pos);
	if (mode == 6)
		{
		sprintf(cmdlist,"Command: %s, Parameters: Joint->Shoulder: %d, Position:
%.2f",buf,mode,pos);
		}
	if (mode == 7)
		{
		sprintf(cmdlist,"Command: %s, Parameters: Joint->Elbow: %d, Position:
%.2f",buf,mode,pos);
		}
	if (mode == 8)
		{
		sprintf(cmdlist,"Command: %s, Parameters: Joint->Wrist: %d, Position:
%.2f",buf,mode,pos);
```

```
                }
        }


else if (comd == 3)
        {
        float pos[3];
        sscanf(param, "%f%f%f", &pos[0],&pos[1],&pos[2]);
        sprintf(cmdlist,"Command: %s, Parameters: Shoulder: %.2f Elbow: %.2f Wrist:
%.2f",buf,pos[0],pos[1],pos[2]);
        }

else if (comd == 18)
        {
        int mode;
        sscanf(param, "%d", &mode);
        if (mode == 6)
                {
                sprintf(cmdlist,"Command: %s, Parameters: Joint->Shoulder: %d",buf,mode);
                }
        if (mode == 7)
                {
                sprintf(cmdlist,"Command: %s, Parameters: Joint->Elbow: %d",buf,mode);
                }
        if (mode == 8)
                {
                sprintf(cmdlist,"Command: %s, Parameters: Joint->Wrist: %d",buf,mode);
                }
        }
else
        {
        sprintf(cmdlist,"Command: %s, Parameters: %s",buf,param);
        }
//------------------End Print command History------------------

while (online)
{

while ((kk != -1) && (ok == 1))
{
        ok=0;
        if((comd == 100) || ((comd >= 1) && (comd<=15)))
        {
                if (comd == 13)
                {
                        sscanf(param,"%f %f %f",&px,&py,&phi);

                        ret = inv_kin(px, py, phi, q_jnt);
```

```
		if (ret == 1)
		{
			comd = 3;
			sprintf(data,"%d %f %f %f",comd,q_jnt[0],q_jnt[1],q_jnt[2]);
			n = write(sockfd,data,strlen(data));

			k = read(sockfd,Buff,sizeof(Buff));
			printf ("Buff is: %s\n",Buff);
			if (n < 0)
			{
				error("ERROR writing to socket");
			}
			bzero(data,sizeof(data));
			bzero(sendd,sizeof(sendd));
			bzero(Buff,sizeof(Buff));
		}
		else
		{
			gtk_widget_show(warningw);
		}

	}//INV KIN

	else
	{
sprintf(data,"%s",sendd);
	n = write(sockfd,data,strlen(data));
	k = read(sockfd,Buff,sizeof(Buff));
	printf ("Buff is: %s\n",Buff);
	if (n < 0)
		{
		error("ERROR writing to socket");
		}
	bzero(data,sizeof(data));
	bzero(sendd,sizeof(sendd));
	bzero(Buff,sizeof(Buff));

	gtk_label_set_text(mainstatus, "Status: Sent...");

	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	if (gtk_text_buffer_get_char_count(buffer1))
	{
		gtk_text_buffer_insert (buffer1, &iter, "\n",1);
	}
	gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
	gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);
	break;
```

```
            }
        }//end if

        else if((comd<=21) && (comd>=16))
        {
            sprintf(data,"%s",sendd);
                n = write(sockfd,data,strlen(data));
                if (n < 0)
                        error("ERROR writing to socket");
                bzero(Buff,sizeof(Buff));
                k = read(sockfd,Buff,sizeof(Buff));
                if (k < 0)
                        error("Error reading from socket");

                if (comd==18)
                {

                        int jnt_id=test[0];
                        float jnt_pos;
                        sscanf(Buff, "%*d %f", &jnt_pos);
                        if (jnt_id==6)
                        {
                                sprintf(recvv,"joint[%d] - Shoulder .pos=%f\n", jnt_id, jnt_pos);
                        }
                        else if (jnt_id==7)
                        {
                                sprintf(recvv,"joint[%d] - Elbow .pos=%f\n", jnt_id, jnt_pos);
                        }
                        else if (jnt_id==8)
                        {
                                sprintf(recvv,"joint[%d] - Wrist .pos=%f\n", jnt_id, jnt_pos);
                        }
                        gtk_label_set_text(mainstatus, recvv);
                }
                else if (comd==17)
                {
                        float jnt_pos[3];
                        sscanf(Buff, "%*d %f %f %f\n", &jnt_pos[0], &jnt_pos[1], &jnt_pos[2]);
                        sprintf(recvv,"Mast Configuration is:\n - Shoulder: %f \n - Elbow: %f \n - Wrist:
%f \n", jnt_pos[0], jnt_pos[1],jnt_pos[2]);
                        gtk_label_set_text(mainstatus, recvv);
                }
        }//else end


                gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
                if (gtk_text_buffer_get_char_count(buffer1))
                {
```

```
                        gtk_text_buffer_insert (buffer1, &iter, "\n",1);
                }
                gtk_text_buffer_get_iter_at_offset(buffer1, &iter, 0);
                gtk_text_buffer_insert (buffer1, &iter, cmdlist,-1);
        break;
        }//while Buff[0] end
break;
}//online while end
while (task_plan && ok == 1)
{
        ok = 0;
        model = gtk_tree_view_get_model(GTK_TREE_VIEW(treeview1));
        gtk_list_store_append(GTK_LIST_STORE(model), &newrow);
        gtk_list_store_set(GTK_LIST_STORE(model), &newrow, COL_TASK, buf,
COL_STATUS,"Pending",COL_CMD,sendd,-1);
        break;

}//task plan while end

break;
}//while empty array check end
}
//------------------End Send Command------------------------------

// Current Status -----------
extern "C"
void on_mainstatusbut_clicked (GtkButton *button, gpointer user_data)
{
        int tmpcmd = 17;
        sprintf(data,"%d",tmpcmd);
        n = write(sockfd,data,strlen(data));
        if (n < 0)
                error("ERROR writing to socket");
        bzero(Buff,sizeof(Buff));
        k = read(sockfd,Buff,sizeof(Buff));
        if (k < 0)
                error("Error reading from socket");


        float jnt_pos[3];
        sscanf(Buff, "%*d %f %f %f\n", &jnt_pos[0], &jnt_pos[1], &jnt_pos[2]);
        sprintf(recvv,"Current Pos: \nShoulder: %f \nElbow: %f \nWrist: %f", jnt_pos[0],
jnt_pos[1],jnt_pos[2]);
        gtk_label_set_text(mainstatus, recvv);
        bzero(Buff,sizeof(Buff));

}
```

```
//End Current status

//Execute Command start

extern "C"
void on_task_execute_but_clicked (GtkButton *button, gpointer user_data)
{

        getcwd(cwd,sizeof(cwd));
        time_t now; //time_t should be declared in time.h as long
        now = time(NULL); //current time in C representation
        current = localtime(&now); //IMPORTANT you have to use a pointer to time_t
        sprintf(dir,"%s/data/data_%d%d%d.txt",cwd,current->tm_hour,current->tm_min,current-
>tm_sec);

    data_out = fopen(dir,"a");

        model = gtk_tree_view_get_model(GTK_TREE_VIEW(treeview1));

        gint num = gtk_tree_model_iter_n_children(model, NULL);

        gtk_tree_model_get_iter_first (model, &newrow);
        bzero(data,sizeof(data));

        int i,errno, conf;
        conf = -1;
        for (i = 1; i<=num;i++)
        {
        int jnt;
        entry_ins = '\0';
        errno = 0;
        gtk_tree_model_get(model, &newrow, COL_CMD, &entry_ins,-1);

        printf ("%d String is: %s\n",i,entry_ins);
        sscanf (entry_ins,"%d",&comd);

        if (comd == 1)
        {
                errno = 0;
                while (errno < 5)
                {
                bzero(Buff,sizeof(Buff));
                n = write(sockfd,entry_ins,strlen(entry_ins));
                k = read(sockfd,Buff,sizeof(Buff));
                sscanf(Buff,"%d",&conf);
                        if (conf == 1)
                        {
```

```
                                        break;
                                        fprintf(data_out,"\nDONE\n");
                                }
                                else
                                {
                                        errno++;
                                        printf ("Err: %d\n",errno);
                                        sleep(5);
                                }
                        }//while err end
                }

                else if (comd == 3)
                {
                        while (errno < 5)
                                {
                                int cnt;
                                int count = 1;
                                float pos[3],pos_now[3];
                                pos_now[0] = -300;pos_now[1] = -300;pos_now[2] = -300;
                                bzero(data,sizeof(data));
                                n = write(sockfd,entry_ins,strlen(entry_ins));
                                k = read(sockfd,Buff,4095);
                                printf ("Buffer %s \n",Buff);
                                sscanf(Buff,"%d %f %f %f
%d",&conf,&pos_now[0],&pos_now[1],&pos_now[2],&cnt);
                                sscanf(entry_ins,"%*d %f %f %f",&pos[0],&pos[1],&pos[2]);
                                printf("Pos_temp: %f %f %f and conf %d and cnt:
%d\n",pos_now[0],pos_now[1],pos_now[2],conf,cnt);
                                fprintf (data_out,"%f %f %f\n",pos_now[0],pos_now[1],pos_now[2]);
                                bzero(Buff,sizeof(Buff));

                                while(conf == 1) //TRY conf == 1
                                        {
                                        if ((fabs(pos[0]-pos_now[0]) <=0.1) && (fabs(pos[1]-pos_now[1]) <=0.1)
&& (fabs(pos[2]-pos_now[2]) <=0.1))
                                                {
                                                k = read(sockfd,Buff,sizeof(Buff));
                                                count++;
                                                printf ("count: %d\n",count);
                                                bzero(Buff,sizeof(Buff));
                                                usleep(100);
                                                break;
                                                }

                                        bzero(Buff,sizeof(Buff));
                                        k = read(sockfd,Buff,sizeof(Buff));
                                        count++;
```

```
                                sscanf(Buff,"%d %f %f %f
%d",&conf,&pos_now[0],&pos_now[1],&pos_now[2],&cnt);
                                printf("Pos_temp: %f %f %f and conf: %d and cnt:
%d\n",pos_now[0],pos_now[1],pos_now[2],conf,cnt);
                                fprintf (data_out,"%f %f %f\n",pos_now[0],pos_now[1],pos_now[2]);
//                              usleep(50);
                                }

                        if (conf == 1)
                                {
                                if (cnt != count)
                                {
                                sleep(2);
                                k = read(sockfd,Buff,sizeof(Buff));
                                //printf ("!count > %s\n",Buff);
                                }
                                printf ("DONE\n");
                                fprintf(data_out,"\nDONE\n");
                                bzero(Buff,sizeof(Buff));
                                gtk_list_store_set(GTK_LIST_STORE(model),
&newrow,COL_STATUS,"Complete",-1);
                                sleep(5);
                                break;
                                }
                        else
                                {
                                errno++;
                                printf ("K is: %d Err: %d\n",k,errno);
                                sleep(5);
                                }

                        }//while err end
        }//if comd end



        else if (comd == 4)
        {
                while (errno < 5)
                        {
                        float pos,pos_now;
                        pos_now = -300;
                        bzero(Buff,sizeof(Buff));
                        bzero(data,sizeof(data));
                        n = write(sockfd,entry_ins,strlen(entry_ins));
                        k = read(sockfd,Buff,sizeof(Buff));
                        sscanf(Buff,"%d",&conf);
                        sscanf(entry_ins,"%*d %d %f",&jnt,&pos);
```

```
                        printf("Pos_temp: %f, conf %d\n",pos_now,conf);
                        sprintf(data,"%d %d",18,jnt);
                        while(conf == 1) //TRY conf == 1
                                {
                                bzero(Buff,sizeof(Buff));
                                n = write(sockfd,data,strlen(data));
                                k = read(sockfd,Buff,sizeof(Buff));
                                sscanf(Buff,"%d %f",&conf,&pos_now);
                                printf("Pos_temp: %f and conf: %d\n",pos_now,conf);
                                fprintf(data_out,"%f\n",pos_now);
                                if (fabs(pos-pos_now) <=0.1)
                                        {
                                        bzero(Buff,sizeof(Buff));
                                        break;
                                        }
                                //usleep(5000);
                                }
                        if (conf == 1)
                                {
                                fprintf(data_out,"\nDONE\n");
                                gtk_list_store_set(GTK_LIST_STORE(model),
&newrow,COL_STATUS,"Complete",-1);
                                sleep(5);
                                break;
                                }
                        else
                                {
                                errno++;
                                printf ("Err: %d\n",errno);
                                sleep(5);
                                }

                        }//while err end
        }//if comd end

        else if (comd == 8)
        {
        float pos_des[3] = {80.0f,80.0f,50.0f};                 // Moving back to the ready position

                while (errno < 5)
                        {
                        int cnt;
                        int count = 1;
                        float pos_now[3];
                        bzero(Buff,sizeof(Buff));
                        bzero(data,sizeof(data));
                        n = write(sockfd,entry_ins,strlen(entry_ins));
                        k = read(sockfd,Buff,sizeof(Buff));
```

```
                        printf ("Buffer %s \n",Buff);
                        sscanf(Buff,"%d %f %f %f
%d",&conf,&pos_now[0],&pos_now[1],&pos_now[2],&cnt);
                        printf("Pos_temp: %f %f %f and conf %d and cnt:
%d\n",pos_now[0],pos_now[1],pos_now[2],conf,cnt);
                        fprintf (data_out,"%f %f %f\n",pos_now[0],pos_now[1],pos_now[2]);
                        while(conf == 1) //TRY conf == 1
                                {
                                if ((fabs(pos_des[0]-pos_now[0]) <=0.1) && (fabs(pos_des[1]-
pos_now[1]) <=0.1) && (fabs(pos_des[2]-pos_now[2]) <=0.1))
                                        {
                                        k = read(sockfd,Buff,sizeof(Buff));
                                        count++;
                                        printf ("count: %d\n",count);
                                        bzero(Buff,sizeof(Buff));
                                        usleep(100);
                                        break;
                                        }
                                bzero(Buff,sizeof(Buff));
                                k = read(sockfd,Buff,sizeof(Buff));
                                count++;
                                sscanf(Buff,"%d %f %f
%f\n",&conf,&pos_now[0],&pos_now[1],&pos_now[2]);
                                printf("Pos_temp: %f %f %f and conf:
%d\n",pos_now[0],pos_now[1],pos_now[2],conf);
                                fprintf (data_out,"%f %f %f\n",pos_now[0],pos_now[1],pos_now[2]);
//                              usleep(50);
                                }
                        if (conf == 1)
                                {
                                if (cnt != count)
                                {
                                sleep(2);
                                k = read(sockfd,Buff,sizeof(Buff));
                                //printf ("!count > %s",Buff);
                                }
                                printf ("DONE\n");
                                fprintf(data_out,"\nDONE\n");
                                bzero(Buff,sizeof(Buff));
                                gtk_list_store_set(GTK_LIST_STORE(model),
&newrow,COL_STATUS,"Complete",-1);
                                sleep(5);
                                break;
                                }
                        else
                                {
                                errno++;
                                printf ("Err: %d\n",errno);
```

```
                                sleep(5);
                                }

                        }//while err end
        }//if comd end


        else if (comd == 9)
        {
        int cnt;
        int count = 1;
        float pos_des[3] = {80.0f,80.0f,50.0f};                 // Moving back to the ready position
        float pos_dump[3] = {40.0f,140.0f,-30.0f};   // Dumping position

                while (errno < 5)
                        {
                        int dump = -1;
                        float pos_now[3];
                        bzero(Buff,sizeof(Buff));
                        bzero(data,sizeof(data));
                        n = write(sockfd,entry_ins,strlen(entry_ins));
                        k = read(sockfd,Buff,sizeof(Buff));
                        printf ("Buffer %s \n",Buff);
                        sscanf(Buff,"%d %f %f %f
%d",&conf,&pos_now[0],&pos_now[1],&pos_now[2],&cnt);
                        printf("Pos_temp: %f %f %f and conf %d and cnt:
%d\n",pos_now[0],pos_now[1],pos_now[2],conf,cnt);
                        fprintf (data_out,"%f %f %f\n",pos_now[0],pos_now[1],pos_now[2]);

                        while(conf == 1) //TRY conf == 1
                                {
                                if ((fabs(pos_dump[0]-pos_now[0]) <=0.1) && (fabs(pos_dump[1]-
pos_now[1]) <=0.1) && (fabs(pos_dump[2]-pos_now[2]) <=0.1))
                                {
                                dump = 1;
                                printf ("Dumped\n");
                                }
                        if ((dump == 1) && (fabs(pos_des[0]-pos_now[0]) <=0.1) && (fabs(pos_des[1]-
pos_now[1]) <=0.1) && (fabs(pos_des[2]-pos_now[2]) <=0.1))
                                        {
                                        printf ("count: %d\n",count);
                                        bzero(Buff,sizeof(Buff));
                                        usleep(100);
                                        break;
                                        }
                                bzero(Buff,sizeof(Buff));
                                k = read(sockfd,Buff,sizeof(Buff));
                                count++;
```

```
                                sscanf(Buff,"%d %f %f %f
%d\n",&conf,&pos_now[0],&pos_now[1],&pos_now[2],&cnt);
                                printf("%d Pos_temp: %f %f %f and conf: %d and cnt:
%d\n",count,pos_now[0],pos_now[1],pos_now[2],conf,cnt);
                                fprintf (data_out,"%f %f %f\n",pos_now[0],pos_now[1],pos_now[2]);
//                              usleep(50);
                                }
                        if (conf == 1)
                                {
                                if (cnt != count)
                                {
                                sleep(2);
                                k = read(sockfd,Buff,sizeof(Buff));
                                printf ("!count > %s\n",Buff);
                                }
                                printf ("DONE\n");
                                fprintf(data_out,"\nDONE\n");
                                bzero(Buff,sizeof(Buff));
                                gtk_list_store_set(GTK_LIST_STORE(model),
&newrow,COL_STATUS,"Complete",-1);
                                sleep(5);
                                break;
                                }
                        else
                                {
                                errno++;
                                printf ("Err: %d\n",errno);
                                sleep(5);
                                }

                        }//while err end
        }//if comd end

        else
        {
        errno = 0;
        while (errno < 5)
        {
                bzero(Buff,sizeof(Buff));
                n = write(sockfd,entry_ins,strlen(entry_ins));
                k = read(sockfd,Buff,sizeof(Buff));
                sscanf(Buff,"%d",&conf);
                        if (conf == 1)
                        {
                                printf ("%s\n",Buff);
                                bzero(Buff,sizeof(Buff));
                                break;
                        }
```

```
                        else
                        {
                                errno++;
                                printf ("Err: %d\n",errno);
                                sleep(5);
                        }
        }//while err end
        }


        gtk_tree_model_iter_next(model, &newrow);
        }//for end
        fclose(data_out);
}

// Execute command end


//------------------GTK+ GUI CALLBACK END----------------------------
int main (int argc, char *argv[])
{


 //----------------GUI CODE----------------------------------
    gtk_init (&argc, &argv);

    main1 = gtk_builder_new ();
    parampop1 = gtk_builder_new ();
    parampop2 = gtk_builder_new ();
    parampop3 = gtk_builder_new ();
    parampop4 = gtk_builder_new ();
    quicktabs = gtk_builder_new();
    warning = gtk_builder_new();
    configwarn = gtk_builder_new();
    taskplan = gtk_builder_new();
    netconn = gtk_builder_new();

    gtk_builder_add_from_file (main1, "main.glade", NULL);
    gtk_builder_add_from_file (parampop1, "parampop1.glade", NULL);
    gtk_builder_add_from_file (parampop2, "parampop2.glade", NULL);
    gtk_builder_add_from_file (parampop3, "parampop3.glade", NULL);
    gtk_builder_add_from_file (parampop4, "parampop4.glade", NULL);
    gtk_builder_add_from_file (quicktabs, "quicktabs.glade", NULL);
    gtk_builder_add_from_file (warning, "warning.glade", NULL);
    gtk_builder_add_from_file (configwarn, "configwarn.glade", NULL);
    gtk_builder_add_from_file (taskplan, "taskplan.glade", NULL);
    gtk_builder_add_from_file (netconn, "netconn.glade", NULL);
```

```
mainwindow = GTK_WIDGET (gtk_builder_get_object (main1, "mainwindow"));
parampop1w = GTK_WIDGET (gtk_builder_get_object (parampop1, "parampop1"));
parampop2w = GTK_WIDGET (gtk_builder_get_object (parampop2, "parampop2"));
parampop3w = GTK_WIDGET (gtk_builder_get_object (parampop3, "parampop3"));
parampop4w = GTK_WIDGET (gtk_builder_get_object (parampop4, "parampop4"));
quicktabsw = GTK_WIDGET (gtk_builder_get_object (quicktabs, "quicktabs"));
warningw = GTK_WIDGET (gtk_builder_get_object (warning, "warning"));
configwarnw = GTK_WIDGET (gtk_builder_get_object (configwarn, "configwarn"));
    taskplanw = GTK_WIDGET (gtk_builder_get_object (taskplan, "taskplan"));
    netconnw = GTK_WIDGET (gtk_builder_get_object (netconn, "netconn"));
    treeview1 = GTK_WIDGET( gtk_builder_get_object( taskplan, "treeview1" ));

    funclabelmain = GTK_LABEL (gtk_builder_get_object (main1, "funclabelmain"));
gtk_builder_connect_signals (main1,funclabelmain);

    parampop4lab = GTK_LABEL (gtk_builder_get_object (parampop4, "parampop4lab"));
gtk_builder_connect_signals (parampop4,parampop4lab);

    parampop3lab = GTK_LABEL (gtk_builder_get_object (parampop3, "parampop3lab"));
gtk_builder_connect_signals (parampop3,parampop3lab);

    parampop2lab = GTK_LABEL (gtk_builder_get_object (parampop2, "parampop2lab"));
gtk_builder_connect_signals (parampop2,parampop2lab);

    parampop1lab = GTK_LABEL (gtk_builder_get_object (parampop1, "parampop1lab"));
gtk_builder_connect_signals (parampop1,parampop1lab);

parampop1entry1 = GTK_ENTRY (gtk_builder_get_object (parampop1, "parampop1entry1"));


parampop2entry1 = GTK_ENTRY (gtk_builder_get_object (parampop2, "parampop2entry1"));
parampop2entry2 = GTK_ENTRY (gtk_builder_get_object (parampop2, "parampop2entry2"));

    parampop3entry1 = GTK_ENTRY (gtk_builder_get_object (parampop3, "parampop3entry1"));
    parampop3entry2 = GTK_ENTRY (gtk_builder_get_object (parampop3, "parampop3entry2"));
    parampop3entry3 = GTK_ENTRY (gtk_builder_get_object (parampop3, "parampop3entry3"));

    parampop4entry1 = GTK_ENTRY (gtk_builder_get_object (parampop4, "parampop4entry1"));
    parampop4entry2 = GTK_ENTRY (gtk_builder_get_object (parampop4, "parampop4entry2"));
    parampop4entry3 = GTK_ENTRY (gtk_builder_get_object (parampop4, "parampop4entry3"));
    parampop4entry4 = GTK_ENTRY (gtk_builder_get_object (parampop4, "parampop4entry4"));

    netconn_ip = GTK_ENTRY (gtk_builder_get_object (netconn, "netconn_ip"));
    netconn_port = GTK_ENTRY (gtk_builder_get_object (netconn, "netconn_port"));

    mainparamlabel = GTK_LABEL (gtk_builder_get_object (main1, "mainparamlabel"));
```

```
        taskplanentry = GTK_ENTRY (gtk_builder_get_object (taskplan, "taskplanentry"));

    gtk_builder_connect_signals (main1,mainparamlabel);
    gtk_builder_connect_signals (quicktabs,NULL);
    gtk_builder_connect_signals (warning,NULL);
    gtk_builder_connect_signals (configwarn,NULL);
    gtk_builder_connect_signals(taskplan, NULL);
    gtk_builder_connect_signals(netconn, NULL);

    g_signal_connect(taskplanentry, "activate", G_CALLBACK(on_insert_but_clicked), treeview1);
    g_signal_connect(netconn_port, "activate", G_CALLBACK(on_netconn_enter_clicked), netconn);

        mainstatus = GTK_LABEL (gtk_builder_get_object (main1, "mainstatus"));
    gtk_builder_connect_signals (main1,mainstatus);

//-----------------------Text View Code---------------------------------------

    textviewm = GTK_TEXT_VIEW (gtk_builder_get_object (main1, "maintextview"));
    buffer1 = gtk_text_view_get_buffer(GTK_TEXT_VIEW (textviewm));


//-----------------------Text view code end-----------------------------------



//Creating colors
        GdkColor my_red;
        my_red.red = 0xffff;
        my_red.green = 0x0000;
        my_red.blue = 0x0000;


//Colors end

    mainhalt = GTK_WIDGET (gtk_builder_get_object (main1, "mainhalt"));
    qthaltall = GTK_WIDGET (gtk_builder_get_object (quicktabs, "qthaltall"));
        onlinebut = GTK_WIDGET (gtk_builder_get_object (main1, "onlinebut"));
        taskplanbut = GTK_WIDGET (gtk_builder_get_object (main1, "taskplanbut"));

//Color changing !!

        gtk_widget_modify_bg(mainhalt, GTK_STATE_NORMAL, &my_red);
        gtk_widget_modify_bg(mainhalt, GTK_STATE_PRELIGHT, &my_red);

        gtk_widget_modify_bg(qthaltall, GTK_STATE_NORMAL, &my_red);
        gtk_widget_modify_bg(qthaltall, GTK_STATE_PRELIGHT, &my_red);
```

```
//End of changing color


      gtk_widget_show_all (mainwindow);
      gtk_widget_show_all (netconnw);

           g_object_unref (G_OBJECT (main1));
      g_object_unref (G_OBJECT (parampop1));
      g_object_unref (G_OBJECT (parampop2));
      g_object_unref (G_OBJECT (parampop3));
      g_object_unref (G_OBJECT (parampop4));
      g_object_unref (G_OBJECT (quicktabs));
      g_object_unref (G_OBJECT (warning));
      g_object_unref (G_OBJECT (configwarn));
      g_object_unref (G_OBJECT (taskplan));
      g_object_unref (G_OBJECT (netconnw));
 //---------------GUI CODE END------------------------------



 gtk_main ();
 js_tele();
 system("clear");

//}// END While
 close(sockfd);
 return 0;
}
```

A.IV inputcheck.h

Code presented below is the header file for validating the input from GUI.

```
#include<stdio.h>

#ifndef INPUT_CHECK
#define INPUT_CHECK
/* function prototypes here */
int user_input(int comd, int test[])
{


int i;

if (comd == 4)
{
```

```
	if((test[0] < 6) || (test[0]>8))
			{
				i=-1;
			}
		if ((test[0]==6) && ((test[1] <-90) || (test[1]>90)))
			{
				i=-1;
			}
		if ((test[0]==7) && ((test[1] <-150) || (test[1]>110)))
			{
				i=-1;
			}
		if ((test[0]==8) && ((test[1] <-90) || (test[1]>5)))
			{
				i=-1;
			}

}

else if (comd == 3)
{

		if ((test[0] <-90) || (test[1]>90))
			{
				i=-1;
			}
		if ((test[1] <-150) || (test[2]>110))
			{
				i=-1;
			}
		if ((test[2] <-90) || (test[3]>5))
			{
				i=-1;
			}
}


else if (comd == 18)
{

	if((test[0] < 6) || (test[0]>9))
			{
				i=-1;
			}
}
return i;
}
```

```
#endif
```

A.V task_signals.h

This part of the code contains necessary function to run the interface for Task Scheduling functionality of the interface.

```
/*
 * Compile me with:
 *   g++ -o main main.c $(pkg-config --cflags --libs gtk+-2.0 gmodule-2.0)
 */

enum
{
  COL_TASK = 0,
  COL_STATUS,
  COL_CMD,
};



#include <gtk/gtk.h>

   GtkWidget  *treeview1, *taskplanbut, *onlinebut;
   GtkTreeModel *model;
   GtkEntry *taskplanentry;
   GtkTreeIter newrow;
   GtkTreePath *path1;



const gchar *entry_ins;

extern "C"
void on_insert_but_clicked (GtkButton *button, gpointer user_data)
{

        entry_ins= gtk_entry_get_text (taskplanentry);

if (entry_ins && *entry_ins)
        {

        model = gtk_tree_view_get_model(GTK_TREE_VIEW(treeview1));

        gtk_list_store_append(GTK_LIST_STORE(model), &newrow);
```

```
	gtk_list_store_set(GTK_LIST_STORE(model), &newrow, COL_TASK, "Manual",
COL_STATUS,"Pending",COL_CMD,entry_ins,-1);

	printf ("Entry: %s \n",entry_ins);

	gtk_entry_set_text(GTK_ENTRY(taskplanentry), ""); /* clear entry */

	}

}


extern "C"
void on_read_but_clicked (GtkButton *button, gpointer user_data)
{

	gint num = gtk_tree_model_iter_n_children(model, NULL);

	if (num != 0)
	{
	printf ("Num of iter: %d\n",num);
	}

}

extern "C"
void on_task_del_but_clicked (GtkButton *button, gpointer user_data)
{


	GtkTreeSelection *sel;
	GtkTreeIter      selected_row;

	sel = gtk_tree_view_get_selection(GTK_TREE_VIEW(treeview1));

//	g_assert(gtk_tree_selection_get_mode(sel) == GTK_SELECTION_SINGLE);

	if (gtk_tree_selection_get_selected(sel, &model, &selected_row))
	{
		gtk_list_store_remove(GTK_LIST_STORE(model), &selected_row);
	}
	else
	{
		/* If no row is selected, the button should
		 *  not be clickable in the first place */
		printf ("No row selected\n");
	}
```

```
}
```

A.VI inv_kin.h

File inv_kin.h is the header file containing needed the code for inverse kinematics calculation of the three link robotic manipulator.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>

#ifndef INV_KIN
#define INV_KIN

# define PI 3.14159265

int inv_kin (float px, float py, float phi, float q_jnt[3])
{
float l1,l2,w,phir,q[3];
float x2,y2,cq2,sq2,qr2,ck,sk,k,qr1,qr3;              //Intermediate variables



l1 = 0.460;
l2 = 0.440;
w = 0.11;

phir = (phi*PI)/180;

x2 = px - (w*cos(phir));
y2 = py - (w*sin(phir));

//elbow down
cq2 = (x2*x2 + y2*y2 - l1*l1 - l2*l2)/(2*l1*l2);
sq2 = sqrt(1-(cq2*cq2));

qr2 = atan2(sq2,cq2);
q[1] = (qr2*180)/PI;

ck = (x2*x2 + y2*y2 + l1*l1 - l2*l2)/(2*sqrt(x2*x2 + y2*y2)*l1);
sk = sqrt(1-(ck*ck));
k = atan2(sk,ck);
qr1 = atan2(y2,x2) - k;
```

```
q[0] = (qr1*180)/PI;

qr3 = (phir-qr1-qr2);
q[2] = (qr3*180)/PI;
//elbow up
float squp2,qrup2,qup[3],cupk,supk,kup,qrup1,qrup3;
squp2 = -1*sqrt(1-(cq2*cq2));

qrup2 = atan2(squp2,cq2);
qup[1] = (qrup2*180)/PI;

cupk = (x2*x2 + y2*y2 + l1*l1 - l2*l2)/(2*sqrt(x2*x2 + y2*y2)*l1);
supk = -1*sqrt(1-(cupk*cupk));
kup = atan2(supk,cupk);
qrup1 = atan2(y2,x2) - kup;
qup[0] = (qrup1*180)/PI;

qrup3 = phir-qrup1-qrup2;
qup[2] = (qrup3*180)/PI;

printf ("Outside if --> q1: %f, q2: %f, q3: %f\n",q[0],q[1],q[2]);
printf ("Outside if --> qup1: %f, qup2: %f, qup3: %f\n",qup[0],qup[1],qup[2]);

if (((qup[0] > -90.0) && (qup[0] < 94.0)) && ((qup[1] > -150.0) && (qup[1] < 140.0)) && ((qup[2] > -90.0)
&& (qup[2] < 5.0)))
{
        //printf ("qup1: %f, qup2: %f, qup3: %f\n",qup[0],qup[1],qup[2]);
        q_jnt[0] = qup[0];
        q_jnt[1] = qup[1];
        q_jnt[2] = qup[2];
        return 1;
}
else if (((q[0] > -90.0) && (q[0] < 40.0)) && ((q[1] > -150.0) && (q[1] < 140.0)) && ((q[2] > -90.0) && (q[2]
< 5.0)))
{
        //printf ("q1: %f, q2: %f, q3: %f\n",q[0],q[1],q[2]);
        q_jnt[0] = q[0];
        q_jnt[1] = q[1];
        q_jnt[2] = q[2];
        return 1;
}
else
{
        printf ("Not Possible\n");
        return 0;
}

}
```

```
#endif
```

A.VII testjs.h

Following code collects joystick input data and sends it to main.c file. Upon pressing trigger button, manual mode is activated and joystick is used to control the manipulator.

```
#include <pthread.h>
#include <stdio.h>
#include <iostream>
#include <string.h>
#include <time.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/joystick.h>
#include <math.h>

#ifndef TESTJS
#define TESTJS
#define NUM_THREADS3
#define JOY_DEV                 "/dev/input/js1"
#define HZ                              12
int sockfd;
char axisDesc[15][5] = {"X", "Y", "Z", "R", "A", "B", "C", "D", "E", "F", "G", "H", "I" };
char numAxis[20], numButtons[20], value[20][20], timeSec[20], timeMilli[20],buffer[1024];
int num_of_axis,n,k;
int jsdata[50];
        int cnt, joy_fd, *axis=NULL, num_of_buttons=0;
        char *button=NULL, name_of_joystick[80];
        struct js_event js;
        time_t theTime;
void *event_read(void *nothing)
{
while(1)
{
usleep(10000);
//      READ THE JOYSTICK STATE, IT WILL BE RETURNED IN THE JS_EVENT STRUCT
                read(joy_fd, &js, sizeof(struct js_event));
        //      CHECK THE EVENT
                switch (js.type & ~JS_EVENT_INIT){
                        case JS_EVENT_AXIS:
```

```
                                axis   [ js.number ] = js.value;
                                break;
                        case JS_EVENT_BUTTON:
                                button [ js.number ] = js.value;
                                break;
                }
                //      PRINT THE RESULTS
                for(cnt = 0; cnt < num_of_axis; cnt++)
                {
                        sprintf(value[cnt], "%d", axis[cnt]);
                }
                for(cnt = 0;cnt < num_of_buttons;cnt++)
                {
                        sprintf(value[cnt+num_of_axis], "%d",button[cnt]);
                }
                usleep(100);
                bzero(jsdata,sizeof(jsdata+1));
                for(cnt = 0;cnt < (num_of_buttons+num_of_axis);cnt++)
                {
                        sscanf(value[cnt], "%d",&jsdata[cnt]);
                }
                jsdata[1] = -1*jsdata[1];
                //printf("%d %d %d\n",jsdata[0],jsdata[1],jsdata[2]);
}
}
void *write_msg (void *msg)
{
                        //sprintf(buffer,"%d",jsdata[1]);
                        n = write(sockfd,buffer,strlen(buffer));
                        bzero(buffer,sizeof(buffer+1));
                        usleep(75000);
}
int js_tele()
{
        if( ( joy_fd = open( JOY_DEV , O_RDONLY)) == -1 )
{
                printf( "Couldn't open joystick " );
        }
        ioctl( joy_fd, JSIOCGAXES, &num_of_axis );                              //      GET THE
NUMBER OF AXIS ON JS
        ioctl( joy_fd, JSIOCGBUTTONS, &num_of_buttons );        //      GET THE NUMBER OF
BUTTONS ON THE JS
        ioctl( joy_fd, JSIOCGNAME(80), &name_of_joystick );     //      GET THE NAME OF THE JS

        axis = (int *) calloc( num_of_axis, sizeof( int ) );
        button = (char *) calloc( num_of_buttons, sizeof( char ) );
        printf("numAxis, %d   ", num_of_axis);
        printf("numButtons, %d\n", num_of_buttons);
```

```
//      CHANGE THE STATUS FLAG OF THE FILE DESCRIPTOR TO NON-BLOCKING MODE
        fcntl( joy_fd, F_SETFL, O_NONBLOCK );
                while(1)
                {
                //usleep(10000);
        //      READ THE JOYSTICK STATE, IT WILL BE RETURNED IN THE JS_EVENT STRUCT
                read(joy_fd, &js, sizeof(struct js_event));
        //      CHECK THE EVENT
                switch (js.type & ~JS_EVENT_INIT){
                        case JS_EVENT_AXIS:
                                axis   [ js.number ] = js.value;
                                break;
                        case JS_EVENT_BUTTON:
                                button [ js.number ] = js.value;
                                break;
                }
                //      PRINT THE RESULTS
                for(cnt = 0; cnt < num_of_axis; cnt++)
                {
                        sprintf(value[cnt], "%d", axis[cnt]);
                }
                for(cnt = 0;cnt < num_of_buttons;cnt++)
                {
                        sprintf(value[cnt+num_of_axis], "%d",button[cnt]);
                }
                usleep(100);
//              printf("%s %s %s\n",value[0],value[1],value[2]);
                bzero(jsdata,sizeof(jsdata+1));
                for(cnt = 0;cnt < (num_of_buttons+num_of_axis);cnt++)
                {
                        sscanf(value[cnt], "%d",&jsdata[cnt]);
                }
                jsdata[1] = -1*jsdata[1];
                // Determine joint angles
                if (jsdata[1] == 0)
                        {
                        printf("Ignore\n");
                        }
                int count = 0;
                if (jsdata[5]==1)
                {
                printf("Button 5\n");
                bzero(buffer,sizeof(buffer+1));
                while ((jsdata[1] >= -32767)&&(jsdata[1] <= 32767))
                {
                        float jnt_posd[3];
                        jnt_posd[0] = jsdata[1]*(180.0/65534.0);
                        printf("%f\n",jnt_posd[0]);
```

```
		int tmpcmd = 4;
		int jntid = 6;
		pthread_t threads1, threads2;
		int t = 0;
			pthread_create(&threads1, NULL, event_read, (void *)t);
			sprintf(buffer,"%d %d %f",tmpcmd,jntid,jnt_posd[0]);
			pthread_create(&threads2, NULL, write_msg, (void *)t);
			pthread_join(threads2,NULL);
		count++;
		printf("First -ve: %d and count: %d\n",jsdata[1],count);
		if (jsdata[5] != 1)
		{
		pthread_detach(threads2);
		break;
		}
}
}//Button 5
if (jsdata[6]==1)
{
printf("Button 6\n");
bzero(buffer,sizeof(buffer+1));
while ((jsdata[1] >= -32767)&&(jsdata[1] <= 32767))
{
		float jnt_posd[3];
		jnt_posd[0] = (jsdata[1]*(260.0/65534.0))-20.0;
		printf("%f\n",jnt_posd[0]);
		int tmpcmd = 4;
		int jntid = 7;
		pthread_t threads1, threads2;
		int t = 0;
			pthread_create(&threads1, NULL, event_read, (void *)t);
			pthread_create(&threads2, NULL, write_msg, (void *)t);
			sprintf(buffer,"%d %d %f",tmpcmd,jntid,jnt_posd[0]);
			pthread_join(threads2,NULL);

		count++;
		printf("First -ve: %d and count: %d\n",jsdata[1],count);

		if (jsdata[6] != 1)
		{
		pthread_detach(threads2);
		break;
		}
}
}//Button 6
if (jsdata[7]==1)
{
printf("Button 7\n");
```

```
                while ((jsdata[1] >= -32767)&&(jsdata[1] <= 32767))
                {
                        float jnt_posd[3];
                        jnt_posd[0] = (jsdata[1]*(95.0/65534.0))-42.5;
                        printf("%f\n",jnt_posd[0]);
                        int tmpcmd = 4;
                        int jntid = 8;
                        pthread_t threads1, threads2;
                        int t = 0;
                                pthread_create(&threads1, NULL, event_read, (void *)t);
                                pthread_create(&threads2, NULL, write_msg, (void *)t);
                                sprintf(buffer,"%d %d %f",tmpcmd,jntid,jnt_posd[0]);
                                pthread_join(threads2,NULL);
                        count++;
                        printf("First -ve: %d and count: %d\n",jsdata[1],count);
                        if (jsdata[7] != 1)
                        {
                        pthread_detach(threads2);
                        break;
                        }
                }
                }
}
        return (0);
}
#endif
```

A.VIII server.c

File server.c is coded for running TCP/IP server for remotely control of the robotic manipulator.

```
/* A simple server in the internet domain using TCP
  The port number is passed as an argument */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#include "SchunkData.h"

#ifndef _Schunk_FNC_H
```

```
#include "Schunk_FNC.h"
#endif

void error(const char *msg)
{
   perror(msg);
   exit(1);
}

int main()
{
tv.tv_sec = 0; /* seconds */
tv.tv_usec = 0;

int ret = -1;
bool server = true;
//    char buffer_send[1024];
//    int sockfd, newsockfd, portno;
//    int n,k;
    printf ("Insert Port no: ");
    scanf ("%d",&portno);


    FD_ZERO(&read_msg);


    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
          error("ERROR opening socket");
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);

    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
    {
          error("ERROR on binding");
    }
    listen(sockfd,5);
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    bzero(buffer,sizeof(buffer));
    bzero(buffer_send,sizeof(buffer_send));

ret = Schunk_Init();
```

```
 if(ret != 0)
 {
  printf("Schunk Initialization failed\n");
  Schunk_Cleanup();
  running = false;
 }
 else
 {
        printf("Initialization Successful\n");
 }

while (server == true)
{
   if (newsockfd < 0)
   {
        error("ERROR on accept");
   }

   n = read(newsockfd,buffer,sizeof(buffer));

   if (n < 0)
   {
        error("ERROR reading from socket");
   }
   if (buffer[0] == '\0')
        {
        printf ("Client disconnected\n");
        Schunk_Cleanup();
        break;
        }

   printf("Here is the message: %s\n",buffer);
   sscanf (buffer,"%d\n",&cmd);

while(cmd >= 1000)
{
        if (cmd == 1000)
                {
                online = true;
                task_plan = false;
                }
        if (cmd == 1001)
                {
                task_plan = true;
                online = false;
                }
   bzero(buffer,sizeof(buffer));
   n = read(newsockfd,buffer,sizeof(buffer));
```

```
    if (n < 0)
    {
        error("ERROR reading from socket(in while)");
    }
    sscanf (buffer,"%d\n",&cmd);
    printf("Here is the message(in while): %s\n",buffer);
    break;
}

    if (buffer[0] == '\0')
        {
        printf ("Client disconnected\n");
        Schunk_Cleanup();
        break;
        }

//    n = write(newsockfd,"I got your message",18);
//    if (n < 0) error("ERROR writing to socket");

        switch (cmd)
        {

            case SCHUNK_INITIALIZE:
            {
            ret = Schunk_Init();
                if (ret != 0)
                {
                    printf ("Error\n");
                    k = write(newsockfd,"-1",2);
                }
                else
                {
                    conf = 1;
                    k = write(newsockfd,"1",1);
                }
            bzero(buffer,sizeof(buffer));
            break;
            }

            case SCHUNK_CLEANUP:
            {
            ret = Schunk_Cleanup();
                if (ret != 0)
                {
                    k = write(newsockfd,"-1",2);
                    printf ("Error\n");
                }
                else
```

```
        {

                conf = 1;
                k = write(newsockfd,"1",1);
        }
bzero(buffer,sizeof(buffer));
break;
}

case SCHUNK_RESET_ALL:
{
ret = Schunk_Mov_Reset_All();
        if (ret != 0)
        {
                printf ("Error\n");
                k = write(newsockfd,"-1",2);
        }
        else
        {
                conf = 1;
                k = write(newsockfd,"1",1);
        }
bzero(buffer,sizeof(buffer));
break;
}

case SCHUNK_MOV_LOCK:
{
send_pos = false;
if (online)
        k = write(newsockfd,"1",1);
ret = Schunk_Mov_Lock();
        if (ret != 0)
        {
                printf ("Error\n");
                Schunk_Cleanup();
                k = write(newsockfd,"-1",2);
        }
        else
        {
                printf ("Schunk Lock pos \n");
                if (task_plan)
                        k = write(newsockfd,"1",1);
                conf = 1;
        }
bzero(buffer,sizeof(buffer));
break;
}
```

```
case SCHUNK_MOV_UNLOCK:
{
send_pos = false;
if (online)
        k = write(newsockfd,"1",1);
ret = Schunk_Mov_Unlock();

        if (ret != 0)
        {
                printf ("Error\n");
                Schunk_Cleanup();
                k = write(newsockfd,"-1",2);
        }
        else
        {
                printf ("Schunk Unlock pos \n");
                if (task_plan)
                        k = write(newsockfd,"1",1);
                conf = 1;
        }
bzero(buffer,sizeof(buffer));
break;
}

case SCHUNK_SET_PARA:
{
send_pos = true;
cnt = 0;
float shoulder, elbow, wrist;
sscanf (buffer,"%*d %f %f %f\n",&shoulder, &elbow, &wrist );
if (online)
        k = write(newsockfd,"1",1);
ret = Schunk_Mov_All_Joints(shoulder, elbow, wrist);
        if (ret != 0)
        {
                printf ("Error\n");
                Schunk_Cleanup();
                k = write(newsockfd,"-1",2);
        }
        else
        {
                printf ("schunk_set_param\n");
                if (task_plan)
                        k = write(newsockfd,"1",1);
                conf = 1;
        }
bzero(buffer,sizeof(buffer));
```

```
break;
}

case SCHUNK_MOV_JNT:
{
send_pos = false;
int modID;
float posd;
sscanf (buffer,"%*d %d %f\n",&modID,&posd);
ret = Schunk_Mov_Jnt_Pos(modID,posd);
        if (ret != 0)
        {
                printf ("Error\n");
                Schunk_Cleanup();
                k = write(newsockfd,"-1",2);
        }
        else
        {
                printf ("schunk_mov_jnt\n");
                k = write(newsockfd,"1",1);
                conf = 1;
        }
bzero(buffer,sizeof(buffer));
break;
}

case SCHUNK_HALT_ALL:
{
ret = Schunk_Mov_Halt_All();

        if (ret != 0)
        {
                printf ("Error\n");
                Schunk_Cleanup();
                k = write(newsockfd,"-1",2);
        }
        else
        {
                printf ("halt all joints\n");
                k = write(newsockfd,"1",1);
                conf = 1;
        }

bzero(buffer,sizeof(buffer));
break;
}

case SCHUNK_MOV_RETRIEVE:
```

```
{
send_pos = true;
cnt = 0;
if (online)
        k = write(newsockfd,"1",1);
ret = Schunk_Sample_Retrieve();
        if (ret != 0)
        {
                printf ("Error\n");
                Schunk_Cleanup();
                k = write(newsockfd,"-1",2);
        }
        else
        {
                printf ("Retrieving Sample\n");
                if (task_plan)
                        k = write(newsockfd,"1",1);
                conf = 1;
        }
bzero(buffer,sizeof(buffer));
break;
}

case SCHUNK_MOV_DUMP:
{
send_pos = true;
cnt = 0;
if (online)
        k = write(newsockfd,"1",1);
ret = Schunk_Sample_Dump();

        if (ret != 0)
        {
                printf ("Error\n");
                Schunk_Cleanup();
                k = write(newsockfd,"-1",2);
        }
        else
        {
                printf ("Dumping Sample\n");
                if (task_plan)
                        k = write(newsockfd,"1",1);
                conf = 1;
        }
bzero(buffer,sizeof(buffer));
break;
}
```

```
case SCHUNK_GET_JNT_POS:
{
send_pos = false;
int modID;
float posd,pos;
sscanf (buffer,"%*d %d\n",&modID);
ret = Schunk_Get_Jnt_Pos(modID,&posd);
printf ("Schunk Get Pos: \n");
        if (ret != 0)
        {
                printf ("Error\n");
                Schunk_Cleanup();
                k = write(newsockfd,"-1",2);
        }
        else
        {
                pos = (float)posd;
                conf = 1;
                sprintf (buffer_send,"%d %f",conf,pos);
                k = write(newsockfd,buffer_send,strlen(buffer_send));
                printf ("Pos_temp: %f\n",pos);

        }
bzero(buffer,sizeof(buffer));
bzero(buffer_send,sizeof(buffer_send));
break;
}

case SCHUNK_GET_MAST_CFG:
{
send_pos = false;
float poscfg[3];
ret = Schunk_Get_Mast_Cfg(poscfg);
        if (ret != 0)
        {
                printf ("Error\n");
                k = write(newsockfd,"-1",2);
                Schunk_Cleanup();
        }
        else
        {
                printf ("Schunk Get Mast Cfg: \n");
                conf = 1;
                sprintf (buffer_send,"%d %f %f %f",conf,poscfg[0],poscfg[1],poscfg[2]);
                k = write(newsockfd,buffer_send,strlen(buffer_send));
        }
bzero(buffer,sizeof(buffer));
```

```
                bzero(buffer_send,sizeof(buffer_send));
                break;
                }


                case QUIT:
                {
                        server = false;
                        printf ("Server false\n");
                        Schunk_Cleanup();
                        break;
                }

                default:
                {
                        printf("Unknown request command\n");
                        Schunk_Cleanup();
                        break;
                }

        }
}// server == running
    close(newsockfd);
    close(sockfd);
    return 0;
}
```