Design of a Graphical User Interface to Augment a Telerobotic Stereo-Vision System

BY

Syed Mohammed Shamsul Islam

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

In

Computer Engineering

December 2005

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **SYED MOHAMMED SHAMSUL ISLAM** under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING.**

Thesis Committee

Dr. MAYEZ AL–MOUHAMED (ThesisAdvisor)

Dr. MUHAMMAD SARFRAZ (Member)

Dr. TALAL AL–KHAROBI (Member)

Department Chairman
Dr. Jarallah S. AlGhamdi

Dean of Graduate Studies
Dr. Mohammad A. Al-Ohali

Date 6-2-2006

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES

Heartily dedicated to my **parents**

Syed Nazrul Islam and Mosammat Jebunnesa

and also to my **elder brother** Syed Rafiqul Islam
whose prayers, guidance and encouragement led me to the
accomplishment of this work.

Acknowledgements

All praise is due to Allah (SWT), the Most Gracious, the Most Merciful. May peace and blessings be upon Prophet Muhammed (SAW), his family and his companions.

I pay a heartily tribute to all of my family members and especially to my parents, who guided me during all my life endeavors. Their love and support motivated me to continue my education and achieve higher academic goals. Without their moral support and sincere prayers, I would have been unable to accomplish this work.

I would like to express my deepest gratitude and appreciation to my advisor Dr. Mayez Al-Mouhamed, Professor of Computer Engineering for his consistent help, fatherly guidance and cautious attention throughout the course of this work. His valuable suggestions and useful discussions made this work interesting for me. My than ks also go to my thesis committee members - Dr. Muhammad Sarfraz, Professor of Information and Computer Science and Dr. Talal Al-Kharobi, Assistant Professor of Computer Engineering. Both of them provided me with invaluable and prompt feedback and guidelines to improve the quality of the work. I would also like to thank Dr. Muhammad Farrukh Khan, former Assistant Professor of Computer Engineering, who served as a committee member for a good amount of time and extended his kind co-operation and gave many important suggestions.

It would like to express my cordial thanks and profound gratitude to Dr. Onur Toker, former Associate Professor, Department of System Engineering, KFUPM. I

will be always indebted to him for his scholarly suggestions and patient co-operation by spending lots of hours in day and night during his visit to KFUPM and also in communicating with emails. I am also indebted to Dr. Syed Mohamed Buhari, Lecturer of Information and Computer Science for his kind co-operation, specially in the implementation phase of the work.

I would like to acknowledge the support and facilities provided by College of Computer Science and Engineering and particularly, the Department of Computer Engineering, King Fahd University of Petroleum and Minerals (KFUPM) for the completion of this work.

Finally, I appreciate the friendly support and encouragement from all my colleagues at KFUPM. In particular, I want to thank Mr. Mahedi, Asif, Nazeeruddin, Sumon, Sanaullah, Abid, Shazli, Fahimuddin, Shareef, A. Kafi, Shibly, Tareq, Kaosar and Shahed.

Contents

List of Tables

List of Figures

THESIS ABSTRACT

Name: Syed Mohammed Shamsul Islam

Title: Design of a Graphical User Interface
to Augment a Telerobotic Stereo-Vision System

Degree: Master of Science

Major Field: Computer Engineering

Date of Degree: December 2005

Telerobotics as a multidisciplinary area that aims at extending eye-hand motion co-ordination through a distance using real-time wired or wireless computer networks. It consists of a master arm to describe hand motion, slave arm (the robot arm) to reproduce hand motion, and stereo vision, haptic and force feedback to extend human depth perception and senses. Some of the telerobotic applications are operating in deep space, underwater, nano and micro scales, carrying out surgery inside the patient body, etc. One of the most critical problems of such system is the communication delay that often causes teleoperation instability. In this work, an Augmented Reality (AR)-based telerobotic system is proposed that allows developing and validating a teleoperation plan by superimposing some virtual objects onto the real video image of the workspace. The plan substitutes frequent low-level interactions between the user and the remote site by sending only the finalized data and thus, reduces real-time network interactions. It also increases the task safety. To develop the system, a serial six DOF 3D graphical arm is designed based on its geometric model. Then, accurate camera calibration and registration methods are used to superimpose the graphical arm onto the video image. Motion activation algorithms are developed and a Graphical User Interface (GUI) is designed to facilitate the task simulation. Direct 3D of Microsoft DirectX is used as graphics API that exploits hardware acceleration for graphics rendition through Hardware Abstraction layer (HAL). The overall system is designed using Microsoft .NET framework with Visual C#.NET programming and tested on a stereovision system operating over a LAN with a PUMA-560 robot. The system performed well with significant accuracy

and refresh rate of graphics on the output screen.

Master of Science Degree

King Fahd University of Petroleum and Minerals, Dhahran.

December, 2005

ملخص الرسالة

الا ســــــــم: سيد محمد شمس الإسلام

عنــوان الـرسالــة: تصميم واجهة مستخدم رسومية لتعزيز نظام الرؤية الثنائية في أنظمة الأذرع الآلية العاملة عن بعد

الـتـخصـــص: هندسة الحاسب الآلي

تـاريـخ الـتـخـرج: ذو القعدة 1426هـ

تهدف أنظمة الأذرع الآلية العاملة عن بعد إلى نقل الحركة المتوائمة مع الرؤية عبر شبكات الحاسب (السلكية أو اللاسلكية). تتكون هذه الأنظمة من ذراع آمر يحركه المستخدم وذراع منفّذ يقوم باعادة تمثيل الحركة. يعزز النظام برؤية ثنائية ونظام تغذية عكسي لنقل ردة الفعل إلى المستخدم. تستخدم أنظمة الأذرع الآلية العاملة عن بعد في العديد من التطبيقات مثل العمل في الفضاء الخارجي و تحت الماء والتقنيات الدقيقة والمتناهية الدقة وكذلك في العمليات الجراحية. أحد أهم المعوقات التي تؤثر سلباً على أداء أنظمة الأذرع الآلية العاملة عن بعد هو التأخير في تبادل المعلومات بين طرفي النظام مما يسبب فقدان الثبات. في هذه الرسالة نطرح نظام أذرع آلية يعمل عن بعد يمكّن المستخدم من التخطيط و تجربة المهمة المطلومة قبل تنفيذها الفعلي. يقوم النظام بإظهار صورة تخيلية لخطة المهمة على صورة الفديو الحقيقية لموقع العمل. تجربة المهمة قبل تنفيذها الفعلي يزيد من درجة سلامة العملية بالإضافة الى تقليل كمية المعلومات المتبادلة وذلك بتنفيذ الخطة النهائية للمهمة. لتطوير هذا النظام قمنا بعمل نموذج هندسي ثلاثي الأبعاد لذراع آلي ذو 6 درجات حرة. بمعايرة دقيقة للكاميرا أمكننا إضافة صورة تخيلية للذراع الآلي على صورة الفديو الحقيقية. ولتعزيز عملية المحاكاة قمنا بتطوير واجهة مستخدم رسومية وخوارزميات لتفعيل الحركة. تم تصميم النظام المقترح باستخدام بيئة ميكروسوفت وبرامج C#, .NET, DirectX9. تم اختبار النظام باستخدام ذراع بوما 560 ونظام رؤية ثنائي وتم تبادل المعلومات عبر شبكة حاسب محلية. أعطى النظام نتائج جيدة بدقة أفضل في عرض الرسوم على الفديو.

درجة الماجستر في العلوم
جامعة الملك فهد للبترول و المعادن
الظهران- المملكة العربية السعودية
ذو القعدة 1426 هـ

Chapter 1

Introduction

In this chapter, the preliminary concepts required to understand the thesis problem is first described. Then, the problems involved in the conventional approach and probable solutions are mentioned. The thesis objectives and the technical platform used to meet the objectives are also described.

1.1 Preliminary Concepts

In this section, some of the basic concepts will be made clear which will let us understand the underlying problem of our work and the thesis objectives to be discussed later in this chapter.

1.1.1 Telerobotics

Telerobotics is a modern technology of robotics that extends an operator's sensing and manipulative capabilities to a remote environment. A telerobotic system consists of a master arm (workstation) and a slave arm (workstation) that are connected through a computer network and a stereo-vision system to provide 3D views of slave scene; see Fig. 1.1. Tele-operator is also provided with force feedback to have a better sense of his task manipulation.

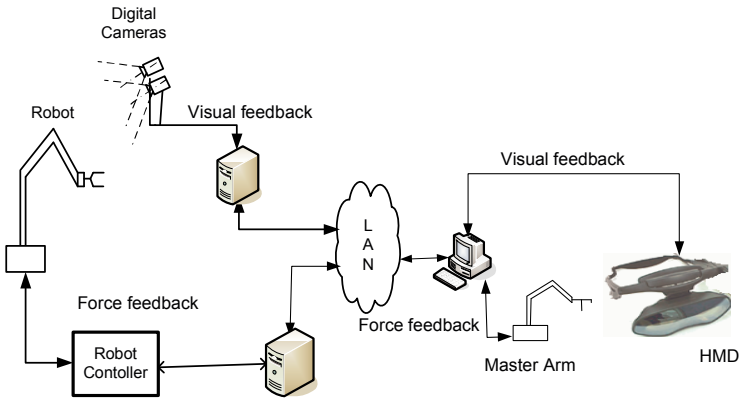

Figure 1.1: A typical telerobotic stereo-vision system.

Telerobotics is now becoming very useful to be applied in many situations specially in scaled down and scaled up situations, hazardous and hostile situations and environment where human presence adversely affect the task operation. Telerobotics has enhanced the surgery through improved precision, stability and dexterity. Stereo image guided telerobots allow surgeons to work inside the patient's body precisely and without making large incision. Telerobots are now routinely used for biopsy

brain lesions with minimal damage to adjacent tissue, for closed-chest heart bypass, for shaping the femur to precisely fit prosthetic hip joint in orthopedic surgery, for microsurgical procedures in ophthalmology and for surgical training.

1.1.2 Stereo Vision

Stereo-vision is a technique to get 3D perception of a remote scene. In a stereo-vision system, visualization of a remote scene is made in such a way that the viewer has clear idea about the relative distances, depths and dimensions of the objects present in the stereo image [1]. These advantages of stereo vision over monoscopic vision are well-described in [5]. Stereovision has a wide range of potential application areas including 3D map building, data visualization and robot pick and place.

1.1.3 Augmented Reality

Augmented Reality (AR) is a process of superimposing one or more registered computer graphics objects, or 3D virtual objects (3DVOs) over the stereo views of a real scene to augment the stereo space and thus enhancing visual information (video in general). Thus, AR is a variation of Virtual Reality (VR) in a sense that it supplements reality, rather than completely replacing the reality. According to [6], AR is a system that has the following three characteristics:

1. combines real and virtual,

2. interactive in real time, and

3. registered in 3D Area-based stereo

Therefore, only overlaying two-dimensional (2D) virtual objects on the real world as done in films like "Jurassic Park" could not be considered AR since they are not interactive media.

In robotics, AR lies between telepresence (completely real) & Virtual reality (completely synthetic) and between manual teleportation & autonomous robotics [7].

A comprehensive survey on AR is made in [6] and [8] which explored number of applications of AR including medical visualization, maintenance & repair, annotation, robot path planning, entertainment and military aircraft navigation & targeting, interior design and many more. It increases reliability and correctness in tele-surgery and give assistance to the surgeons.

1.2 Problem of the Conventional Tele-Operation

The conventional teleoperation suffers from number of problems. As discussed in [5], this method requires the operator continually engaged in dynamic control of the robot whenever any actions are to be executed. This may lead to the operator himself retarding the system due to relatively slow sensory and motor capabilities.

Also in most cases, operational environment is unstructured and unpredictable where repetitive programmed procedures may not work. Another, the most important problem is the time-delay in communication between local and remote site. Due to time delay operator has to go often for a move-n-strategy. Maintaining large bandwidth may reduce this delay but it is expensive and sometimes cumbersome. All these issues should be handled properly in order to apply telerobotics for successful tele-surgery.

1.3 Application of AR for Solving the Problem of Telerobotics

Various techniques such as predictive feedbacks, supervisory control and most recently AR have been proposed to solve the problems [1]of conventional telerobotics. Information from various medical imaging sensors, such as Computer Aided Tomography (CAT), Positron Emission Tomography (PET), and Nuclear Magnetic Resonance (NMR) scanners can be used to generate graphic images of the interior of the human body. These images can be super-imposed onto a live video image of the body using AR tools, and seen in three dimensions (3Ds), providing a clear advantage over the systems that use flat 2D displays.

AR also provides the tele-operator e.g. surgeon with the facility of making simu-

lation plan with probable rehearsal and corrections before going for exact operation with patient's body. That means, AR allows task specification (requiring human intelligence) to be separated from task execution (that can be done by autonomous modules) [2]. Operator (for example surgeon) can preview the effect of the move (action) on the local display overlaid on the remote world image (i.e. patient's body image) and once he is satisfied with the move, he can issue the actual command. Thus, it will give additional safety in task space which is crucial specially in tele-surgery.

The graphical overlaying also helps to overcome the adverse effect of communication delay and saving bandwidth by sending (less frequently) only the finalized planned trajectory points. In [9], it is reported that the researchers conducted human factors experiments using a PUMA robot to perform a simple tapping task with time delays up to four sec. Results showed about 50% reduction in task completion time when using a phantom robot (a virtual robot properly registered with the position of its real counter part) instead of simple teleoperation.

1.4 Challenges of AR to be Applied in Telerobotics

To apply AR on a telerobotic stereovision system, it requires proper overlay of real world i.e. robot workspace scene data (e.g. patient's anatomy in case of surgery) onto the graphics image data. Proper overlaying, on the other hand, requires establishing a bidirectional one-to-one mapping of coordinate spaces between the virtual world and the remote world viewed through the video; see Fig. 1.2. In stereo vision system, this requires respective mappings of both right and left video frames. This can be done by proper camera calibration and image registration processes.

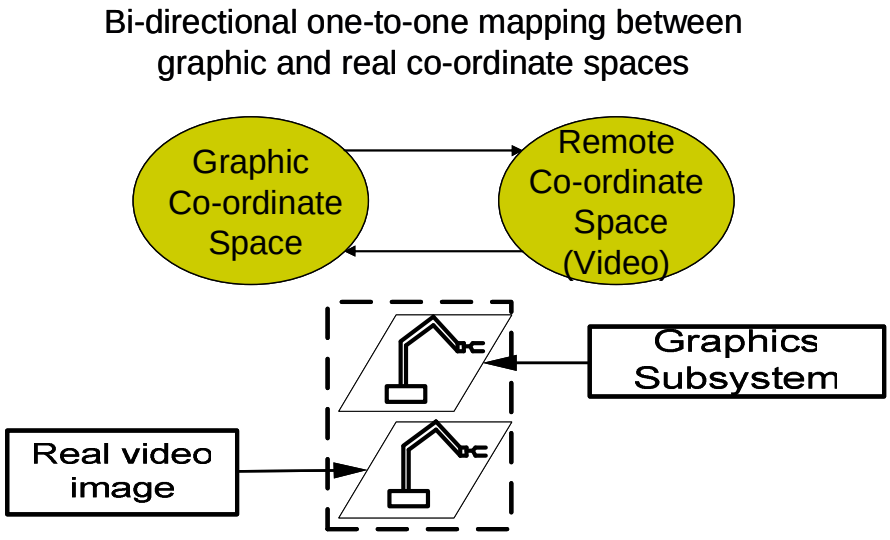

Figure 1.2: Requirement of AR.

Camera calibration is the establishment of the projection from the 3D world co-ordinates to the 2D image co-ordinates by finding the intrinsic and extrinsic camera

parameters [10]. Intrinsic parameters include optical and electronic properties of a camera, such as focal length, lens distortion coefficients, image center, scaling factors of the pixel array in both directions. While the extrinsic parameters are the pose estimation (rotation and translation) of the camera system relative to a user-defined 3D world coordinate frame [2]. In multi-camera systems, the extrinsic parameters also describe the relationship between the cameras [11].

Registration refers to the proper alignment of the virtual object with the real world. The accuracy of registration is mostly dependant on the accuracy of calibration [10]. Two kinds of registrations are: static and dynamic [12, 2]. In the static registration, user and the objects in environment remain still. It is done at initialization with the help of Human Operator. The dynamic registration is done while the viewpoint starts moving to automatically update the registration data.

The main sources of static errors are optical distortion, errors in the tracking system, mechanical misalignments and incorrect viewing parameters (e.g., field of view, tracker-to-eye position and orientation, interpupillary distance). A detail on static errors and algorithms to rectify them can be found in [13, 12, 14, 6].

The main cause of dynamic error is the system delay or lag [12]. The end-to-end system delay is defined as the time difference between the moment that the tracking system measures the position and orientation of the viewpoint to the moment when the generated images corresponding to that position and orientation appear in the

displays [6]. However, dynamic errors can be reduced by reducing system lag and apparent lag, matching temporal streams (with video-based systems) and predicting future locations [1, 15].

1.5 Thesis Objectives

1. To review camera models and techniques used to provide mapping between 3D space and camera reference as well as associated calibration methods and adopting an existing method or developing a new one.

2. To study various graphics rendering systems and to choose one for drawing and displaying virtual objects.

3. To incorporate the concept of augmented reality into the user's operating environment i.e. to display virtual objects on the real stereo image.

4. To create a graphical user interface by incorporating additional display and control features for simple graphical manipulation in the telerobotic AR system.

5. To carry out performance evaluation of the proposed augmented stereo-vision system.

1.6 Technical Platform

The programming language and the operational equipment used in work will be described in this section. We have chosen the Microsoft .NET framework for our core application framework and Microsoft DirectX 9.0 for the graphics implementation purposes. All the software and hardware used in this thesis work are enlisted as below.

1.6.1 Hardware used

We have used PUMA-560, an industrial robot arm. The client and server are run on two PCs having 2-GHz Intel P4 processors with 1GB DRAM and 512 KB cache memory. Control of master and slave arms is done using Eagle PCI 30FG data acquisition cards. Each of client and server PCs is attached to a campus network by using a 100 Mbps NIC card (3com EtherLink XL PCI). The server PC is interfaced to two Sony Handycam digital cameras using a 400 Mbps FireWire PCI (IEEE-1394) card. The client PC uses an NVIDIA GetForce4 Ti4600 as display adaptor to interface with an SVGA resolution Cy-visor DH-4400VP 3D Head-Mounted Display.

1.6.2 Software used

Both client and server PCs run under MS Window 2000. The vision server software uses MS Visual C++ with .NET framework 1.1 under Microsoft development

environment 2003. The imaging device driver used is Microsoft DV camera and VCR. The PUMA server and the client is implemented using MS Visual C# with the above .NET framework.

Graphics design is implemented using Microsoft DirectX9. MATLAB calibration tool is used for camera calibration purposes. MATLAB is also used for analyzing performance data.

1.7 Organization of the Thesis

A brief literature review of the works pertinent to the thesis objectives is provided in chapter 2. The design methodology and implementation aspects are described in chapter 3 and 4 respectively. Then, in chapter 5 the performance evaluation and some comparative studies are described. We conclude in chapter 6.

Chapter 2

Literature Review

This chapter presents a review of existing tools, techniques and methods proposed for augmentation of a telerobotic stereovision system. At first, ways and means of 3D stereo-visualization are described. Then, some camera calibration and registration methods are discussed. Some complete telerobotic systems with AR are also presented followed by a comparative study. Our findings are summarized at the end of the chapter.

2.1 Stereo Vision

A stereovision does not produce true 3D images, but it provides a 3D effect by presenting a different view to each eye of an observer so that scenes do appear to have depth [16]. In stereo vision system, two cameras are used to take two 2D pho-

tographs of the same scene at slightly different angles. Then, these two photographs are presented for stereoscopic view using various stereoscopic devices/techniques. Stereovision system enhances operator's efficiency during telemanipulation [17]. In this section, we will discuss various types of stereo systems and stereoscopic devices found in the literatures.

2.1.1 Output Devices for 3D visualization

There are mainly two types of 3D visualization output devices in stereo vision system which are described as follows.

1. Shuttering glasses [1]: The glasses alternately shut or block the viewer's left, then right eyes from seeing an image and thus a stereoscopic image is alternatively shown in sequence left-image, right-image in sympathy with the shuttering of the glasses. Problem with device is that the user can experience the annoying phenomena of flickering which can effect his or her ability to control the robotic arm at low refresh frequencies. But as most of the available monitors and display adapters can support refresh frequencies equal or above 120 Hz at resolutions of 1024x768 or above, 3D visualization with very high detail is possible with most shuttering glasses [18, 19]. For example, the *Eye3D Premium* shuttering glasses can support resolutions (in pixels) up to 2048 x 1538 at 120 Hz, and 1856 x 1392 at 140 Hz.

2. Head mounted displays (HMD) [1, 20, 21]: HMDs provide a much larger virtual monitor size for the user, usually in the range of 2 meters large. They are more comfortable to work with, forces to use to see the 3D object and nothing else, and there is no problem of flickering. Most of them support the INTERLACED 3D video format, but not the so called ABOVE/BELOW format which is robust under video compression and resizing. Most HMDs also support page flipping, but this requires special drivers for each display adapter/chipset. Some HMDs are also equipped with ear-phones and head trackers. However, their main disadvantage is that their resolutions are either VGA or SVGA (at least the ones which are commercially available during this period of time). But compared to shuttering glasses, they are a factor of 10-20 or more times expensive, yet they are limited to SVGA resolutions.

2.1.2 3D Visualization Techniques

There can be different methods to produce 3D effects on the client side. Among them two mostly used methods summarized in [1] are described below:

1. Sync-Doubling: This is the most effective 3D presentation method. It does not require any special device inside the computer. We only need to arrange the left and right eye images up and down on the computer screen. A sync-doubler sits or hooks between the display (VGA) output from the PC and the

monitor to insert an additional frame v-sync between the left and right frames (i.e. the top and bottom frames). This allows the left and right eye images to appear in an interlaced pattern on screen. Using the frame v-sync as the shutter alternating sync allows us to synchronically transmit the right and left frames to respective left and right eyes, thus creating a 3D image. This method is not limited by the computer hardware specs or by the capabilities of the monitor. But it is limited in a way that we get only half of the resolution of the screen for the 3D image.

2. Page Flipping: Page-flipping means alternately showing the left and right eye images on the screen. Combining the 3D shuttering glasses with this type of 3D presentation only requires the application of frame v-sync as the shutter alternating sync to create a 3D image. Page-flipping requires higher hardware specifications. Since synchronized registration of left and right eye frames is necessary, the minimum capacity of its frame buffer is twice as usually required. In order to overcome the *flashing* problem of 3D imaging, frames provided should be at least 60 frames per second; hence v-scan frequency should be 120Hz or higher. As it involves hardware frame buffer and page-flipping synchronization, it often requires specially designed hardware for double-buffering the stereo image.

   Page-flipping provides full resolution picture quality, hence it has the best

visual effect among all available 3D display modes. But being highly dependent on software and hardware is the biggest drawback of this technique [1].

2.1.3 3D Visualization systems

There are variety of different ways to generate 3D video content as mentioned in the online document, Eye3D Manual [22]. Some of the visualization systems chosen and described in [1] are discussed below:

1. Parallel camera configuration [23]: This is a very commonly used technique for 3D video generation. In this system, a 3D object can be observed with high accuracy under magnification and dept and requires simpler computation than the tilted case. However, in case of the near stereoscopic viewing, it shows some problems. Most of the time, some sort of video mixer may be required to convert two video streams into a single synchronized stream.

2. Tilted camera configuration [24, 25, 26]: this configuration provides a larger area of stereoscopic vision, such that the total area for 3D display is more, the depth resolution is enhanced, and near stereoscopic viewing is better than the parallel configuration. But, the computational aspects are more complicated and demanding. Again, it produces more accuracy in the horizontal direction than in the vertical direction. However, this problem can be overcome by using different horizontal and vertical scaling factors. In this case, also some sort of

video mixer may be required to convert two video streams into a synchronized single stream.

3. NuView 3D adapter: It is a simple and practical solution to 3D video generation consisting of two LCD-shutters, a prismatic beam splitter and an adjustable mirror. Two images are seen by watching through the Nu-View, while it is switched off. The mirror/prism system puts the camera lens into the center of the light rays of a left and a right eye view. The shutters allow the camera lens to get only one of the views at a time. The adaptor is connected to the video-out port of the camcorder. This way the shutter can synchronize to the recording (50 or 60 Hz). But while zooming to the widest angle, parts of the NuView adapter may appear in the frame, producing a dark border and it produces some ghosting in hi-contrast scenes. Further details can be found at the online documentation at [21].

2.2 Augmented Reality Technologies

In literatures, two different types of technologies: optical and video have been proposed for combining the real and the virtual worlds in computer vision-based AR system [1]. In the optical AR equipments, the operator gets a direct view of the real world while the virtual objects are super-imposed on optical see through mirrors in front of his eyes. In video-based equipment, the operator does not have any direct

view of the real world. Instead, he has to use the video input from the camera altered by the local scene generator in order to add virtual objects to the scene. The advantages and disadvantages and other details of both the techniques can be found in [6].

2.3 Camera Calibration

Camera calibration is a necessary step in AR system in order to extract metric information from 2D images. Much work has been done, starting in the photogrammetry community and more recently in computer vision. In this section, the proposed methods will be classified and described after a short discussion on various camera models.

2.3.1 Classification of Camera Models

In computer vision and other related subjects, there are numerous different camera models which model the imaging process by mapping points in the world to positions on the image plane. These models can be categorized into two groups:

1. The '$tow-plane$' Model: This model is proposed by [27]. It takes into account all geometric and optical camera features. It establishes the correspondence between a point in the image and the same point in the scene. But since

obtained rays do not cut each other, this model is not suited for stereo vision where scenes are captured by video cameras involving perspective projection.

2. The Pin-hole Model: The ideal camera model commonly used in computer graphics and computer vision to capture the imaging geometry is a simple pinhole camera model. It defines the basic projective imaging geometry with which the 3D objects are projected onto the 2D image surface. This model can further be classified into linear or non-linear models based on whether optical characteristics of camera is taken into consideration or not.

The simple pin-hole camera model assumes that the imaging process is a perfect perspective projection from world to image coordinate frames. However, real cameras are not perfect perspective projections (especially, when used with a short focal length lens) and non-linear distortions are introduced into the imaging process. There are several different forms of non-linear distortion, but usually only radial distortion is modeled [28]. In radial distortion, the error is a radial displacement proportional to an even power of the distance from the center of the image. Several methods have been suggested which estimate and correct for radial distortion. Where accuracy that really matters like for 3D motion estimation and reconstruction problems perspective projection model with a lens distortion model is the best solution. But in [29], it is reported that for CCD camera non-linear distortion is very negligible (less than 0.01 pixels).

Another concern with the pin-hole model is if pinhole is too big, many directions are averaged leading to the blurring of the image. On the other hand, if pinhole is too small, diffraction effects blurring the image. Generally, pinhole cameras are dark, because a very small set of rays from a particular point hits the screen.

Another classification of camera model is given by [30] dividing camera models with either orthographic or perspective projection (e.g. pinhole). The first model assumes the objects in 3D space to be orthogonally projected on the image plane. Here the projectors are parallel to the co-ordinate axes of the object. Since it is a linear form of mapping, it is simpler and computationally less expensive. It is suitable for vision applications where the geometric accuracy are somewhat low.

2.3.2 Description of a Pin-hole Camera Model with Perspective Projection

A pinhole camera model developed in [1] is shown in Fig. 2.1. A right-handed co-ordinate system having the center of projection at the origin is used and the image plane is considered to be at a distance of (focal length) away from it.

Let a camera is placed at the center of the world reference point. Then, a point $\mathrm{P}(X, Y, Z)$ in 3D space maps to the 2D camera plane as,

$$x_{cam} = \frac{f}{Z}X \tag{2.1}$$

$$y_{cam} = \frac{f}{Z}Y \tag{2.2}$$

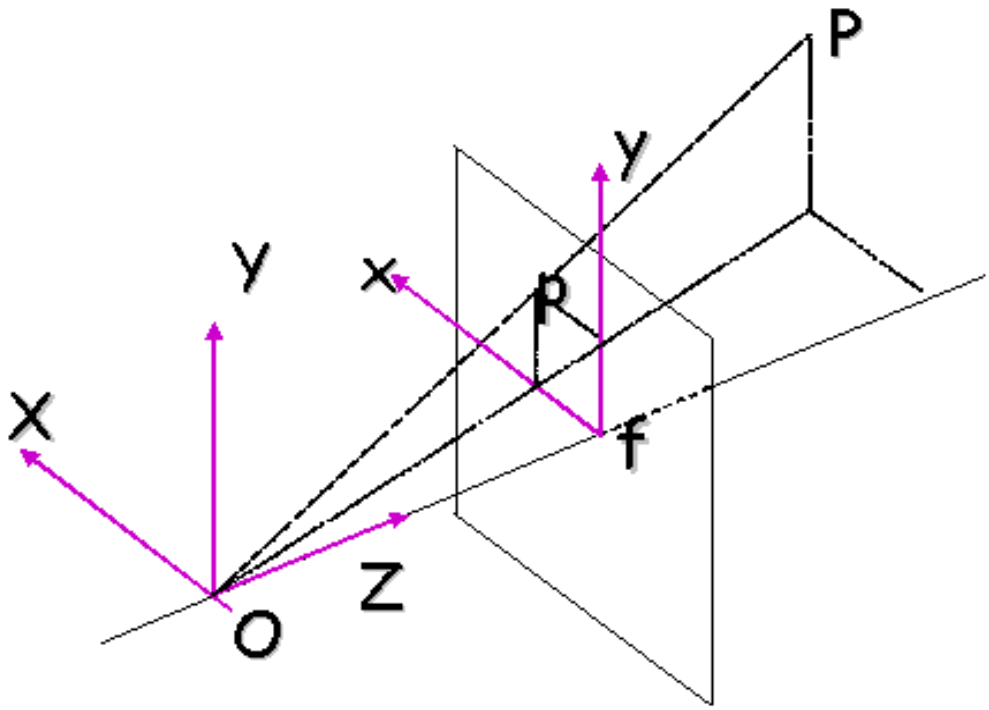

Figure 2.1: Pinhole camera model [1].

where, Z is the depth of the point.

In matrix form, this can be written as:

$$\begin{bmatrix} x_{cam} \\ y_{cam} \end{bmatrix} = \begin{bmatrix} f\frac{X}{Z} \\ f\frac{Y}{Z} \end{bmatrix} \tag{2.3}$$

Using homogeneous co-ordinates, the pinhole projection in Equation 2.3, can be expressed as:

$$\begin{bmatrix} x'_{cam} \\ y'_{cam} \\ z'_{cam} \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.4}$$

where,

$$x_{cam} = \frac{x'_{cam}}{z'_{cam}}$$
$$y_{cam} = \frac{y'_{cam}}{z'_{cam}}$$

Simplifying Equation 2.4, we get:

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ f \end{bmatrix} = f/Z \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.5}$$

where, the term f/Z is is the scale factor.

Now, the relationship between world co-ordinate frame and camera reference frame can be expressed as:

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ 1 \end{bmatrix} = \begin{bmatrix} f/Z & 0 & 0 & 0 \\ 0 & f/Z & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.6}$$

Or,

$$p_{cam} = K * S * P \tag{2.7}$$

where, K is the matrix that projects the points from camera coordinates to image plane using a scale factor of f/Z. And, S is a transformation matrix from world to camera coordinates with $t_x,\ t_y$ and t_z forming the translation vector and r_{xx} as the elements of a rotation matrix.

Equation 2.6 can be further compacted as

$$\begin{bmatrix} x_{cam} \\ y_{cam} \end{bmatrix} = f/Z * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.8}$$

Now, applying weak perspective projection; see [1, 16] for details, we can express the above equation as:

$$\begin{bmatrix} x_{cam} \\ y_{cam} \end{bmatrix} = f/Z_0 * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.9}$$

where, Z_0 is average distance of camera from the objects and if δZ is the depth of scene, $\delta Z << Z_0$.

Finally, the relationship between image plane coordinates (x_{cam}, y_{cam}) and their pixel addresses (u, v) can be expressed using affine transformation [31] and homogeneous form:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.10}$$

where, m_{ij} are the elements of projection matrix, M.

Describing origin offset separately the projection can be expressed as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \tag{2.11}$$

where, (u_0, v_0) is the origin of the pixel array.

For a stereo-vision system, two projection matrices, one for each left and right images are to be computed.

2.3.3 Classification of Camera Calibration Methods

According to [32], we can classify camera calibration methods as below:

- 3D reference object-based calibration: Calibration is performed by observing a calibration object whose geometry in 3D space is known with very good

precision. Calibration can be done very efficiently. The calibration object usually consists of two or three planes orthogonal to each other. Sometimes, a plane undergoing a precisely known translation is also used. These approaches require an expensive calibration apparatus, and an elaborate setup.

A frame of reference with four fiducial points is used in [1, 33] as calibration object. But most classical camera calibration methods typically use a precise grid of points to solve for the intrinsic and extrinsic camera parameters [2, 34]. Multiple marker points (3-8) can be used on the grid. But results are not unique for less than 6 points [2]. Since in most cases a calibration grid cannot be inserted into the scene and one must rely on existing objects for the calibration. [2, 35, 29] directly uses a robot arm as calibration fixture.

- Self-calibration: Instead of using any calibration object, in this technique a camera is moved in a static scene, the rigidity of the scene provides in general two constraints on the cameras' internal parameters from one camera displacement by using image information alone. Therefore, if images are taken by the same camera with fixed internal parameters, correspondences between three images are sufficient to recover both the internal and external parameters which allow us to reconstruct 3D structure up to a similarity. While this approach is very flexible, it is not yet mature. Because there are many parameters to estimate, we cannot always obtain reliable results.

In [10], a self-calibration technique for camera in a monitor-based AR display used in ARToolKit (a software used for AR application) is proposed. It overcomes the need for human intervention in the existing ARToolKit. But it is not as robust as the existing one and its accuracy can be improved.

Camera parameters are found in a 3×3 calibration matrix and used to calculate the transformation matrix to make sure that the virtual object is overlaid accurately.

Pinhole model is chosen for camera model with assumption that image axes are orthogonal. Weak perspective projection is used for transform from 3D camera co-ordinates to image co-ordinates and affine transformation is used for image co-ordinates to pixel surface co-ordinates. Correction for radial distortion of lens is also made. A publicly available software *image matching* [36] is used to get the accurate determination of point correspondence and to estimate the fundamental matrix (epipolar geometry). The algebraic Dornaika's method [10, 37] is used for self-calibration that is derived from this fundamental matrix. By experiment, it is shown that the principal point estimation for self-calibration without distortion correction is quite reliable and taking distortion into account improves the accuracy of the camera parameters.

- Other techniques: Vanishing points for orthogonal directions and calibration from pure rotation.

In [10, 38], camera calibration techniques are categorized into situations with known scene (using a planar pattern) and unknown scene (using camera motion) and self-calibration is defined for the situation where neither the scene nor the camera motion is known.

In [11], calibration methods are divided into implicit and explicit types.The methods where the camera model is based on physical parameters, like focal length and principal point, are called explicit methods. In most cases, the values for these parameters are in themselves useless, because only the relationship between 3D reference coordinates and 2D image coordinates is required. In implicit camera calibration, the physical parameters are replaced by a set of non-physical implicit parameters that are used to interpolate between some known tie-points.

2.4 Generation of 3D Computer Graphics

The process of generating 3D computer graphics can be sequentially divided into three basic phases: Modeling, Scene layout setup and Rendering. These are discussed in detail in [39] and summarized in the following sections.

2.4.1 Modeling

The modeling stage could be described as shaping individual objects that are later used in the scene. Modeling processes may include editing object surface or material

properties (e.g., color, luminosity, diffuse and specular shading components-more commonly called roughness and shininess, reflection characteristics, transparency or opacity, or index of refraction), adding textures, bump-maps and other features.

Objects may be fitted with a skeleton, a central framework of an object with the capability of affecting the shape or movements of that object. This aids in the process of animation, in that the movement of the skeleton will automatically affect the corresponding portions of the model. Sometimes, objects are broken down from abstract representations called *primitives* such as spheres, cones etc, to so-called meshes, which are nets of interconnected triangles. Meshes of triangles (instead of e.g. squares) are popular as they have proven to be easy to render using scanline rendering.

Some of the vastly used modeling techniques are:

1. Constructive Solid Geometry: It allows a modeler to create a complex surface or object by using Boolean operators to combine objects.

2. Non uniform rational B-spline (NURBS): It is a mathematical model commonly used in computer graphics for generating and representing curves and surfaces.

3. Polygonal modeling: A polygon is a closed planar path composed of a finite number of sequential line segments. The straight line segments that make up the polygon are called its sides or edges and the points where the sides meet

are the polygon's vertices. If a polygon is simple, then its sides (and vertices) constitute the boundary of a polygonal region.

4. Subdivision Surfaces: In computer graphics, subdivision surfaces are used to create smooth surfaces out of arbitrary meshes. Subdivision surfaces are defined as the limit of an infinite refinement process.

Modeling can be performed by means of a dedicated program (e.g., Lightwave Modeler, Rhinoceros 3D, Moray), an application component (Shaper, Lofter in 3D Studio) or some scene description language (as in POV-Ray).

2.4.2 Scene layout setup

Scene setup involves arranging virtual objects, lights, cameras and other entities on a scene which will later be used to produce a still image or an animation.

For animation of complex object *keyframing* technique is sometimes used where instead of having to fix an object's position, rotation, or scaling for each frame in an animation, one needs only to set up some key frames between which states in every frame are interpolated.

Lighting is an important aspect of scene setup to give the aesthetic and visual quality of the output. However, these effects can contribute greatly to the mood and emotional response effected by a scene in case of photography but not much important in telerobotics.

2.4.3 Rendering

Rendering is the final process of creating the actual 2D image or animation from the prepared scene with mathematical models.

Rendering for interactive media, such as games and simulations, is calculated and displayed in real time, at rates of approximately 20 to 120 frames per second. Animations for non-interactive media, such as video and film, are rendered much more slowly. Non-real time rendering enables the leveraging of limited processing power in order to obtain higher image quality. Rendering times for individual frames may vary from a few seconds to an hour or more for complex scenes.

Rendering software may simulate such visual effects as lens flares, depth of field or motion blur to lend an element of realism to a scene. Techniques have been also developed for the purpose of simulating other naturally-occurring effects, such as rain, smoke, fire, fog, dust etc. Simulation of light focusing by uneven light-refracting surfaces, such as the light ripples seen on the bottom of a swimming pool) and the light reflecting inside the volumes of solid objects such as human skin. But such effects are not important in our telerobotic application.

2.5 Registration Techniques

The mostly used approach for AR registration is model-based approach. In this approach, shape of the target is modeled in the form of a computer-aided design

(CAD) model and superimposed on the real scene. This is closely related to classical camera calibration problem.

The slave robot arm picture is drawn on a real or computer generated background image at the master stations display unit. For this a complete and accurate model of the slave robot arm is used at the master station. The slave station is supposed to send position and orientation parameters of the slave robot arm to the master station in a continuous manner. Based on these received parameters, the master station can draw an artificial image (graphically computed) of the master robot arm based on the available model.

In [2], a model-based registration and a model-based prediction oriented tracking system using Extended Kalman Filter (EKF) are approached for AR in telerobotic application.

As shown in Fig. 2.2, taking a gross estimate of the camera focal length, a geometric model of the robot, its configuration and one image of the scene initial registration or static registration is performed. It improves the evaluation of the focal length, estimates the pose of robot's base frame relative to the camera and estimates the first order radial distortion coefficient.

For feature detection and matching, a semi-automatic approach is adopted. The HO aligns the virtual model with the real one on screen using an AR-based interface. The robot or manipulator is marked with round colored stickers. The visible virtual

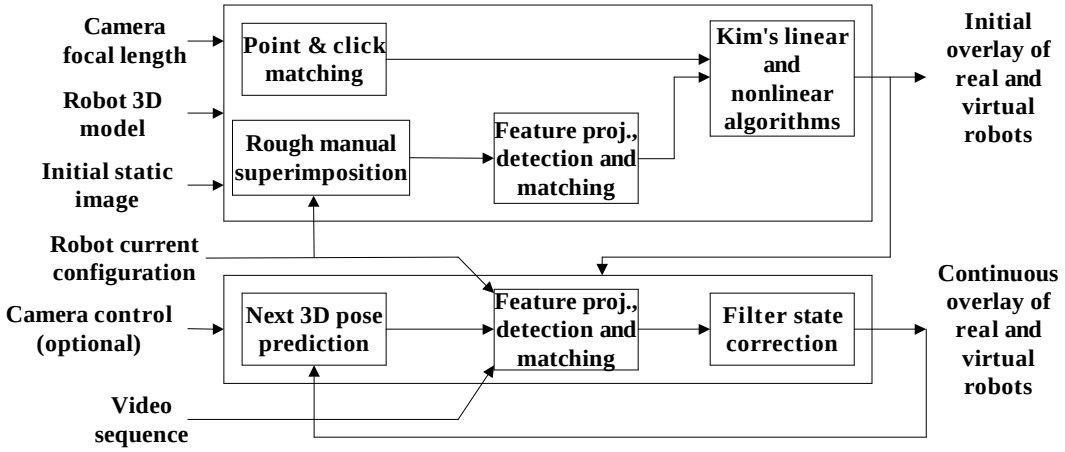

Figure 2.2: Model-based registration and tracking system [2].

features are determined and projected onto the image plane. Then, the real features around the image projections of the virtual ones are searched and matched by using color segmentation. Following this matching process, the camera's intrinsic and extrinsic parameters are determined. At first, Kim's linear algorithm is used to compute the approximate pose parameters using only nominal values of the camera's focal length. Then, taking this as the initial state an iterative non-linear algorithm is used to compute directly the roll, pitch & yaw angles and a more accurate value of the focal length and the radial distortion co-efficient.

An algorithm is proposed to update the computation of camera parameters through filter-based tracking whenever the real camera and/or the real robot move. For this, two processes are operated continuously: (1) feature tracking and detection process and (2) computation of new values for the camera's extrinsic parameters using the image locations of the detected marks. EKF prediction phase is used to

predict the location in the image of visible markers and the image co-ordinates of the markers are then directly used in the correction phase, where a pose is computed for each frame. The Kalman filer is used only to smooth the values computed over time.

Simulation result obtained in [2] shows that the manipulator and the camera both follow trajectories generated using an arbitrary control law where the correct pose is very closely followed to the point where it is difficult to distinguish the estimator from the real state on the graphs.

Although the EKF's correction phase performs very well, it can do very little if the prediction is so poor that the markers cannot be located in the image. Thus, most visible markers remain detectable based on the position predicted by the EKF.

The ARGOS (Augmented Reality through Graphic Overlays on Stereo-video) [40] project also uses model-based approach. Detail of this project will be given in Section 2.6.3.

2.6 Stereovision and AR Approaches in Telerobotics

In this section, we will discuss some of the important approaches made so far towards the integration of stereovision and augmented reality with telerobotic applications.

2.6.1 SMART Approach

In [41], an augmented reality interface for Telerobotic application in remote unstructured environment is proposed. It allows on-demand generation of virtual workspace views, 3D embedding of virtual replica of the equipment into workspace views and haptic feedback to the operator.

The system architecture consists of data acquisition, interactive perception and enhanced task representation modules. Camera calibration and image acquisition is performed in the first module using 3 fixed cameras. The interactive perception module finds sparse correspondences among the three input views, determines current viewpoint relative to the three fixed cameras using head-tracker device and generates the virtual view for the requested viewpoint using the set of matched feature points among the three views. The third module performs 3D interposition of graphical replica of the equipment into synthesized workspace views and provides haptic feedback using 6 DOF haptic manipulator.

The matching algorithms used here treats all images identically and does not assume any a priori knowledge of the scene while most of the conventional matching algorithms use two of the input images to search for matches and the third to validate potential matches. The proposed algorithm first detects the features, matches them and then removes the inconsistency. The Head Mounted Tracker (HMT) device tracks the movement of the operator's viewpoint. Whenever the operator moves

his/her head, a new viewpoint is computed and passed to view synthesis module.

In view synthesis module, parameter expressing the relationship between the original viewpoint and the requested viewpoint are calculated. Then, the triangulation of visible regions is computed based on the matches obtained from the matching stage and a validity check is made to detect and remove possible mismatches and invalid triangle pairs. New views are generated by projecting the matched feature points onto the new image plane, computing the new local texture and then determining the rendering order.

The interposition sub-module of the Enhanced task representation is based on the Point-based interposition algorithm described in [42] using sparse matching.

The interface is tested in a drill operation to specify a number of drilling locations. A drill bit like a cylindrical cone with 24 vertices is used as a graphical object in the scene of rocks. The visibility changes are shown when the tool penetrates into the scene surface.

2.6.2 The MDTF Approach

In the work of [1], a Multi-threaded Distributed Telerobotic Framework with AR functionality is developed. It can enable the operator to insert a small ball into the most recent video scene at the gripper position of PUMA-590 robot in the absence of fresh video data. The gripper position is calculated locally from the command

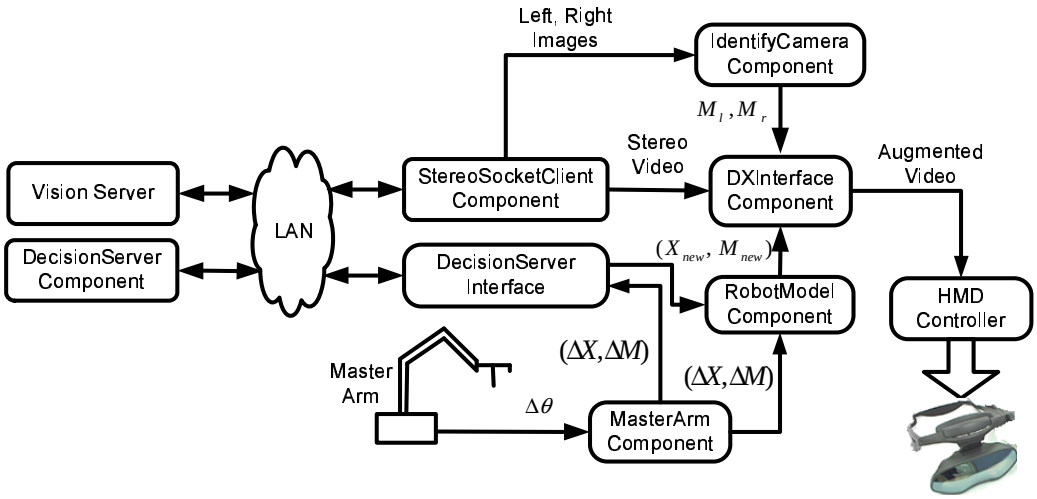

Figure 2.3: Software architecture of the complete AR system on client side [1].

data coming from the master arm of the robot using a direct geometric model of the robot. To perform the task, a stereo video client-server framework is developed over a LAN where robot arm is connected to the server and master arm is connected to client machine so that the operator can control the robot from a distant place. The server interface is developed using MS Visual C++ and MS Visual C#.NET programming languages while the client GUI is developed using MS Visual C#.NET and MS DirectX. The architectural view of the complete AR system proposed in [1] is shown in the Fig. 2.3.

The server continuously captures stereo images from two cameras using FireWire, PCI card and Sample Grabber component of DirectShow interface of Microsoft DirectX and sends the stereo frame through window socket upon client's request. On the client side after receiving the video data from network using synchronous win-

dows sockets, Graphics Device Interface (GDI) functions are used to show the pictures on the monitor. The software architecture of the AR system on the client side for connecting with server and displaying 3D stereo image using HMD is shown in the Fig. 2.3. Input from the user is taken through the Master Arm component that provides the incremental position vector and orientation matrix to DecisionServer interface and RobotModel component. DecisionServer interface executes the incremental move command on remote DecisionServer and RobotModel provides the new 3D position of gripper to DXInterface component. In the mean time, DXInterface also acquires a stereo frame of remote scene through StereoSocketClient component as well as left and right projection matrices from the IdentifyCamera component. Then, DXInterface projects a virtual ball at the gripper position in 2D stereo image and sends the stereo image to HMD controller in order to display it to user. The local model is updated as IDecisionServer sends the current angular position of the robot upon reception of the ONMove event from the remote side.

The projection matrices are calculated in this system assuming a pinhole camera model with weak perspective projection. The relationship between image plane coordinates and their pixel addresses is modeled by affine transformation. Using the graphical interface shown in Fig. 2.4 user can select the four non-coplanar points of affine reference on both left and right images. An analytical solution of the system is developed using Faugeras's linear method [43] and evaluated by just substituting

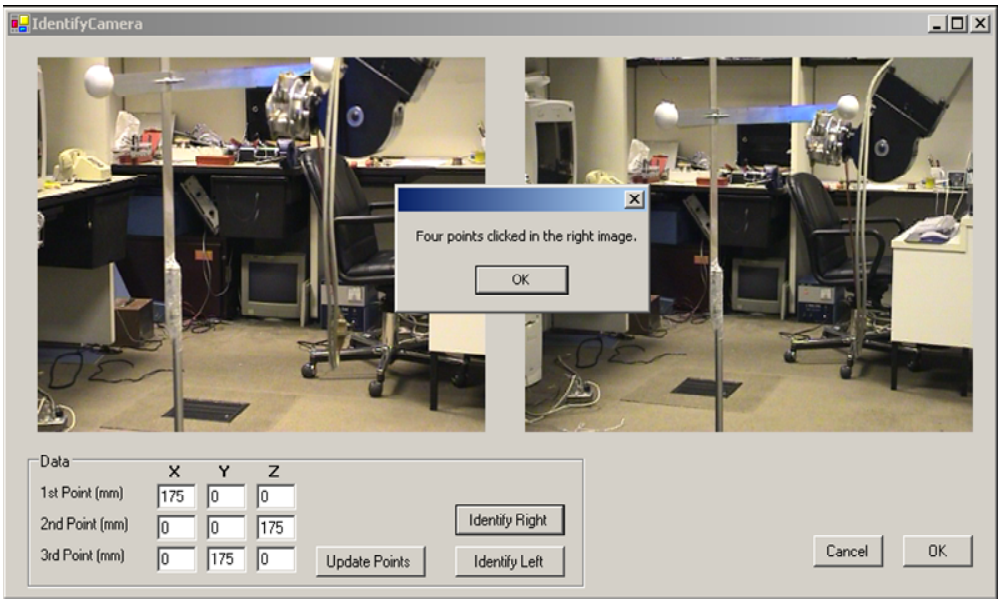

Figure 2.4: Camera identification GUI [1].

the values recorded by the user. This approach requires a physical reference frame to be setup at the server side during the identification phase and any point in 3D scene is described with respect to that frame [44]. Page flipping is used for stereo visualization.

It is a true multi-stream distributed framework. As the identification of cameras and other projection related data across different runs are preserved in the permanent memory, this approach requires the identification only when the cameras or the objects have been moved from their previous locations. In this system, stereo video is updated in a page-by-page format instead of pixel-by-pixel and thus time delays are reduced. It also implements a single buffer with serialized transfer and double buffer with de-Serialized transfer approaches. It is shown that the later approach

enables to send higher number of stereo frames over the same LAN and hardware. The only overhead is the allocation of extra buffer in the server DRAM which is not a real problem with available systems containing large memory.

The system does not deal with complex geometrical shapes, only a small ball is inserted in the real video at the Robot gripper position. But in most other works, like [5, 45] whole robot is modeled and superimposed on the real image of the environment objects. Again, in this work only the position matrix is considered while computing the gripper location ignoring the orientation matrix representing the orientation of the gripper. The accuracy of the augmented ball at the gripper position is also dependant on the position of cameras from the robot gripper and the distance between the reference frame and robot itself. Accuracy is increased with increase distance between cameras and robot. Moreover, no experiment is done to evaluate the performance of the AR system.

2.6.3 ARTEMIS Proposal

In [5, 3], an AR-based telerobotic interface named ARTEMIS (Augmented Reality TEleManipulation Interface System) is proposed. It is based on ARGOS (Augmented Reality through Graphic Overlays on Stereovideo) [40] that works on local machine only. The schematic diagram of ARTEMIS system is shown in Fig. 2.5.

In this approach, a graphical model of robot is overlaid on real stereo image for

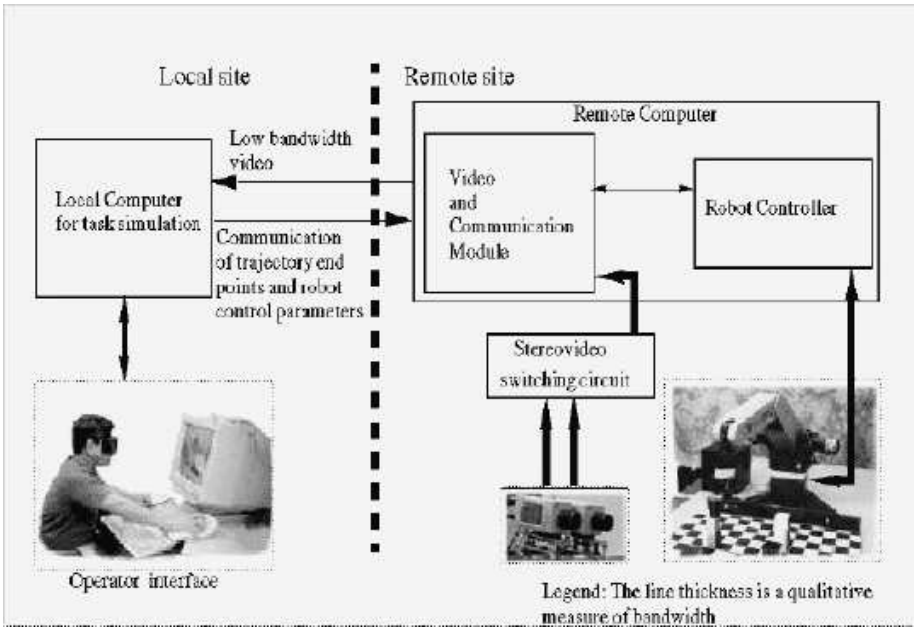

Figure 2.5: Schematic diagram of ARTEMIS [3].

local teleoperation task simulation. The graphical interface allows the operator to take the end effector of the graphical model of the robot to a succession of new locations within the task space and mark each of these as a trajectory endpoint. The final sequence of trajectory endpoint commands defines a complete path which is then used for execution of the robot at remote site.

In their calibration and registration process, first the graphic software is calibrated with the video cameras at the remote site. It ensures the graphic images are drawn with the identical perspective projection parameters as those which determine the video camera images. Since the graphical models are rendered in a separate graphical coordinate system, they have also matched these with the world

coordinate system of the remote site. Thus, all six degrees of freedom of the origins and the axes of both coordinate systems overlap on the resulting mixed display of video and graphics. It is ensured that the base link of the graphical robot model coincided exactly with the base link of the actual telerobot observed in the stereo image. By setting the joint angles of the graphical model equal to those of the real robot, the entire model of the robot was overlapped on the stereo image of the real robot. Then, by changing the joint angles of the kinematically modeled robot, its stereo graphic image was manipulated relative to the under laid stereo image, while the base links of both robot images remained fixed with respect to each other. An example of a pick-and-place task simulation was also illustrated. The graphic model of the robot was rendered with transparent wire-frame polygons.

It is assumed that the remote task space remains relatively unchanged within the duration of a particular trajectory sequence. This system provides benefits of reducing programming errors by making simulation locally with higher levels of control and thus become invariant to time-delay and requiring less communication bandwidth. The display implemented in this system showed only an exocentric view of the manipulator, because the video cameras were placed external to the robot.

The camera calibration of the system do not account for nonlinear distortions in the video images, such as radial distortions. It also assumes a constant coordinate transformation matrix between the stereocameras and the base link of the robot

during the task. Hence, the system will not work if any movement of the telerobot base occurs during task execution. A 5 dof telerobot is used in this implementation limiting some gripper orientations in the task space. The effective frame rate reported is also low: about 6.5 to 7 frames/second on the average.

2.6.4 The UJI Online Robot Project

In [45], a high level web-based AR as well as VR user interface is developed using Java, Java3D and CORBA. It allows the manipulation of objects over a board by means of mouse interactions on the 3D virtual reality environment. The AR environment is provided by showing for example the position of gripper over the board or by adding object recognition information to the stereo camera input. It can specify the voice commands to the robot in a natural way for example, pick up the scissors.

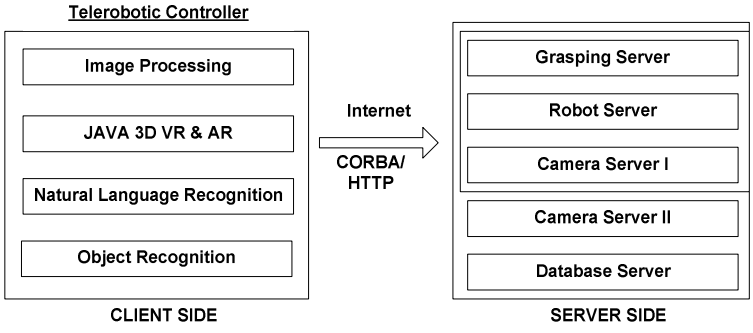

Figure 2.6: System architecture of the web-based telerobotic system.

As shown in Fig. 2.6, the main component of the client side is the telerobotic controller which is divided into four sub-modules. The image processing sub-module

takes care of capturing and segmentation of images. The second one, of which we are interested in, implements the 3D virtual environment. It allows the VR interaction with the robot as well as the AR feature. It shows graphical representation of the position of the gripper over the board as well as the superimposition of objects information to the camera images. The third sub-module interprets and simplifies natural language command from the user to the robot. The object recognition sub-module processes the camera images and returns every objects name. On the other end, the server side consists of several concurrent processes running on the server machine and interacting through the CORBA and HTTP standards. The robot server module of the server side accepts CORBA request to move the real robot to a given world position (x, y, z) managing directly the values for the joints and controlling the opening of the gripper. The grasping server module calculates the set of grasping points for every object present into the scene that can be used in order to manipulate an object according to the stability requirements. The camera server module consists of commercial *WebCam32* and offers HTTP interface to the server cameras. There is also a database server module that stores a mathematical description of every object already learned by the system.

The human-robot interaction using mouse and natural language commands made the system very user-friendly. It is also suitable for web-based teleoperation as it consumes very little bandwidth. The performance results show that program

launching and object recognition are the most time consuming operations in this system. Its VR facility can engage users in some interesting work while another user is having control over the real robot.

2.6.5 Discussion and Comparison of the Systems

The basic features used in the cited approaches are tabulated and compared as shown in the Table 2.1.

2.7 Summary of the Findings

In all the examples cited, regardless of whether the system uses a Telepresence, AR or VR technique, the following conditions were observed:

- Time Delay: When the time delay grows much larger than the dynamic time scale of the remote mechanism, the telepresence method is inefficient because of the delay between the sensing and the actuation. This delay makes the control system unstable.

- Communications Bandwidth: A related problem to that of time delay is the communications bandwidth from the human controller to the remote mechanism. When a close loop control is difficult or impossible, a move-and-wait strategy or local decision making capability is incorporated into the system.

Table 2.1: Comparison of the cited AR telerobotic system.

Features	SMART	MDTF	ARTEMIS	UJI
Manipulation environment	Unstructured	Structured	Unstructured	Structured
Client-Server comm.	Not provided	LAN	RS 232C lines, also Internet	Internet
Calibration method	Self-calibration	3D ref. object-based, linear	3D ref. object-based, linear	Self-calibration
Calibration object	N/A	Ref. frame with 4 points	Grid plane with 14 points	N/A
No. of camera	3	2	2	3
Comm. framework	Not specified	MS .NET	Not specified	JAVA and CORBA
Graphics software	Not provided	DirectX and WindowsGDI	SGI Inventor*TM* 1.0, the SGI gl*TM* graphics library and X-Windows*TM*	JAVA3D
3D visualization	Monitor, HMT,3D reconst.	Monitor, HMD	Monitor, shutter glasses	Monitor, 3D reconst.
Input Device	6-DOF haptic device	Master arm, keyboard, mouse	Spaceball, keyboard	Mouse, Keyboard

The operator interface should be usable and efficient when operating over low bandwidth communication channels.

- Efficient Command Cycles: A teleoperator should not require to send a dozen of commands to the remote system in order to execute a single operation. The operator's access to high and low level commands may provide maximum flexibility in sending efficient command sequences.

- Situational Awareness: The operator interface should lead to greater situational awareness than can be accomplished with a traditional control approach.

- Modular Components: Modern large software and hardware systems can only be efficiently maintained and extended if the modular components are written in a modular fashion. This allows the system to be designed more easily. It also facilitates future modification, re-use and ease of support.

- Use of Commercial Products: Wherever possible, leveraging from commercially available tools and products is a cost effective approach to design and build the system.

- GUI with Suitable Control Features: GUI should provide some visible navigation aids and attractive data presentation for the novice user to drive the system better.

- Virtual-Reality Synchronization: To provide a meaningful interaction virtual objects should be properly synchronized with the real world information. The accuracy of matching and refresh rate of graphics rendering is required.

Chapter 3

Design

In this chapter, we will develop the conceptual model of our telerobotic stereo-vision system with AR. At first, the overall system design is described with diagram to show the additional components to be needed to augment the telerobotic stereo-vision system chosen. Then, the design approach chosen will be described. Then, we will describe the mathematical model of the robot in use. The description of all the concepts and algorithms used for designing graphic robot simulator will follow. Then, the design of the interface to the client-server communication module and the technique of camera calibration and image registration will be discussed. The design of GUI and its use for graphical tele-manipulation will be described at the end of the chapter.

3.1 System Architecture

We have chosen the telerobotic stereo-vision system developed by Iqbal, A. [1] since it provides higher video transfer rate (around 17 fps). The system is shown in Fig. 3.1. To augment this with graphical overlays, we need to add some more components as shown in Fig. 3.2.

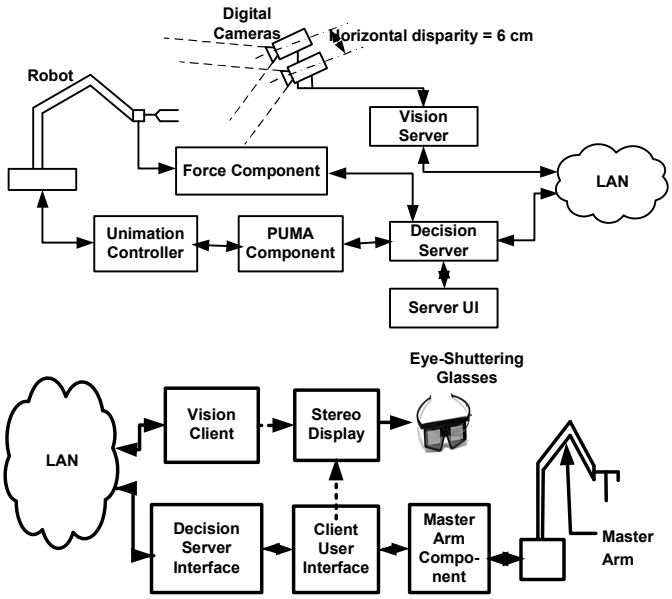

Figure 3.1: Telerobotic stereo-vision system to be augmented [1].

There should be a virtual object module to model the graphic object, a camera calibration module to update the projection parameters and modification to gripper and robot controller module to get position updates and on the top of all a GUI to facilitate user's interaction with the system. Details of the modules will be discussed in subsequent sections.

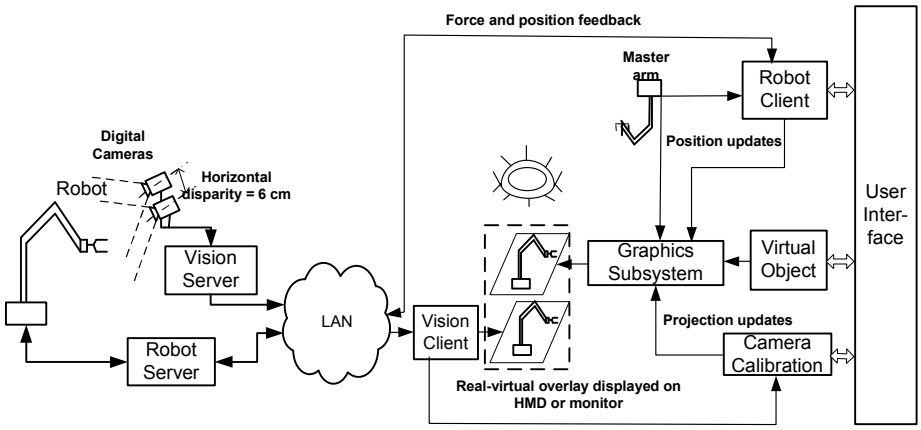

Figure 3.2: Overall AR system design.

3.2 Methodology

The design methodology of our augmentation of the telerobotic stereo vision system is discussed as follow and shown in Fig. 3.3.

- Developing the Mathematical Model of the Robot: A set of geometric equations are developed to build the mathematical model of the robot.

- Drawing Graphic Robot Arm and Other Graphical Objects: Having chosen the graphics software that will allow exchange of data into and out of the model, the next step is to build the graphic robot arm model and other graphic objects.

- Animating the Graphical Objects for Simulating Task: Having constructed the graphic model, codes are then inserted to enable manipulating the graphics with the interface by handshaking and exchanging data to and from the model

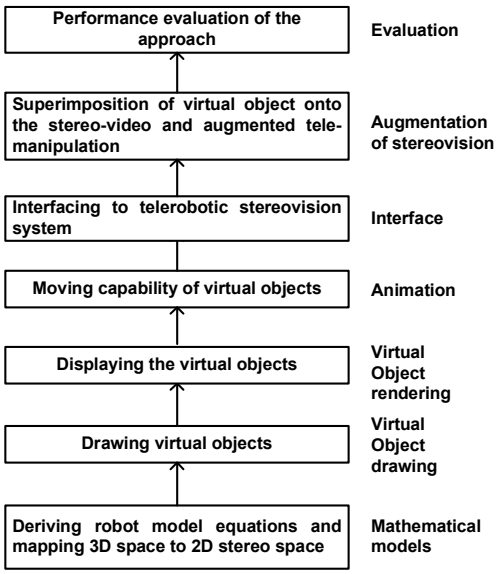

Figure 3.3: Hierarchy of design abstraction.

and interface. Algorithms are developed for the movements.

- Interfacing to the Telerobotic Stereovision System: Graphics sub-system is interfaced with the stereo-vision system.

- Identification of Camera Calibration Technique and Superimposition of Graphic Model into the Real Image: Appropriate camera calibration and image registration techniques are determined to perfectly superimpose the graphic model into real video image captured in server PC and received in client PC through network.

- Augmented Tele-manipulation: Once the methodology of data transfer is stabilized, tapping of all necessary data is implemented to perform the AR tele-

robotic manipulation.

- Evaluating the Design: Lastly, the full integration is tested.

3.3 Mathematical Model of the Robot Manipulator

An industrial robot is a multi-function manipulator that can be modeled as an open chain of rigid bodies, called *links*, connected in series by kinematic joints. The function of the joint is to control the motion between the links. The first link is attached to the supporting base by the first joint, and the last link contains the end effector or other type of manipulator device. Each *joint-link* pair constitutes one degree of freedom. An n degree of freedom manipulator contains n joints, or in more general terms, n link-attached coordinate system. The joints and links are numbered starting from the base. The lowest joint is fixed to the reference coordinate system, while the highest joint is fixed to the local coordinate system of the end effector. Robotic joints can be categorized as either *revolute* or *prismatic* joints as shown in Fig. 3.4-(a). A revolute joint allows link L_{i+1} to rotate with respect to the previous link L_i. A rotation angle θ_{i+1} can be used to define the angular position of L_{i+1} relative to L_i. This is shown on Fig. 3.4-(b) and (c). A prismatic joint allows a link to translate with respect to the previous link. A translation variable θ_{i+1} can also

be used to define the linear position of L_{i+1} relative to L_i.

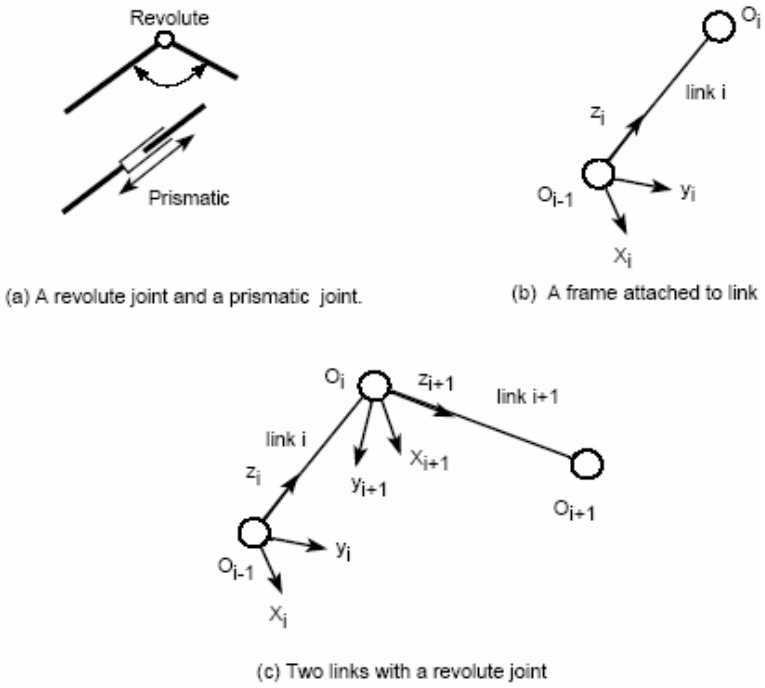

Figure 3.4: Joint types and frame of reference.

The relative position and movement of the individual links with respect to their preceding links provide a description of an entire articulated structure in the operating space and formulate a mathematical model of a kinematic chain of the robotic system. Since robotic manipulation can be achieved only by maneuvering the arm linkages in the task's environmental space, robot kinematics is an important tool in work space design, trajectory planning and motion rate control. Kinematics is concerned with the analytical description of spatial position, orientation, displacement, velocity and acceleration. There are two fundamental problems in studying robotic kinematics: the direct kinematic and the inverse kinematic problem. The direct

kinematic problem involves the determination of the position and orientation of the end effector with respect to the reference coordinate system, given the joint variables of the robot arm. The inverse kinematic problem, on the other hand, involves the determination of the joint controlled variables, given the position and orientation of the end effector.

As we have modeled and carried out experiments with the PUMA-560 robot arm we will first give a short description of the it and then we will derive its kinematic models.

3.3.1 The PUMA-560 Manipulator Arm

PUMA 560 robot is an industrial robot with six degree of freedom i.e. there are six independent variables (or co-ordinates) to completely specify the configuration of its mechanics. There are six links in this robot connected as a serial chain i.e. each of the links, excepted the first and the last links, is connected to two other links. All the six joints of PUMA-560 robot arm are rotational joints. The last three have concurrent rotation axes, which simplify their geometric and kinematic models. All the joints are driven and controlled by DC-Servomotors. The servomotors are equipped with electromechanical brakes that can lock the arm in a fixed position. The brakes are released by the controller when the arm power is on. The components of the robot arm are the Trunk or Base, Shoulder, Upper Arm (the inner link), Fore-arm

(the outer link), Wrist and Gripper as shown in Fig. 3.5. Functionally this arm can be divided into two parts which are the *Transporter* and the *Effector* parts. The *Transporter* is responsible for transferring and positioning the effector which include the grasping system and the work piece. On the other hand, the *Effector* is responsible for the orientation of the arm. The *Transporter* includes three links: the shoulder, elbow and forearm. The later part includes the pitch, yaw and roll.

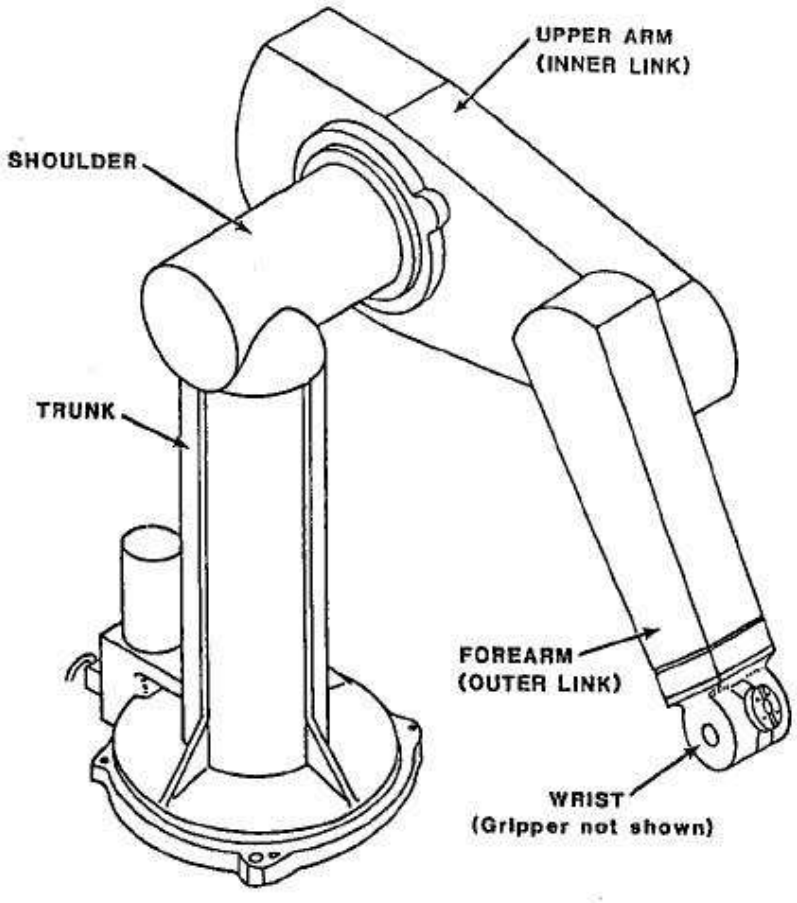

Figure 3.5: Schematic diagram of PUMA 560 robot manipulator.

3.3.2 Motion Coordination and Selection of Cartesian Frames

The problem is to determine the position and orientation of objects in the 3-dimensional space. The objects are the links of the manipulator, and the tools with which it deals.

In geometry, objects (in our case links of the manipulator and the tools with which it deals) are described by just two attributes: their position and their orientation. In order to describe these two attributes, a frame of reference is attached to each link. The frame of reference is defined using three orthogonal vectors $\{X, Y, Z\}$. Fig. 3.4-(b) gives an example of frame R_1 translation and rotation relative to frame R_0.

The position of the manipulator is generally described by giving a description of the tool frame, which is attached to the end effector, relative to the base frame which is attached to the fixed base of the manipulator as shown in Fig. 3.6.

In the 3D space, there are six DOF; three position parameters: X, Y, Z and three angular orientation parameters specifying the orientation of the gripper in the space. We specify both position and orientation using a 3×1 translation vector and a 3×3 rotation (orientation) matrix. These notations are opposed to the Denavit-Hartenberg (DH) notations which are used in the majority of cases and represented by a 4×4 homogeneous matrix that transforms a vector from one coordinate system to another.

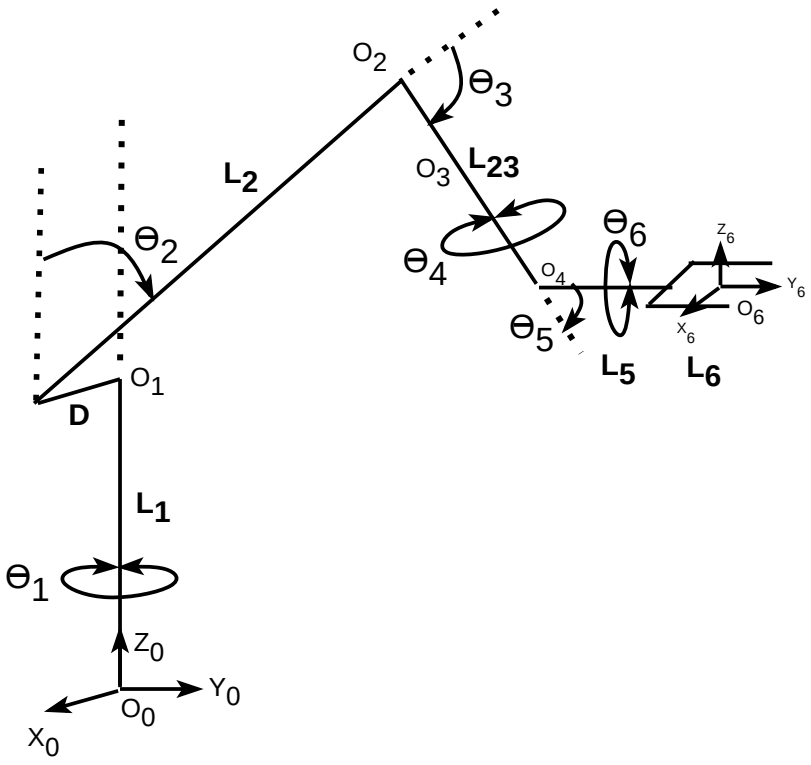

Figure 3.6: The kinematic model of the PUMA 560 robot arm.

A Cartesian coordinate system is defined in the 3D space, by introducing three orthogonal vectors X, Y, Z. A frame can be defined by the orthogonal vectors with origin O. We say that the link L_{i+1} revolute with respect to link L_i when frame R_{i+1} rotates relative to either axes X_i, Y_i or Z_i as shown in Fig. 3.4-(c). The end point O_{i+1} of L_{i+1} can be associated with a vector O_iO_{i+1} which will be denoted by $O_iO_{i+1,i}$ to indicate that the vector is observed in frame R_i. $M_i^{i+1} = [X_{i+1,i}, Y_{i+1,i}, Z_{i+1,i}]$ is the transfer matrix from frame R_{i+1} to R_i, which represents the rotation between links L_i and L_{i+1}. Therefore, the link vector $O_iO_{i+1,i}$ can be expressed as follows:

$$O_iO_{i+1,i} = M_i^{i+1}.O_iO_{i+1,i+1} \tag{3.1}$$

where, $O_iO_{i+1,i+1}$ denote the vector O_iO_{i+1} observed in frame R_{i+1}. Note that the vector $O_iO_{i+1,i+1}$ has a simple expression because link L_{i+1} is parallel to axis Z_{i+1}. Therefore, the position vector $O_0O_{n,0}$ can be decomposed as the sum of the link vectors:

$$O_0O_{n,0} = \sum_{i=1}^{n} M_0^i . O_{i-1}O_{i,i} \tag{3.2}$$

vector $O_{i-1}O_{i,i}$ has a simple expression because it is represented with respect to its own frame of reference R_i. Both the vector $O_0O_{i,0}$ and $\sum_{i=1}^{n} M_0^i$ can be expressed in a recursive form, leading to:

$$M_0^i = M_0^{i-1} . M_{i-1}^i \tag{3.3}$$

$$O_0O_{i,0} = O_0O_{i-1,0} + M_0^i . O_{i-1}O_{i,i} \tag{3.4}$$

where M_0^{i-1} is the transfer matrix from R_0 to R_{i-1}, and M_{i-1}^i is the transfer matrix from R_i to R_{i-1}. The orientation matrix

$$M_0^n = [X_{n,o}, Y_{n,o}, Z_{n,o}] = \begin{pmatrix} X_x & Y_x & Z_x \\ X_y & Y_y & Z_y \\ X_z & Y_z & Z_z \end{pmatrix} \tag{3.5}$$

is used to determine the orientation of the frame R_n with respect to the frame R_0, and the position vector $O_0O_{n,0}$ is used to determine the position of the frame R_n

with respect to the frame R_0

$$O_0O_{n,0} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.6}$$

3.3.3 Direct Geometric Model of the PUMA-560

The fundamental direct kinematics problem for robotic manipulators is to formulate kinematic equations or what we call geometric model that transform the joint space to Cartesian space. The geometric model is denoted by $E = G(\theta)$, where the end effector vector E is a function of the joint variables vector θ. Given the joint variables $\theta_1, \theta_2, ...\theta_n$ we can compute the basic representation of the end effector with respect to a reference Cartesian coordinate R_0 by using the direct geometric model:

$$(\theta_1, \theta_2, ...\theta_n)^t \rightarrow \ \{O_0O_{n,0}(\theta), M_0^n(\theta)\} \tag{3.7}$$

The basic geometrical representation of the arm is reduced to the following expression:

$$G(\theta) = \{O_0O_{n,0}(\theta), M_0^n(\theta)\} \tag{3.8}$$

Since PUMA-560 has six links, the geometric model solution starts by assigning a reference frame to each link and then, tabulating these link-parameters and establishing the transformation matrix M_i^{i-1} for each link. The state of the end effector

link attached frame, with respect to the base reference frame, can be determined by means of the following information:

- The robot hand orientation matrix $M_0^n = [X_n^0, Y_n^0, Z_n^0]$, determines the orientation of frame R_n with respect to frame R_0.
- The position vector $O_0O_{n,0}$, references the origin of R_n with respect to R_0.

The orientation matrix M_0^i is the products of the rotation matrices:

$$M_0^n = M_0^1.M_1^2.M_2^3...M_{n-1}^n \tag{3.9}$$

The position vector can be determined as follows:

$$O_0O_{n,0} = O_0O_{n-1,0} + M_0^n.O_{n-1}O_{n,n} \tag{3.10}$$

Now, assuming joint angles variable $(\theta_1, \theta_2, ...\theta_6, \theta_g)$ with joint angle of shoulder as θ_g and the length of links as $(l_1, l_2, ...l_6, l_g)$ with length of shoulder as l_g,we can describe the geometric structure of the arm in the following topological form:

$$Link_l(ROTZ(\theta_1), Z(L_1))$$

$$Link_2(ROTX(\theta_2), Z(L_2))$$

$$Link_3(ROTX(\theta_3), Z(L_3))$$

$$Link_4(ROTZ(\theta_4), Z(L_4))$$

$$Link_5(ROTX(\theta_5), Z(L_5))$$

$$Link_6(ROTZ(\theta_6), Z(L_6))$$

where $ROTZ(\theta_i)$ is the rotation of $Link_i$ between frames R_i and R_{i-1} about the Z_{i-1} axis, and $Z(L_i)$ indicates that the link body L_i with length l_i is along the Z_i vector.

Consider L_1 which is revolute and defined as a rotation about the Z_o axis, and the link body L_1 is along vector Z_1 of frame R_1. Then, using the above notations we can express the position, O_1 of the end point of the Base link as:

$$O_0O_{1,0} = M_0^1.O_0O_{1,1} \tag{3.11}$$

where,

$$M_o^1 = ROTZ(\theta_1) \tag{3.12}$$

$$M_o^1 = \begin{pmatrix} C_1 & -S_1 & 0 \\ S_1 & C_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.13}$$

The short-hand notation $C_1 = cos(\theta_1)$ and $S_1 = sin(\theta_1)$.

Equation (3.11) can be solved as,

$$O_0O_{1,0} = \begin{pmatrix} dC_1 \\ dS_1 \\ \ell_1 \end{pmatrix} \tag{3.14}$$

Since shoulder has the same orientation as the base link we can express the position of the end point of the shoulder $O_1^{'}$ as,

$$O_0, O_{1,0}^{'} = O_0O_{1,0} + M_0^1.\begin{pmatrix} \ell_g \\ 0 \\ 0 \end{pmatrix} \tag{3.15}$$

The link body L_2 is along vector Z_2 of frame R_2 and is shifted by value d in the X axis direction from the link body L_1.

$$O_0O_{2,0} = O_0O_{1,0} + M_0^2.\begin{pmatrix} 0 \\ 0 \\ \ell_2 \end{pmatrix} \tag{3.16}$$

where,

$$M_0^2 = M_0^1.M_1^2 = ROTZ(\theta_1).ROTX(\theta_2) \tag{3.17}$$

$$M_0^2 = \begin{pmatrix} C_1 & -S_1 & 0 \\ S_1 & C_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & C_2 & -S_2 \\ 0 & S_2 & C_2 \end{pmatrix} \tag{3.18}$$

Thus, the end point of the link2 can be solved as:

$$O_0O_{2,0} = \begin{pmatrix} C_1.d + S_1S_2\ell_2 \\ S_1.d - C_1S_2\ell_2 \\ \ell_1 + C_2\ell_2 \end{pmatrix} \tag{3.19}$$

The link body L_3 is along vector Z_3 and rotates around X axis.

The orientation matrix M_0^3 is given by:

$$M_0^3 = M_0^2.M_2^3 = M_0^2.ROTX(\theta_3) \tag{3.20}$$

$$M_0^3 = \begin{pmatrix} C_1 & -S_1C_2 & S_1S_2 \\ S_1 & C_1C_2 & -C_1S_2 \\ 0 & S_2 & C_2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & C_3 & -S_3 \\ 0 & S_3 & C_3 \end{pmatrix} \tag{3.21}$$

Given that :

$C_{23} = C_2C_3 - S_2S_3$

$S_{23} = C_2S_3 + C_3S_2$

Solving these equations and making similarity to that of end point of $Link_2$ we can derive position vector of end point of $Link_3$ as:

$$O_0O_{3,0} = \begin{pmatrix} C_1.d + S_1S_2\ell_2 + S_1S_{23}\ell_3 \\ S_1.d + C_1S_2\ell_2 + C_1S_{23}\ell_3 \\ \ell_1 + C_2\ell_2 + C_{23}\ell_3 \end{pmatrix} \tag{3.22}$$

Following similar procedure we can derive orientation and position of the $Link_4$ as below:

$$M_0^4 = \begin{pmatrix} C_1C_4 - S_1S_4C_{23} & -C_1S_4 - S_1C_4C_{23} & S_1S_{23} \\ S_1C_4 + C_1C_4C_{23} & -S_1S_4 + C_1C_4C_{23} & -C_1S_{23} \\ S_{23}C_4 & C_4S_{23} & C_{23} \end{pmatrix} \tag{3.23}$$

$$O_0O_{4,0} = \begin{pmatrix} C_1.d + S_1(S_2\ell_2 + S_{23}\ell_{34}) \\ S_1.d - C_1(S_2\ell_2 - S_{23}\ell_{34}) \\ \ell_1 + C_2\ell_2 + C_{23}\ell_{34} \end{pmatrix} \tag{3.24}$$

To simplify, let

$$M_0^4 = \begin{pmatrix} X_{x4} & Y_{x4} & Z_{x4} \\ X_{y4} & Y_{y4} & Z_{y4} \\ X_{z4} & Y_{z4} & Z_{z4} \end{pmatrix} \tag{3.25}$$

And let

$$O_0O_{4,0} = \begin{pmatrix} X_4 \\ Y_4 \\ Z_4 \end{pmatrix} \tag{3.26}$$

Now, using these notations and following previous method, we can derive orientation and position vector for the link body L_5 which is along vector Z_5 and rotates around X-axis as below:

$$M_0^5 = \begin{pmatrix} X_{x4} & C_5(Y_{x4}) + S_5(Z_{x4}) & -S_5(Y_{x4}) + C_5(Z_{x4}) \\ X_{y4} & C_5(Y_{y4}) + S_5(Z_{y4}) & -S_5(Y_{y4}) + C_5(Z_{y4}) \\ X_{z4} & C_5(Y_{z4}) + S_5(Z_{z4}) & -S_5(Y_{z4}) + C_5(Z_{z4}) \end{pmatrix} \tag{3.27}$$

$$O_0O_{5,0} = \begin{pmatrix} X_4 - S_5(-C_1S_4 - S_1C_4C_{23}) + C_5(S_1S_{23}).\ell_5 \\ Y_4 - S_5(-S_1S_4 + C_1C_4C_{23}) + C_5(-C_1S_{23}).\ell_5 \\ Z_4 - S_5(C_4S_{23}) + C_5(C_{23}).\ell_5 \end{pmatrix} \tag{3.28}$$

Finally, the link body L_6 is along vector Z_6 and rotate around Z axis. The basic orientation matrix M_0^6 is computed as follows:

$$M_0^6 = M_0^5.M_5^6 = M_0^5.ROTZ(\theta_6) \tag{3.29}$$

$$M_0^6 = \begin{pmatrix} X_x & Y_x & Z_x \\ X_y & Y_y & Z_y \\ X_z & Y_z & Z_z \end{pmatrix} \tag{3.30}$$

The final components of the orientation matrix are:

$$X_x = C_6(C_1C_4 - S_1S_4C_{23}) + S_6(-C_1C_5S_4 - S_1C_4C_5C_{23}) + S_1S_5S_{23})$$

$$X_y = C_6(S_1C_4 + C_1C_4C_{23}) + S_6(-S_1S_4C_5 + C_1C_4C_5C_{23}) - C_1S_5S_{23})$$

$$X_z = C_6(S_{23}C_4) + S_6(C_4C_5S_{23} + S_5C_{23})$$

$$Y_x = -S_6(C_1C_4 - S_1S_4C_{23}) + C_6(-C_1C_5S_4 - S_1C_4C_5C_{23}) + S_1S_5S_{23})$$

$$Y_y = -S_6(S_1C_4 + C_1C_4C_{23}) + C_6(-S_1S_4C_5 + C_1C_4C_5C_{23}) - C_1S_5S_{23})$$

$$Y_z = -S_6(S_{23}C_4) + C_6(C_4C_5S_{23} + S_5C_{23})$$

$$Z_x = -S_5(-C_1S_4 - S_1C_4C_5C_{23}) + S_1C_5S_{23})$$

$$Z_y = -S_5(-S_1S_4 + C_1C_4C_5C_{23}) - C_1C_5S_{23})$$

$$Z_z = -S_5(C_4S_{23} + C_5C_{23})$$

The end effector position vector is given by:

$$O_0O_{6,0} = O_0O_{5,0} + M_0^6O_5O_{6,6} \tag{3.31}$$

$$O_0O_{6,0} = O_0O_{5,0} + M_0^6.\begin{pmatrix} 0 \\ 0 \\ \ell_6 \end{pmatrix} \tag{3.32}$$

$$O_0O_{6,0} = \begin{pmatrix} X_4 + (l_5 + l_6).Z_x \\ Y_4 + (l_5 + l_6).Z_y \\ Z_4 + (l_5 + l_6).Z_z \end{pmatrix} \tag{3.33}$$

The final position of the end effector is given by :

$$X = X_4 + (l_5 + l_6).Z_x$$

$$Y = Y_4 + (l_5 + l_6).Z_y$$

$$Z = Z_4 + (l_5 + l_6).Z_z$$

The detail derivation of the above equations can be found in [46].

3.3.4 Inverse Geometric Model of the PUMA-560

Determination of inverse geometric model is a lengthy process. Here we have summarized the solutions. The geometric system for the transporter part of the PUMA-560 can be defined by:

$$(\theta_1, \theta_2, \theta_3) \rightarrow \left\{O_0O_4(\theta), M_0^4(\theta)\right\}$$

The inverse geometric model of the transporter consists of finding closed form solutions for θ_1, θ_2, and θ_3 as functions of the transporter end point coordinates X, Y, and Z.

Evaluation of the geometric model of the transporter allows writing the coordinate of vector $O_0O_{4,0}(\theta)$ as follows:

$$O_0O_{4,0} = \begin{pmatrix} C_1.D + S_1(S_2\ell_2 + S_{23}\ell_{34}) \\ S_1.D - C_1(S_2\ell_2 + S_{23}\ell_{34}) \\ \ell_1 + C_2\ell_2 + C_{23}\ell_{34} \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.34}$$

Depending on the sign, we have two solutions for the angle θ_1 .The solution θ_1^+ can be evaluated as follows:

$$\theta_1^+ = Tan^{-1}(S_1^+, C_1^+)$$

where,

$$S_1 = \frac{XD \pm X\sqrt{X^2 + Y^2 - D^2}}{\sqrt{X^2 + Y^2}} \ and \ C_1 = \frac{XD \mp Y\sqrt{X^2 + Y^2 - D^2}}{\sqrt{X^2 + Y^2}} \tag{3.35}$$

To determine a unique solution θ we may compare θ_1^+ and θ_1^- to the previous value of θ which allows satisfying a continuity criteria on the cartesian trajectory.

Angle θ_3 can then be evaluated as follows:

$$\theta_3 = Tan^{-1}(S_3, C_3)$$

where,

$$C_3 = (X^2 + Y^2 + (Z - L_1)^2 - L_2^2 - -L_{34}^2)/2L_2L_{34}$$

$$S_3 = \pm\sqrt{1 - C_3^2} \tag{3.36}$$

The solution for the angle θ_2 depends on the selected values of θ_1, θ_3, and their respective signs. It can be obtained as follows:

$$\theta_2 = Tan^{-1}(S_2, C_2)$$

where,

$$\begin{aligned} S_2 &= \frac{(XS_1 - YC_1)(L_2 + C_3L_{34}) - (Z - L_1)S_3L_{34}}{L_2^2 + L_{34}^2 + 2L_2L_{34}C_3} \\ C_2 &= \frac{(XS_1 - YC_1)S_3L_{34} + (Z - L_1)(L_2 + C_3L_{34})}{L_2^2 + L_{34}^2 + 2L_2L_{34}C_3} \end{aligned} \tag{3.37}$$

The problem of having multiple solutions and singularities is addressed in detail in [46].

Let us now evaluate the inverse geometrical transform for the effector part. It consists of finding the joint variables $\theta_4, ..., \theta_6$ given the hand position and orientation

matrix:

$$\underbrace{O_0O_{6,0}}_{\text{Hand Center}} \text{ and } \underbrace{M_0^6 = \{X_6, Y_6, Z_6\}}_{\text{Hand Orientation Matrix}} \tag{3.38}$$

This system equations consists of twelve nonlinear, redundant equations with respect to $\theta_1, ..., \theta_6$.

As the first step, we have derived the following equations for the cartesian coordinate and the orientation of the end effector as below (details can be found in [46]).

$$O_0O_{6,0} = \begin{bmatrix} S_1(S_2L_2 + S_{23}(L_3 + L_4)) + L5.Z_{x6} \\ -C_1(S_2L_2 + S_{23}(L_3 + L_4)) + L5.Z_{y6} \\ L_1 + C_2L_2 + C_{23}(L_3 + L_4)) + L5.Z_{z6} \end{bmatrix} \tag{3.39}$$

$$M_0^6 = M_0^4.M_5^6 = \{X_6, Y_6, Z_6\} = \begin{bmatrix} X_x & X_y & X_z \\ X_y & Y_y & Z_y \\ X_z & Y_z & Z_z \end{bmatrix} \tag{3.40}$$

where,

$$\begin{aligned} X_x &= C_1C_4C_6 - S_1C_{23}S_4C_6 - C_1S_4C_5S_6 - S_1C_{23}C_4C_5S_6 \\ &\quad +S_1S_{23}S_5S_6 \end{aligned} \tag{3.41}$$

$$\begin{aligned} X_y &= S_1C_4C_6 + C_1C_{23}S_4C_6 - S_1S_4C_5S_6 - C_1C_{23}C_4C_5S_6 \\ &\quad -C_1S_{23}S_5S_6 \end{aligned} \tag{3.42}$$

$$X_z = S_{23}S_4C_6 + S_{23}C_4C_5S_6 + C_{23}S_5S_6 \tag{3.43}$$

$$
\begin{aligned}
Y_x &= C_1C_4S_6 + S_1C_{23}S_4S_6 - C_1S_4C_5C_6 - S_1C_{23}C_4C_5C_6 \\
&\quad + S_1S_{23}S_5C_6 \qquad (3.44)\\
Y_y &= -S_1C_4S_6 + C_1C_{23}S_4S_6 - S_1S_4C_5C_6 + S_1C_{23}C_4C_5C_6 \\
&\quad - C_1S_{23}S_5C_6 \qquad (3.45)\\
Y_z &= S_{23}S_4S_6 + S_{23}C_4C_5C_6 + C_{23}S_5C_6 \qquad (3.46)\\
Z_x &= C_1S_4S_5 + S_1C_{23}S_4C_5 + S_1S_{23}C_5 \qquad (3.47)\\
Z_y &= S_1S_4S_5 - C_1C_{23}C_4C_5 - C_1S_{23}C_5 \qquad (3.48)\\
Z_z &= -S_{23}C_4S_5C_{23}C_5 \qquad (3.49)
\end{aligned}
$$

Based on the above vectors O_0O_5 can be expressed as follows:

$$O_0O_{4,0} = O_0O_{6,0} - L5.Z_{6,0} \qquad (3.50)$$

Equations of X_4, Y_4 and Z_4 are similar to the inverse geometric model of the transporter part. The solutions for θ_1, θ_2 and θ_3 have also the same form as of transporter.

The second step is to determine solutions for θ_4, θ_5 and θ_6 . To identify C_4, S_4, C_5 and S_5, we may use the following equations:

$$M_3^5 = (M_0^3)^{-1}.M_0^6.(M_3^5)^{-1} = M_3^0.M_0^6.M_6^5 \qquad (3.51)$$

Following equations are derived,

$$
\begin{aligned}
C_5 &= C_{23}(S_1Z_x - C_1Z_y) - C_{23}Z_z \\
S_5 &= \pm\sqrt{1 - C_5^2} \qquad (3.52)
\end{aligned}
$$

$$\begin{aligned} S_4 &= \frac{Z_x C_1 + Z_y S_1}{S_5} \\ C_4 &= \frac{C_{23}(S_1 Z_x - C_1 Z_y) - S_{23} Z_z}{S_5} \end{aligned} \tag{3.53}$$

Therefore, two symmetrical solutions are possible for angle θ_5 and θ_4 within $[-\pi, +\pi]$.

To determine θ_6 when $\theta_5 \neq 0$, expression of C_6 and S_6 are then computed from the product $M_5^3.(M_3^0.M_0^6)$ as follows:

$$\begin{aligned} C_6 &= C_4(C_1 X_x + S_1 X_y) + S_4(-S_1 C_{23} X_x + C_1 C_{23} X_y + S_{23} X_z) \\ S_6 &= C_4(C_1 Y_x + S_1 Y_y) + S_4(-S_1 C_{23} Y_x + C_1 C_{23} Y_y + S_{23} Y_z) \end{aligned} \tag{3.54}$$

Depending on the sign of S_5, i.e. C_4 and S_4, we determine two solution for S_6:

$$\begin{aligned} S_5^+ &= +\sqrt{1 - C_5^2} \rightarrow (C_4^+, S_4^+) \rightarrow (C_6^+, S_6^+) \\ \text{And } S_5^- &= -S_5^+ \rightarrow (C_4^-, S_4^-) \rightarrow (C_6^-, S_6^-) \end{aligned} \tag{3.55}$$

Since, two solutions are also expected for θ_6:

$$\theta_6^+(C_6^+, S_6^+)$$

$$\theta_6^- = \theta_6^+ - Sign(\theta_6^+).\pi \tag{3.56}$$

3.4 Building Body Shapes Around the Skeleton of the Graphical Arm

The mathematical model developed so far gives us the skeleton of the Robot arm as shown in Fig. 3.6. To develop a solid robot arm body we need to draw polygonal shapes around the skeleton to look like real PUMA-560 robot.

The base, shoulder and the wrist part of the robot are almost cylindrical. The upper arm and the forearm are somewhat trapezoidal. Since, for telerobotic manipulation our main focus is on the gripper or end-effector's middle point (between two openings), we will simplify the model by drawing the upper arm and forearm as cylindrical. We will model the gripper with rectangular shapes that can be opened or closed by changing the distance between the openings. We will also made simplification in making connection of $Link_2$ and $Link_3$. We have ignored the horizontal shift in both cases and hence those two links have become aligned vertically. But we were cautious about the end points of the polygons specially of the base cylinder. So, we can match our graphic robot with the original robot's image in the stereo space.

Cylinders can be drawn as finite element representation; see Fig. 3.7. Increasing the number of segments or elements will improve the quality of view but it will increase the complexity of the computation. To draw a cylindrical shape, we need

to specify the number of segments. The more segments there are, the smoother and rounder the cylinder will appear.

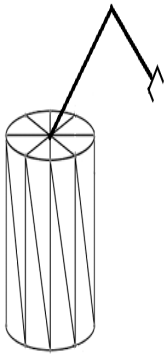

Figure 3.7: Finite element representation of a cylinder used as body shape of graphical arm.

We have alternatives of primitives to be used for drawing the segments of the graphical arm such as line list, line stripe, triangle list, triangle stripe or triangle fan. We have chosen triangle stripe and triangle fan because among the alternatives these are the most memory efficient requiring less number of vertices to be specified.

3.5 Data Structure Design

To design an efficient data structure, we must first consider types of data and nature of manipulation. The robot structure developed in Section 3.3 reveals that PUMA 560 robot is an object of variable geometry where one portion of the object is connected with other portion. Since each link of the robot are represented by position and orientation matrices, movement to a new position or orientation involves mul-

tiplication of two 3×3 matrices and one 3×3 matrix with one 3×1 vector. We have to perform these computations for each point/vertex of the graphical robot. In Augmented Telerobotic systems, robot arm has to be dynamically updated quickly and accurately.

Although modern computers' CPU performance is very high, CPU execution time is very crucial in the field of tele-manipulation as the CPU has to deal with some networking aspect, data acquisition and display as well. To reduce the execution time, we must reduce the computations which eventually necessitate choosing efficient data structure to store vertex data. If we store all the vertex data of the whole robot arm in one array, then we need to perform all the computations even if we change a small portion of the robot arm. An efficient approach would be to keep geometry data of each link of the robot apart from the other link. In that case, to make a change in any link, it will be sufficient to re-compute data of that link and only of those following it.

We should also be careful about the flexibility and generality of the application such that the data structure can be used for drawing different type of robots.

To meet the above objectives we have modeled each link as a different object with properties required to represent the link; see Fig. 3.8. Thus, geometric data of each link is kept apart from the other link but each one is linked with its upper and lower links. The configuration data, such as number of links, number of segments

in cylinder etc. are kept in separate data file. Vertex data of each link is associated with separate vertex buffer.

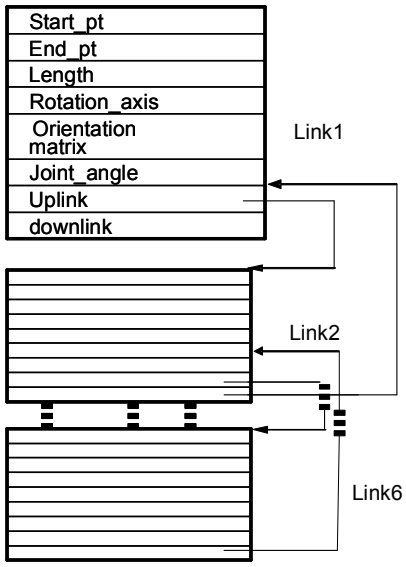

Figure 3.8: Data structure to represent the links.

3.6 Displaying the Graphical Arm

Given the graphical arm structure in its model space, to display it on the screen we need to setup the scene layout and specify the rendering parameters.

3.6.1 Scene Layout Setup

Appropriate transformation matrices are defined to setup the scene. World transformation matrix is used to define the relative position of the objects. Camera position, target position and viewing angle are defined to project the graphic object

accurately. A *material* with the diffuse and ambient color is defined. An *ambient light* that scatters and lights all objects evenly and a *directional light* and its direction are also defined to setup lighting.

The drawing can be represented using wire-frame model or solid model. The psychological studies [47] and the performance studies in telerobotic system of show that there is no significant differences in using different viewing models [48]. But the aircraft simulation studies of [49] suggest that there should be an effect of different viewing models and [3] reports some advantages of using wire-frame model in real-time animation system specially by reducing the occlusion of graphics with real stereo image. We like to provide the user with a choice to select the most appropriate alternative.

3.6.2 Rendering

Since we are not interested in photo-realistic image quality, we have chosen polygon-based rendering for projecting the 3D models onto a 2D image plane. We have used Direct3D's scanline rendering technique as it is faster than ray tracing rendering. It works on a point-by-point basis rather than polygon-by-polygon. To deal with hidden surface determination, we have used Z-buffering algorithm. This technique employs an extra buffer to store the depth of the rendered value at each pixel. If the depth of the polygon that is currently being rendered at that pixel is less than

the z-buffer value at that pixel, the output pixel is overwritten, otherwise nothing is drawn.

A sample view of the displayed graphical arm in solid and wire-frame model with 50 segments in each cylinder is shown in Figure 3.9.

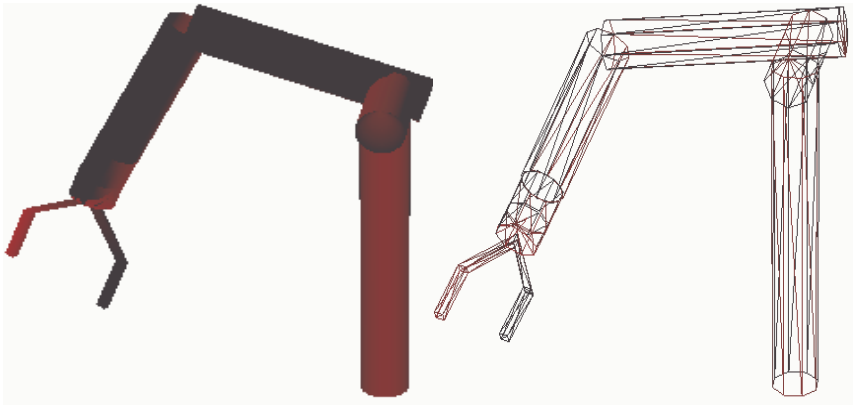

Figure 3.9: 3D PUMA 560 robot structure using cylindrical body shape. Solid model (left) and wire-frame model (right).

3.7 Algorithms for Moving the Graphical Robot Arm

A robot is controlled in the joint space, whereas most of the tasks are done in the Cartesian space. The robot end effector is represented by position vector and orientation matrix in 3D space. Therefore, a point in the joint space can be converted to a position and an orientation in the Cartesian space. Transformation functions relate the position and orientation of the end effector coordinate system to the base

coordinate system. Therefore, we can either use the joint angles as our desired position or we can issue a movement command in the Cartesian space coordinates. More specifically, we can either give an increment in angular position of the robot or we can issue an incremental command in Cartesian space to move the robot from current position to the desired one. Algorithms for moving the real robot that we have used is described in [1]. Here we will describe the algorithms that govern the movement of graphical robot arm in the joint and Cartesian space.

3.7.1 Movement in the Joint Space

Let us assume the current joint position of the robot is kept in a vector, currAngles consisting of 6 double values $[\theta_1, \theta_2, ...\theta_6]$. Whenever a command is received to move a link of the graphical arm incrementally by $\Delta\theta$ in the joint space, new value of the angle (θ) is calculated by adding the increment to the current value ($\theta = \theta + \Delta\theta$). Then, as shown in Figure 3.10, after checking whether the new value is within specified bound, new position and orientation of the joint point of the selected link and the links above that are calculated using the updated angle values sequentially. The computations are performed according the direct geometric equations based on BASE_FRAME of reference.

Once the position and orientation of the end points or skeleton points are calculated, we need to re-calculate the top and bottom vertices of the body shape (i.e.

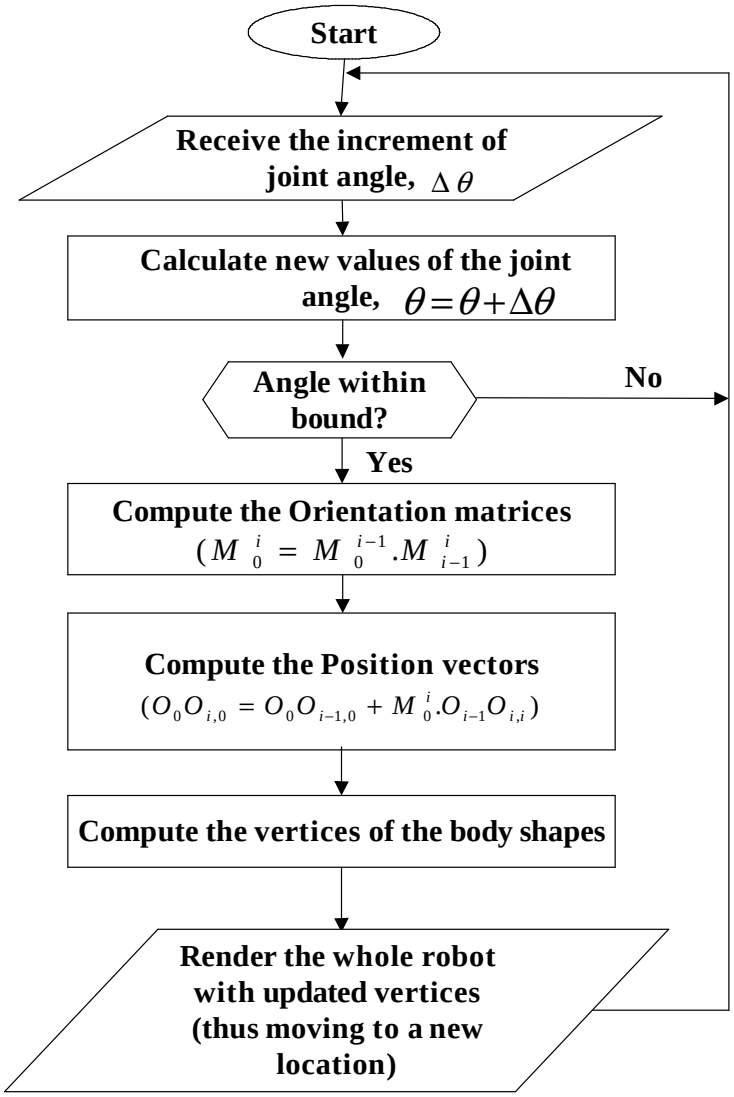

Figure 3.10: Movement of graphical arm in the joint space.

cylinders) as discussed in Section 3.4. Then, the whole graphical robot arm is rendered using the updated vertices. Thus, the arm is moved to a new location with expected position and orientation.

If the absolute values of joint space variables i.e., $\theta_{required}$ (instead of incremental values) are supplied to move to a specified position, then those angle values are directly used for subsequent computation and re-draw the graphical arm as previous.

3.7.2 Movement in the Cartesian Space

To move the graphical robot arm in the Cartesian space we need to take two parameters as input: ΔX and ΔM, where ΔX holds the change in position vector and ΔM is the change in the orientation matrix of the slave arm. At any time t, we need to hold a copy of current position vector $X(t)$, a 3×1 vector, and current orientation matrix $M(t)$, a 3×3 matrix. The new position vector $X_{new}(t)$ and orientation matrix $M_{new}(t)$ are to be calculated from $\{X(t), M(t)\}$ and $\{\Delta X, \Delta M\}$ taking into consideration the current frame of reference. Current frame of reference can be BASE_FRAME, WRIST_FRAME or TOOL_FRAME.

Once $X_{new}(t)$ and $M_{new}(t)$ are calculated, we can use the Inverse Kinematic Model $G^{-1}(X_{new}, M_{new})$ of the PUMA robot to find the joint space variables θ_{new}. This new angle is used to compute position and orientation matrices using Direct Kinematic Model. We also need to re-compute the vertices of the body shapes (i.e.

cylinders) of each link. Then, the whole graphical robot arm is rendered with the updated vertices. The algorithm for moving the graphical robot arm in the Cartesian space is shown in Fig. 3.11.

To move the graphical robot arm to a specified position by supplying the absolute values of a certain position vector and orientation matrix i.e., $\{X_{required}, M_{required}\}$, new angular position is calculated from the supplied values using the Inverse Kinematic Model and the same procedure is used to re-draw the graphical arm in the specified position.

3.8 Acquisition of Real Video Image and 3D Stereo-Visualization

We have adopted the same Client-Server framework for image acquisition and network streaming as used in [1, 50]; see Fig. 3.1.

Real video image of the slave robot at the server side is captured simultaneously by two video cameras. Then, a reliable client-server connection is established. Upon a request from the client, a stereo frame comprising of two pictures is sent over LAN through window sockets. A double buffer concurrent transfer approach is used to maximize overlapped transfer activities between cameras, processor and the network. On the client side, after detecting and making connection with the server, pictures

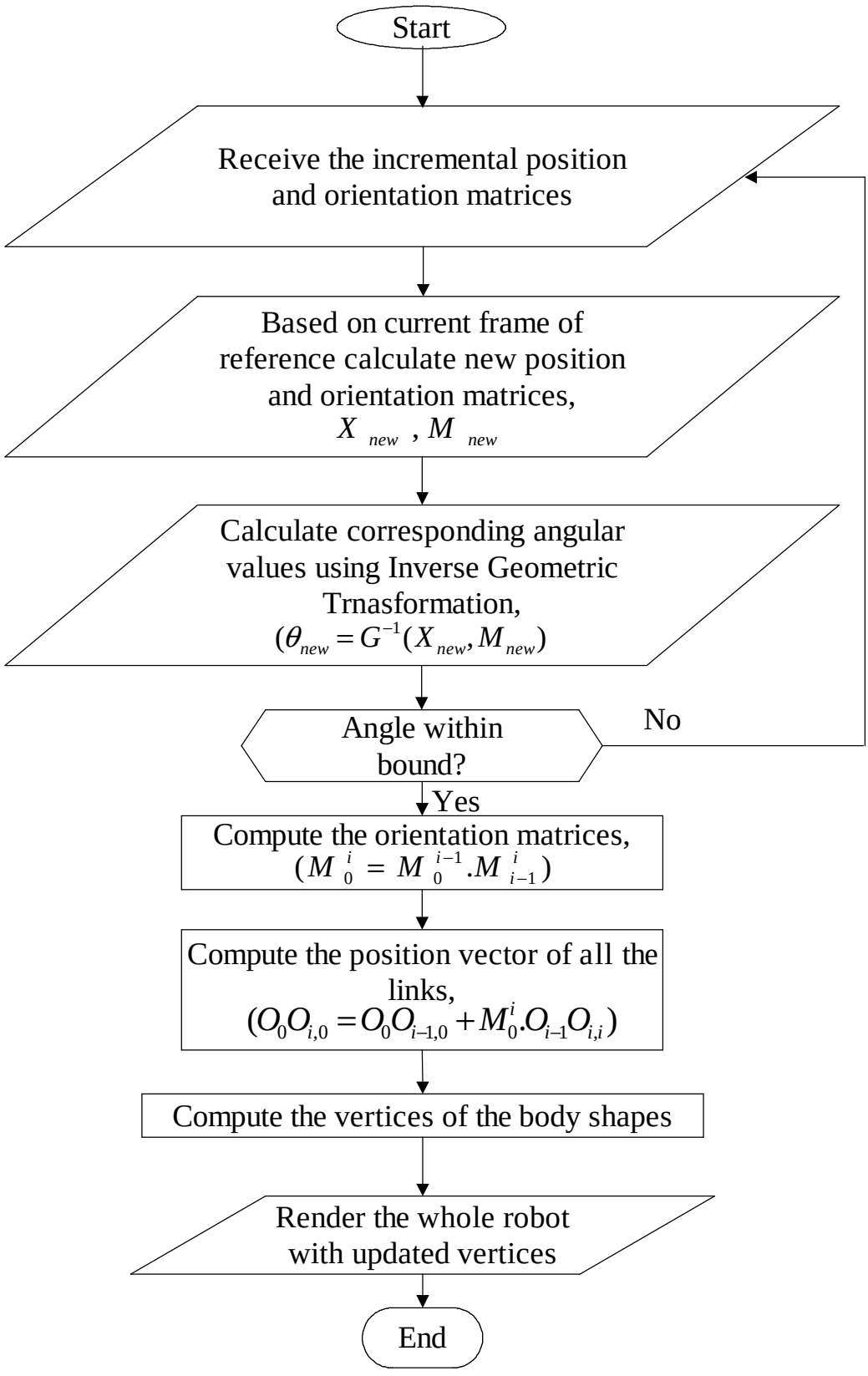

Figure 3.11: Movement of the graphical arm in the Cartesian Space.

Figure 3.12: A sample of the left and right image to be displayed for 3D stereo view.

are received.

Page flipping technique is used for 3D visualization and HMD is used as display device that gives the effect of 3D depth perception. A sample of the left and right image to be displayed for 3D stereo view is shown in Fig. 3.12. The detail of the techniques and devices used is described in Section 2.1.1 and 2.1.2.

3.9 Superimposition of Virtual Object on Real Video

For proper superimposition of graphics on the real video, we need to combine all local coordinate systems centered on the devices and the objects in the scene in a global coordinate system . For this, camera calibration and registration are performed.

Although manufacturers provide a partial set of intrinsic camera parameters, we need to deal with them as they are not accurate enough. Besides, some of these

parameters may vary from time to time, while some of them may be calibrated once for all, depending on the stability of the mechanical and optical construction of the camera. Also, in many situations, the source of the images and therefore camera's internal parameters are not known. We also need to find the position and orientation of the camera i.e. the extrinsic camera parameters.

We have used Heikkila's calibration [30] method implemented in MATLAB Camera Calibration tool box [51] to find out the intrinsic and extrinsic camera parameters. It shows accuracy up to 1/50 of the pixel size. Pinhole camera model with perspective projection and least-square error optimization is used. The intrinsic parameters that can be found by this tool are: focal length (fc), principal point (cc), skew co-efficient (alpha_c) and distortions co-efficient (kc).

MATLAB Calibration Toolbox stores the focal length in pixels in a 2×1 vector **fc** whose elements **fc(1)** and **fc(2)** are the focal distance (a unique value in millimeter) expressed in units of horizontal and vertical pixels. The aspect ratio (τ) is the ratio of the pixel height and width and generally calculated according to Equation 3.57. It can also be defined as the view space width divided by height. It is sometimes different from unity if the pixel in the Charged-Coupled Device (CCD) array are not square.

$$\tau = \mathbf{fc(2)}/\mathbf{fc(1)} \tag{3.57}$$

If P_g (X,Y,Z) is the co-ordinate of P in grid, then the co-ordinate of P in camera reference frame can be expressed as

$$P_c = R_c \times P_g + T_c \tag{3.58}$$

where, translation vector T_c is the co-ordinate vector of the centre of grid reference frame with respect to camera reference frame and the rotation matrix R_c is the surface normal vector of the grid plane in the camera reference frame. External camera parameters are expressed in terms of these two vectors.

We will use perspective projection. So we need to specify the amount of perspective, or zoom, that the resultant image will have. This is done by specifying the viewing frustum, a rectangular cone in 3D that has the from-point as its tip, and that encloses the projection rectangle, which is perpendicular to the cone axis. The angle between opposite sides of the cone is called the viewing angle. The range of its value is from 0 to π. In Fig. 3.13, $\angle$AOC is the horizontal viewing angle, and $\angle$BOD is the vertical viewing angle. The greater the viewing angle, the greater the amount of perspective [52].

MATLAB does not provide the measurement of the angle of the field of view angle which is one of the important parameters used in DirectX to define the virtual camera setup. With reference to the Fig. 2.1 and Fig. 3.13 we can derive the relationship for field of view, **fov** as below:

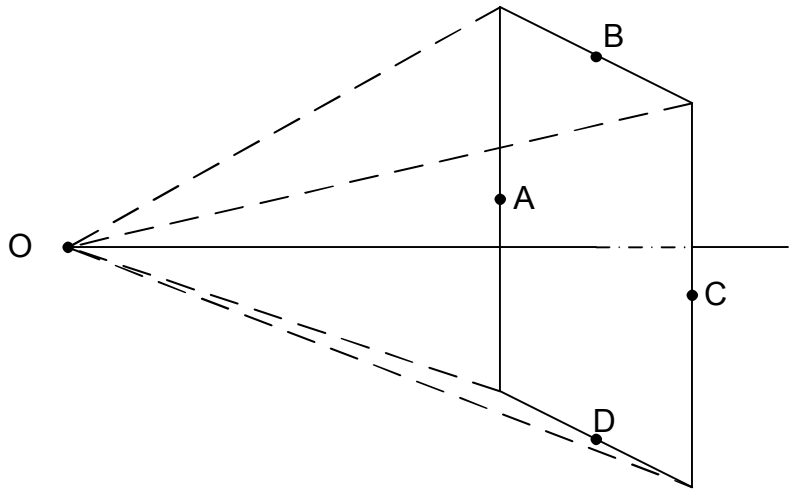

Figure 3.13: Viewing frustum.

$$\mathbf{fc(1)} = (\mathbf{H}/2).cot(\mathbf{fov}_x/2) \tag{3.59}$$

$$\mathbf{fc(1)} = (\mathbf{W}/2).cot(\mathbf{fov}_y/2) \tag{3.60}$$

where, **H** and **W** height and width of the viewing frustum.

3.10 3D Visualization with Superimposed Virtual Objects

The 3D visualization of the graphical objects superimposed on the real stereo image is done in the client side. The stereo video image of the workspace is acquired from the network video stream. It is then copied to a surface, say, *frontSurf* while the

drawing of the current image on graphics screen is in progress in another surface, say *augSurf*. Contents of the *augSurf* are copied to the *backSurf*, the primary surface used to display. Left and right camera images are displayed on two different view-ports on the monitors. The steps of the 3D visualization can be summarized as below:

Step 1: Acquire video image and copy this memory stream to the *frontSurf* surface.

Step 2: Copy the contents of the *frontSurf* surface to the temporary surface *augSurf*.

Step 3: Draw graphical objects or change in graphical objects on *augSurf*.

Step 4: Copy the contents of *augSurf* to the *backSurf* surface that will be used to display.

Step 5: Display the *backSurf* with left camera image to the left view port or monitor.

Step 6: Display the *backSurf* with right image to the right view port or monitor.

Step 7: Observe 3D view with HMD.

Thus, during each flipping operation a complete stereo image is sent down to the HMD. In other words, the stereo video is updated on local display in a page-by-page format and not pixel-by-pixel. This provides a great benefit in terms of reducing time delays.

3.11 GUI Design

GUI is designed for both the server and the client side appliations. The server side GUI forms described in [1] provide user the facility of connection and initialization of the real robot.

The client side GUI is designed to give the user options for connecting to the server PC, master arm and HMD. It facilitates receiving and displaying the stereo video. It also takes user's input for movement of the real and graphical arm for task simulation.

3.12 Input Devices

Flexible algorithms for movement of real and graphical robot arm in joint and Cartesian space by a fixed or incremental value, lead to allow us using any of the input devices such as master arm, joystick, keypad, mouse etc. Keyboard keys and associated functionality provided are listed in Table 3.1.

3.13 Graphical Tele-Manipulation

In teleoperation, if the base link of the real robot remains fixed relative to the video cameras, the base link of the graphical arm will also remain fixed relative to the graphical cameras. The end-effector of the graphical arm then can be manipulated

Table 3.1: Function of the keys used for user interaction.

Keys	**Functions**
Function Keys ($F1$ to $F6$)	Increase the value of current joint angles *by* $\Delta\theta$
Shift and Function Keys ($F1$ to $F6$)	Decrease the value of current joint angles *by* $\Delta\theta$
Function Key ($F7$)	Open the gripper
Shift and Function Key ($F7$)	Close the gripper
Page Up and 'X' or 'Y' or 'Z'	Increase the X, Y or Z component of base reference point respectively
Page Down and 'X' or 'Y' or 'Z'	Decrease the X, Y or Z component of base reference point respectively
Left arrow and 'X' or 'Y' or 'Z'	Rotate about X, Y or Z axis respectively in +ve direction
Right arrow and 'X' or 'Y' or 'Z'	Rotate about X, Y or Z axis respectively in –ve direction
Home and 'X' or 'Y' or 'Z'	Change camera position in +ve direction
End and 'X' or 'Y' or 'Z'	Change camera position in -ve direction
Up arrow and 'X' or 'Y' or 'Z'	Change camera target point in +ve direction
Down arrow and 'X' or 'Y' or 'Z'	Change camera target point in -ve direction

in the graphical coordinate space, relative to objects in the task space (keeping base link in same location of real robot base).

For graphical tele-manipulation, as shown in Fig. 3.14, we need first to connect to the Real Robot Server (running on the server PC connected with PUMA-560 slave arm). Then, the Real Robot arm is initialized with specified or pre-defined convenient position and orientation. The camera calibration parameters (intrinsic and extrinsic) computed earlier using MATLAB Camera Calibration Toolbox are then loaded to the program. Here we also need to specify the graphics parameters such as viewing model (solid or wire-frame), display device (HMD or monitors) etc. If Master Arm is chosen as a tool for user interaction then we need to start the process of engaging it i.e. communicating with the server system. Then, we need to be connected with the Vision Server running on the Server PC to capture video images through camera. If HMD is selected as displaying device for 3D stereo view we need to connect it with the client system. Otherwise, monitor can be used to display the stereo using Sync-Doubling techniques (displaying left and right images up and down on the monitor screen).

Once we have real and virtual image ready to be displayed, we need to perfectly superimpose the graphics onto the real video image by adjusting the camera calibration parameters. Then, for task simulation e.g. pick and place operation, we need to move the graphical arm using the algorithms described in Section 3.7. If

the movement is satisfactory, it will be saved to be used while issuing command to the real robot. At the end of simulation, before exiting the program, we need to disconnect the client system from the robot server and the vision server running on the server PC.

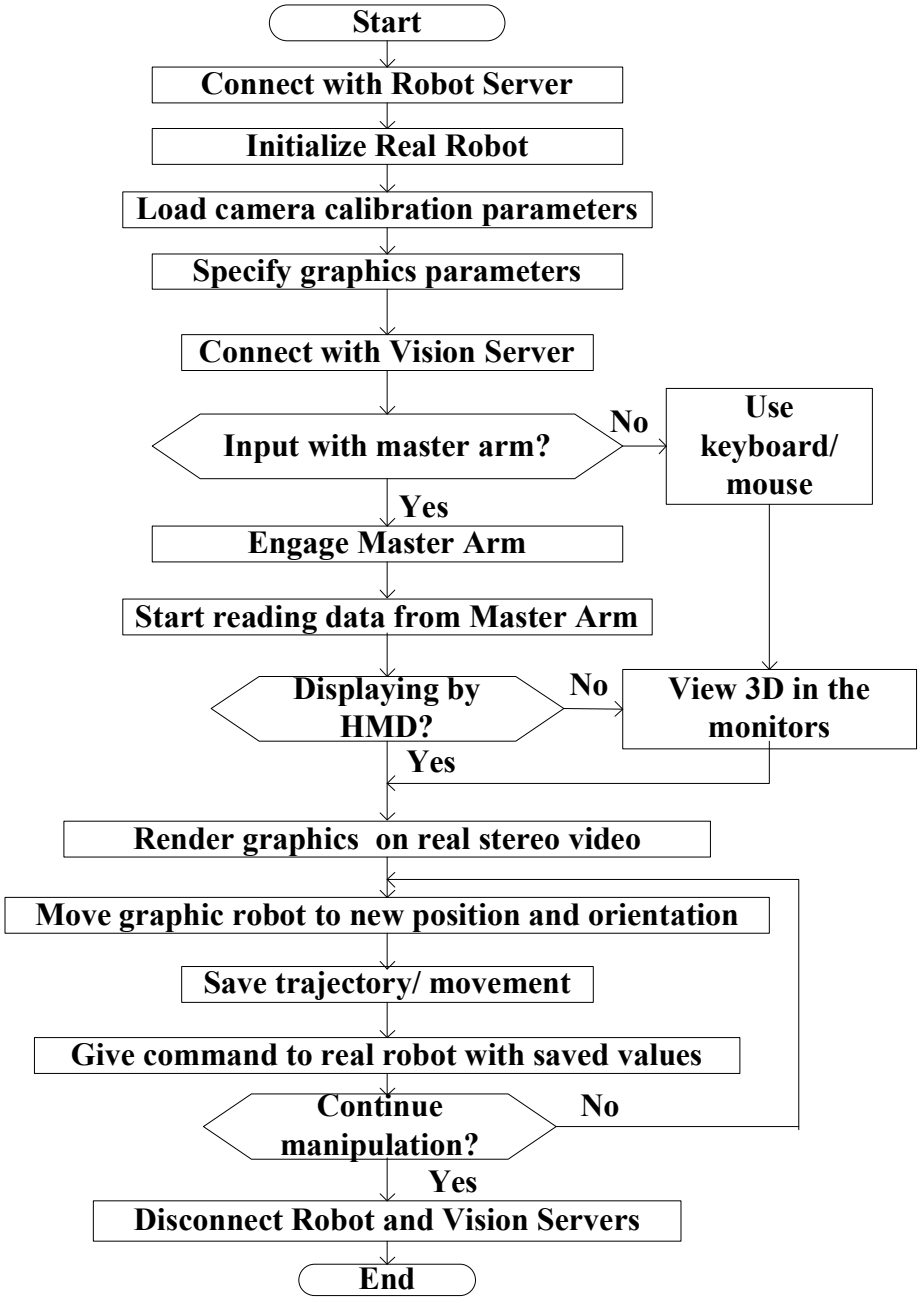

Figure 3.14: Overall client system flow-chart for graphical tele-manipulation.

Chapter 4

Implementation

In this chapter, we will describe how the design concepts described in the previous chapter are implemented to fulfill our objectives . As noted in Section 1.6, Microsoft $.NET$ framework is used to develop the overall client-server framework. The Vision Server running on the server PC (connected to the PUMA 560 robot) and developed by [1] is used for capturing the real video image of the robot's working site. Graphics designs are implemented using DirectX 9.0 functionalities with the $.NET$ Visual $C\#$ programming. In the following sections, we will describe all implementation issues in detail.

4.1 Motivation for using Direct3D API

Among the available alternatives of 3D graphics API (Application Programming Interface) such as windows GDI, Direct3D, Java3D and OpenGL, we have chosen Direct3D. A comparative study is made among various APIs in Section 5.4.2. The main features that motivated us to choose DirectX are summarized below:

- Direct3D increase performance by using hardware acceleration and therefore is used to render 3D graphics in many applications where performance is critical.
- Direct3D also allows applications to run full-screen instead of embedded in a window, though they can still run in a window if programmed for that feature. As we are using HMD this feature is very helpful in our case to have the 3D effect.
- Direct3D perfectly match with Microsoft's various Windows operating systems (Windows 95 and above) and with Microsoft .NET framework which are used in our telerobotic client-server framework.

4.2 Description of DirectX Features Used

Since we are using Microsoft DirectX 9.0 for our graphic design implementation, it is worth describing some of the important features of it which will help in understanding the detail implementation. MS DirectX is a set of low-level APIs for creating

high-performance multimedia applications including supports for 2D and 3D graphics, sound effects, input devices, and networked applications [53, 54]. Some of the features of DirectX 9.0 used in our application are described below:

Direct3D, an API of *DirectX*, delivers real-time full 3D rendering and transparent access to hardware graphics acceleration boards. In other words, it allows Windows to make use of the advanced graphics capabilities found in 3D hardware graphics boards. However, in doing so it utilizes the *HAL*. *HAL* is an extremely thin layer that wraps around the *Device Driver Interface (DDI)*, abstracting it away. Details about *HAL* and how it provides hardware acceleration will be discussed in Section 5.4.2. Another piece of code that runs in Direct3D applications is Hardware Emulation Layer (HEL). It is larger than HAL and performs works with the CPU. It emulates what the hardware would do in the event that a desired effect in Direct3D is not supported by the HAL. The HEL is considerably slower than the HAL because it is not asynchronous and it needs to use the CPU, which is not specialized for graphics operations [55].

A *Microsoft Direct3D device* is the rendering component of Direct3D. It encapsulates and stores the rendering state. In addition, a Direct3D device performs transformations and lighting operations and rasterizes an image to a surface.

Microsoft DirectX currently supports two main types of Direct3D devices: a hardware device with hardware-accelerated rasterization and shading with both

hardware and software vertex processing; and a reference device. Hardware-accelerated devices give much better performance than software/reference devices [53].

Before creating a *Direct3D device* we need to specify the *presentation parameters*. Some of the important presentation parameters are *DepthFormat* (format of the depth stencil surface the device creates), *BackBufferCount* (the number of back buffers), *BackBufferFormat*, *BackBufferHeight*, *BackBufferWidth, DeviceWindow* (setting or retrieving the display window), *EnableAutoDepthStencil* (a Boolean value that indicates whether Microsoft Direct3D manages depth buffers for an application), *FullScreenRefreshRateInHz* (the rate at which the display adapter refreshes the screen), *SwapEffect*, *Windowed* (a Boolean value that indicates whether an application is running in a windowed mode).

We have used *DirectX Surfaces* for drawing and displaying purposes. Internally, a *DirectX Surface* is just a structure that manages image data as a contiguous block of memory. The structure keeps track of the vital statistics of the surface, such as its height, width and format of the pixel. We must specify the dimensions, and color pellet while creating a surface. By default, *DirectX* will try to create the surface in accelerated video memory on video card but if there is not enough room to create the surface in this memory, it creates the surface in system memory. The primary surface is the pixel array that is visible on the output video device. This is always on the video card if it has enough memory.

There is only one primary surface per *DirectX* device. However, we can create off-screen surfaces for other purposes, like drawing and blitting, etc. Again, the off-screen surfaces should ideally be created in the accelerated graphics memory for minimum system delays. A pointer to the primary surface can be attained by asking for a *BackBuffer* from the *DirectX Device*. It is typical to have a *BackBuffer* for the image on primary surface. The *BackBuffer* can be switched easily with the current displayed frame. The purpose of this framework is to allow maximum flexibility while drawing 3D objects onto the screen. The frame data that is to be displayed next on the screen in generally manipulated on the off-screen surfaces. Other than the primary surface we have created two off-screen surfaces for our program.

Once every video frame, the back buffer is updated from one or more off-screen surfaces and then the back buffer is presented to the display screen. This process is called page flipping. During this process, the graphics microprocessor flips the addresses of front and back buffers and the next image drawn on the screen comes from the previous back buffer. While the previous front buffer is now back buffer and is ready to be used for the coming video frame. Ideally this process takes place in video hardware and is extremely fast not allowing any shearing or tearing of the image while changing from one video frame to the next.

In DirectX, each vertex data is stored in *Vertex buffer* structure. There are different types of vertex buffer available in custom vertex class of *CustomVertex*.

We have used *CustomVertex.PositionNormal[]* which can be used to store both position and normal data of a vertex.

The *SetStreamSource* method binds a vertex buffer to a device data stream to create an association between the vertex data and one of several data stream ports that feed the primitive processing functions. The parameters for this method are the number of the data stream, the name of the *VertexBuffer* object, and the stream vertex stride.

BeginScene and *EndScene* pair methods are used to generate a scene to be rendered by DirectX device. Every call to BeginScene should eventually be followed by a call to EndScene. When EndScene succeeds, the scene is queued up for rendering by the driver. Hidden Surface Removal (HSR) is also taken care of in this pair of methods. *Device.Present* method is used to finally display the generated scene in the screen. That is, this method is used to move the contents of the back buffer to the primary surface.

4.3 Transformations involved in DirectX Vertex Processing

Number of transformations takes place while vertex data gets converted from one co-ordinate space to another on its way through the conversion from model space

to pixels on a screen. The transformations are explained below mostly according to [4] and shown in Fig. 4.1.

1. *Transformation into World Space*: This conversion transforms all the objects within a scene. The transform can scale, rotate, translate, or skew objects relative to the origin in world space. The result is that each object is oriented, scaled, and rotated relative to the other objects just as they are in a 3D scene.

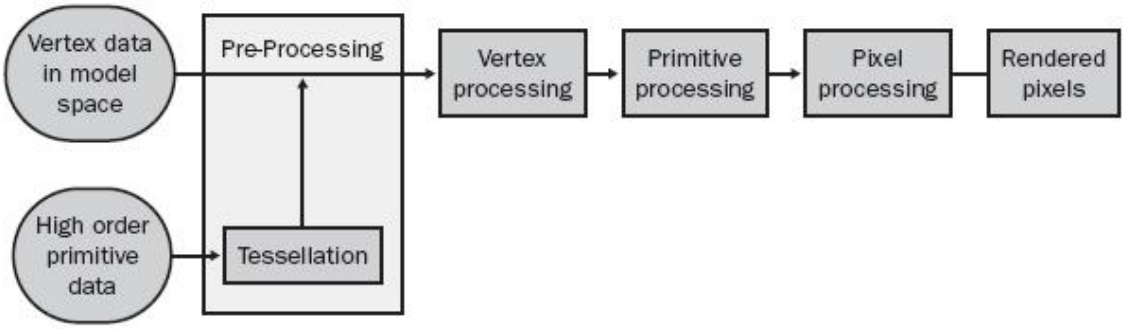

Figure 4.1: Transformation involved in vertex processing [4].

2. *Transformation into View Space*: This conversion orients the camera with respect to the objects. After conversion, objects are said to be in view space, which is commonly called camera-space because the objects are located relative to the camera. In other words, the camera is placed where a viewer looks at the scene.

   There's more than one way to create a view matrix. In all cases, the camera has some position and orientation that's used as a reference point. The view matrix translates and rotates the camera relative to the models. One way to

create a view matrix is to combine a translation matrix with rotation matrices for each axis.

View space assumes that the camera is at the origin of view space looking in the positive Z direction. With left-handed coordinates, positive Z-axis is into the screen. To generate a view matrix, we must pick values for an eye point, an up vector, and a look-at point. The eye point is the position of the camera (or viewer). The up vector is a vector that points up. Usually, (0,1,0) is selected. It is a convenient way to flip the scene upside down with a sign change. The look-at point is the point in the scene that the viewer is looking at.

3. *Camera Model and Projection Transformation:* The projection transform converts vertex data from view space to projection space. The transform performs a linear scale and a nonlinear perspective projection. The effect is to expand objects near the camera and shrink objects away from the camera, which generates the same kind of perspective we see in real life, where objects closer to the camera appear larger than objects farther away. The projection transform can be visualized as a viewing frustum as discussed in Section 3.9. The purpose of the frustum is to identify which objects will be rendered in the view. Objects between the near and far clipping planes (which define the distances at which geometry should no longer be rendered) and inside the diagonal edges of the viewing frustum will be seen by the camera.

For the projection matrix, we set up a perspective transform (which transforms geometry from 3D view space to 2D view port space, with a perspective divide making objects smaller in the distance). To build a perspective transform, we need to specify the field of view, the aspect ratio, and the near and far clipping planes.

4.4 Virtual Object Modeling Module

This module defines the structure of the virtual objects (robot, cubes etc.) and handles functions like movement of virtual objects.

As mentioned earlier, we have used Microsoft DirectX's Direct3D library functionalities to draw the graphical robot arm in the client side. To implement the equations involved in robot's mathematical model described in Section 3.3, we have defined the following data structures.

A class is defined for modeling each link as an object with properties mentioned in Section 3.5. Instances of this object is created at the initialization taking data from user's robot configuration data file.

The *currAnglesOfRob* is a double array structure and is used to store the current value of the joint angles. Microsoft's Vector3 data structure array *RobArm* is used to hold the vertex data of each of the links end point. We have also defined the *OrientationMat[]* data structure to hold the nine elements of the orientation matrix.

The following two methods are used in calculation purposes:

- *OrientationMat MultiplyOrMat* (OrientationMat orientMatrix1, OrientationMat orientMatrix2): It calculates the multiplication of two orientation matrices.

- *MultiplyOrMatTransVecOrientationMat* (orientMatrix1, Vector3 transVec): It is used to calculate multiplication of an orientation matrix with a Vectro3 position matrix.

Options are provided to draw the cylinders of the arm shapes with 8, 50 and 100 segments. The height and the radius of the cylinder are defined according to those of the links of PUMA 560. The position of the vertices together with their *normal* value are also defined .

The cylinders used for drawing body shapes are made up of two triangle fans, one for the top and one for the bottom. The sides of the cylinders are made up of a triangle strip; see Fig. 3.7. Although there are shared vertices (between the sides and the top/bottom faces), to create our cylinder we have not used an index buffer. Only a vertex buffer used as we want different normals so the edges around the top and bottom appear sharp.

Separate methods of type *public void* have been used to draw different links of the robot arm independently using the current values of the joint angles. Also to store up the vertex data separate vertex buffers are used. Following are the methods

used for drawing different links of the robot arm structure.

- **DrawBase** (double [] currAnglesOfRob): This method is used to draw the base or link1 of the robot arm. It uses vertex vb1 of type PositionNormal[] to store the vertices. We have considered a co-ordinate system with positive Z-axis as upward, positive X-axis as outward and positive Y-axis in the right. The start point of the link is considered as $(0, 0, 0)$ and end point as $(0, 0, l_1)$ where l_1 being the length of the base.

- **DrawShoulder** (double [] currAnglesOfRob): This method is used to draw the shoulder of the robot.

- **DrawLink2** (double [] currAnglesOfRob): This method is used to draw $Link_2$ of the robot arm.

  DrawLink3 (double [] currAnglesOfRob) and *DrawLink4* (double [] currAnglesOfRob), *DrawLink5* (double [] currAnglesOfRob) are similar methods for drawing $link_3$, $link_4$ and $link_5$ respectively.

- **DrawLink6AndGripper** (double [] currAnglesOfRob): This method is used to draw the gripper of the robot. At first, the position and orientation of the starting point of the gripper are calculated. Then, four cubes with user defined height and width are drawn. Cube points are calculated using FindCubePoints (Vector3 lowerLeftCorner, float hight, float width) and FindTiltedCubePoints

(Vector3 UpperLeftConrner, float height, float width) methods. Getting the cube points with respect to starting point, corresponding orientation matrix is calculated and final vertex positions are saved in vertex buffers specified for each cube separately.

- **DrawWholeRobot()**: This method is used during initialization to draw the whole robot by calling all the methods for drawing individual links.

4.5 Display Module

The DXInterface component is used as the main display module of our graphics system. It handles with synchronization of real and virtual data Projection on video surface, augmentation of real video and page flipping for stereo visualization.

In our implementation, Direct3D API functionalities of Microsoft DirectX9.0 are used to display or rendering the real and virtual images on the monitor and HMD in the client side application.

CamPos, TargetPos, ViewUpAxis and baseRef are four Microsoft Vector3 data structures that are used to store the co-ordinate of the camera position, the target point or the eye-point, the up axis and the reference of the base point respectively. These variables are assigned from the result we obtain from MATLAB Camera Calibration Toolbox during the initialization of the application.

- *public bool* **InitializeGraphics()**: This method is used to initialize the graphics device and assign the graphics parameters. At first, the presentation parameters are defined. We chose to discard the *SwapEffect*, selected Immediate type of *PresentationInterval*, made the *EnableAutoDepthStencil* Boolean as true, *AutoDepthStencilFormat* as DepthFormat.D16, back buffer width and height as 1280 and 480 pixels. Display mode is also defined as system adapter's current display mode. Then, a Direct3D device of Hardware type is created with *SoftwareVertexProcessing* option.

  One primary surface (*backSurf*) and two off-screen plain surfaces (*frontSurf* and *augSurf*) are created.

  Two events are initiated to be fired on Device reset and lost. The difference between these two events is that *DeviceLost* gets called earlier, giving us a chance to do the cleanup that needs to occur before we can call *Reset()* successfully.

  The methods for setting up the *Device* and drawing the initial state of the graphic robot are also called in this method.

- *protected void* **SetupDevice()**: In this method, all the necessary vertex buffers are created to hold vertex data of joint points and body shape vertices of different links. We have chosen CustomVertex of type PositionNormal so that we can store both position and normal data of a vertex. Then, alpha blending property of the device is enabled, the source and the destination

blend states are set, culling is turned off so that we can see the front and back of the triangle (primitive used for drawing the robot arm). The Z-Buffer is turned on and lighting option is made true. The filling mode of the drawing is set according to the user's choice (solid or wireframe).

A *DirectInput* device is also initialized (respective method is called) to get user's response through mouse or keyboard. Since word transformation matrix is almost fixed, this Matrix is also assigned in this method.

- *private void* **SetupMatrices()**: View transformation and Projection transformation to setup the graphical camera are defined in this method. Associated camera calibration parameters i.e. camera position, target point, field of view, aspect ratio etc are bound to the respective transformation matrix as described in Section 4.3. We chose to build a left-handed perspective projection matrix based on a field of view.

- *private void* **SetupLights()**: Lighting parameters are defined in this method.

- *private void* **Render()**: This is the main and most important method of the display module. Two *Viewports*, leftViewPort and rightViewPort are created to display graphics on left and right camera image respectively.

  All the drawing commands are kept within the device.BeginScene() and device.EndScene() method pair. At first, stereo image acquired by *StereoClient*

component is copied to *frontSurf* surface using the *SurfaceLoader.FromStream* command. Then, using SurfaceLoader with FromSurface, frontSurf is copied to augSurf. Then, StretchRectangle of device is used to copy the contents of the source rectangle (with content of augSurf) to the destination rectangle (with content of backSurf). Finally, Device.Present() is called to display the content of the back surface to the screen of monitor or the HMD.

4.6 Integration to the Stereo-Vision System

The Multi-threaded Distributed Telerobotic Stereovision System chosen is implemented using Microsoft Visual C++ (for Vision Server component), C# (for Robot Server and Robot Client components) and Microsoft DirectShow of DirectX (for image acquisition through digital camera). We have integrated this system with our graphics system modules to develop a complete AR telerobotic stereovision system.

The client system of the chosen stereovision system consists of VisionClient, a module for stereo image acquisition from the network, master arm module and robot control module. As shown in Fig. 4.2, our camera calibration module takes left and right images from VisionClient module. Our DirectX Interface module also gets the stereo video data from the same module. Robot initialization and getting current position and orientation of real robot are achieved through Robot Model and fed to the Virtual object Modeling module. Master arm component is also used to use

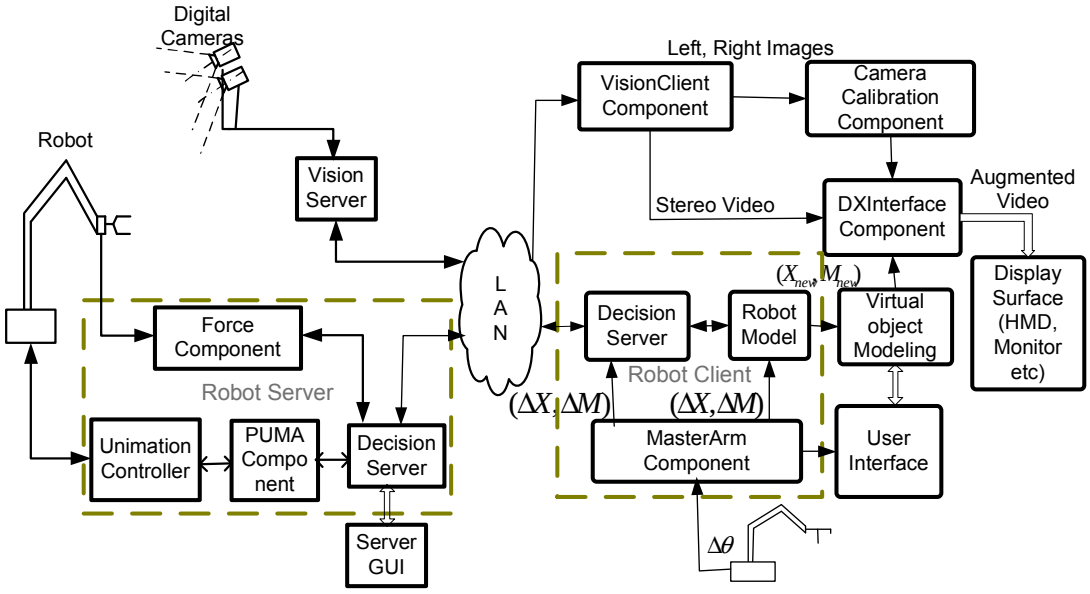

Figure 4.2: Overall software architecture.

master arm as the manipulation tool of graphics robot arm.

4.7 GUI Implementation

Two GUI forms are designed in the client side to provide users interacting with the system. On the main GUI form shown in Fig. 4.3, buttons are attached for connecting with vision and robot server running on the server PC connected with the real robot. It also takes user's option for various graphics parameters. The stereo form GUI shown in Fig. 4.4, can be accessed by pressing the "View 3D Augmented Reality Simulation" button. Stereo form shows the left and right image to two separate monitors (or on the two eye sides of the HMD). It also displays the control features to change the joint angles of real and graphic robot arm, changing

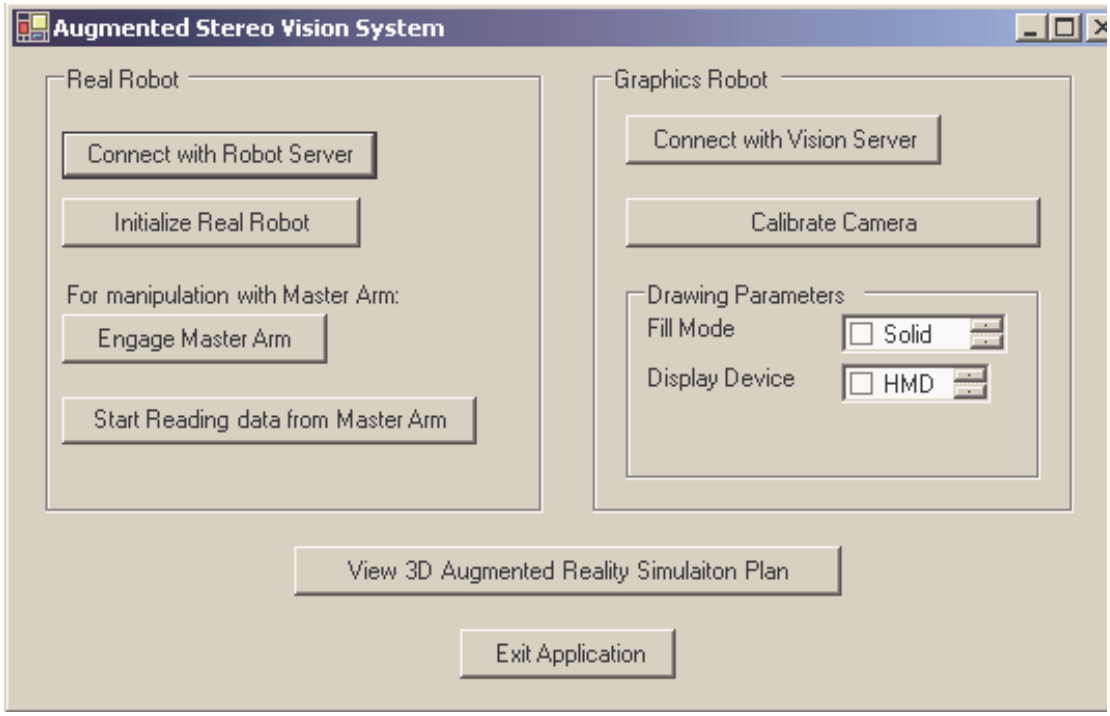

Figure 4.3: Main User Interface Form at Client Side

Cartesian position and opening or closing of gripper which may be used for task simulation.

DirectInput functionality of Microsoft *DirectX*9 is used to implement interaction with user's input. The methods implemented related to *DirectInput* in the client GUI are as follows:

- CreateDirectInputDevice(): It sets necessary parameters and then instantiate a DirectX device.

- FreeDirectInput(): This method is used to un-acquire the device one last time just in case the application tried to exit while the device is still acquired.

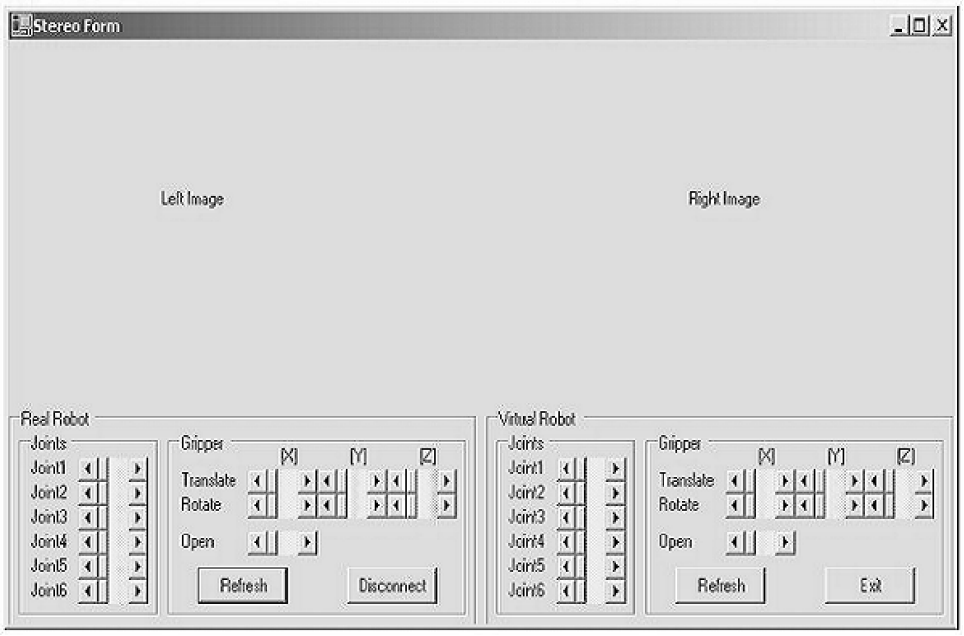

Figure 4.4: User interface form in stereo view at client side

- ReadImmediateData(): It read the input device's state and make changes in appropriate parameters. Then, it calls the graphics rendering method to redraw the virtual robot reflecting users response as described in Section 3.2.

Chapter 5

Performance Evaluation and Comparative Studies

Both quantitative and qualitative evaluations are done for analyzing the performance of the system. In quantitative evaluation, we have measured the execution time of various operations, refresh rate of the output screen, complexity of the algorithms used and the accuracy of the registration of the real and virtual image. Our qualitative evaluation includes comparative studies of our basic tools used with probable alternatives. We have also compared the performance of our systems with similar other approaches.

For quantitative analysis, data was taken by averaging over 1000 samples and running both server and client systems on PCs having 2-GHz Intel P4 processors

with 1GB DRAM and 512 KB cache memory and connected to a campus network by using a 100 Mbps NIC card. The server PC is interfaced to two Sony Handycam digital cameras using a 400 mbps FireWire PCI (IEEE-1394) card.

5.1 Speed of Rendering Graphics

In this section, we will describe the measurement of the refresh rate of the output screen at different levels of graphics complexities and execution environments. We will also describe the results for measuring the time for drawing the graphic robot with different features.

Table 5.1: Refresh rate of the output screen.

Environment	**Graphics Complexities**	**Avg. Refresh Rate (fps)**
Only graphics	No drawing	273.36
	With an object in the scene	243.74
	Only robot with 8 segments in each cylinder	253.59
	Only robot with 50 segments in each cylinder (solid view)	239.78
Graphics on video	Without any drawing	11.498
	With an object in the scene	11.384
	Only robot with 8 segments in each cylinder	11.347
	Only robot with 50 segments in each cylinder (solid view)	11.325

5.1.1 Refresh Rate

We have computed the refresh rate of the output screen displaying the graphics in terms of frame per second. We have tested the system for different environments: with or without overlaying graphics over the real video. For each of the environments, we have recorded the refresh rate with no drawing displayed, graphical arm with a cuboidal object, graphical arm with 8 segments per cylinder and graphical arm with 50 segments per cylinder. As shown in Table 5.1, refresh rate is proportional to the complexity of the drawings and image acquisition time.

5.1.2 Time Required for Rendering the Robot

Time required to render the whole robot with different numbers of triangles in a cylindrical shape and different views is shown in Table 5.2. It is observed that there is no significant difference in performance for using solid or wire-frame model.

5.1.3 Time Required for Video Image Acquisition and Transfer

Image acquisition at server and transfer over LAN using our stereovision system with various schemes are reported in [1, 56]. The results are summarized in Table 5.3.

In the case of a single stereo thread, for the distribution of inter-arrival times of

Table 5.2: Time required to render the graphical robot.

Environment	Graphics Complexity	Rendering Time (ms)	
		Wire-frame	Solid
Only graphics	Only robot with 8 segments in each cylinder	64.088	64.283
	Only robot with 50 segments in each cylinder	65.034	65.151
Graphics on video	Only robot with 8 segments in each cylinder	105.02	108.362
	Only robot with 50 segments in each cylinder	105.37	109.037

Table 5.3: Image acquisition and transfer time.

Operations	**Time in ms**
Copying image from Sample grabber to DRAM with single thread	24.025
Copying image from Sample grabber to DRAM with one thread copying stereo frame and another to read a force data with a transfer	33.46
Copying image from Sample grabber to DRAM with one thread copying stereo frame and another to read a force data without any transfer	60.48
Image transfer over a LAN with double buffer, concurrent transfer	59
Reading image from cameras and displaying in HMD	83

300 video frames, the mean value of 24.025 ms is required.

In the case of a stereo copying thread with a force thread, the force is read as fast as possible without data transfer over the network. This helps in studying the effects on the copy times of video data from SampleGrabber to main memory which allows assessing the performance of the multi-threaded stereovision system.

In the case of a stereo copying thread with a force thread, the force is read as fast as possible without data transfer over the network. The addition of a new force thread on the server causes the mean value to increase from 24.025 ms to 60.48 ms and 90% of the data lies between 8 and 150 ms. Active force thread has significant effect on stereo copying on the server due to PCI resource conflicts. This is a useful feedback to the microprocessor and motherboard architectures.

In the case of stereo copying thread with force thread reading and transferring over the network, a blocking socket is used to transfer force packets and the mean stereo copying time decreases to 33.46 ms. The improvements is due to release of the CPU resources to the video copying thread during blocking transfer of force packets.

Image transfer over a LAN with double-buffer concurrent transfer requires 59 ms and time required from reading image from cameras and displaying in HMD is recorded as 83 ms.

5.2 Complexity of the Movement Algorithm

In the algorithm for moving the graphic arm in the joint space, we need to re-calculate the value of the start and end points of the links above it which may vary according to the link angle chosen. The number of segments used to draw a cylinder is fixed for a particular configuration and we are using only one loop with maximum size equal to the number of segments. Therefore, we can express the number of points to be re-calculated for moving the angle of link n, by an infinitesimal angle $\Delta\theta$ by the following equation:

$$T_n(\Delta\theta) = (2 * s + 1) * (p - n) + g \tag{5.1}$$

where, s is the number of segments used for drawing a cylinder of body shape, p is the number of links, n is link whose angle is to be changed with values 1 to p (considering, n=1 is for the base link and n=p is for gripper) and g is the number of gripper points to be drawn.

From the above expression, we can derive the time complexity of our algorithm as $O(n)$. In Table 5.4, an example is shown for s=50, p=6 and g=68.

Table 5.4: Number of vertices to be drawn for movement in the joint space.

Link whose angle to be changed, n	Number of vertices to be calculated, $T_n(\Delta\theta)$
1(Base)	573
2	472
3	371
4	270
5	169
6(Gripper)	68

5.3 Accuracy of the System

In this section, we will evaluate the accuracy of our calibration method, graphics parameter computation method and the accuracy of the movement algorithms. The re-projection errors of the calibration system, accuracy of the movement and matching error in real and graphic images will be discussed.

5.3.1 Re-projection Errors

To evaluate the accuracy of our calibration method, the projection of grid points are re-computed using the calculated calibration parameters and projected onto the original grid image. The errors are analyzed using MATLAB calibration Toolbox and computed as the standard deviation of the re-projection error (in pixel) in both x and y directions respectively. Fig. 5.1 shows re-projection error of one image. Pixel errors are concentrated to the centre of the graph, mostly within range -0.2

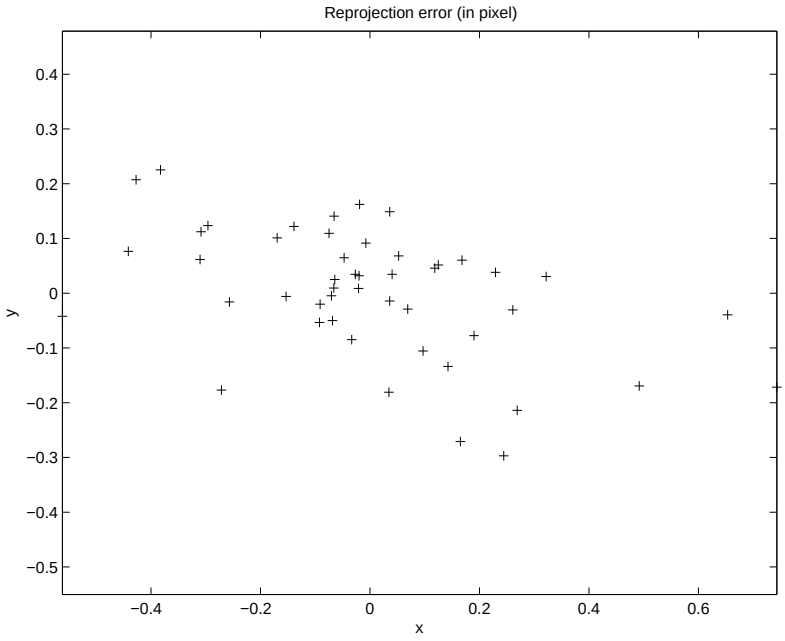

Figure 5.1: Re-projection error in pixel.

to +0.2 in both x and y directions. The errors can be even minimized (up to 1/50 of a pixel size) by identifying the location of the pixels that create larger errors and re-defining the window size used to identify the grid corners.

5.3.2 Accuracy of the Movement

The accuracy of the movements of the graphical object on the output screen mainly depends on the accuracy of the calibration. But it also depends on the accuracy of the computation of the graphics parameters. We have computed the position of the graphic robot arm before and after the movement and compared the difference with the real data provided by the user through user interface.

5.4 Comparative Studies

In this section, several comparative studies are made. The client-server communication platform chosen is compared with available other alternatives. Direct3D, the chosen 3D graphics API is compared with other familiar APIs. Lastly, our overall system is compared with other proposed systems.

5.4.1 Comparison of .NET Framework to Other Client-Server Communication Platforms

The .NET architecture by Microsoft has replaced the DCOM, previously used for distributed computing on, mainly, Windows-based machines. In .NET, the COM (Component Object Model) is replaced by CLR (Common Language Runtime) that supports and integrates components developed in any programming language conforming to CLR specifications. .NET is a loosely coupled architecture for distributed applications. The remote access is based on XML and SOAP (Simple Object Access Protocol) technologies.

The .NET provides two main strategies to use distributed objects: Web services and .NET Remoting. Web services involve allowing applications to exchange messages in a way that is platform, object model, and programming language independent. Web services use XML and SOAP to form the link between different objects. Remoting, on the other side, relies on the existence of the common lan-

guage runtime assemblies that contain information about data types. For the closed environments where faster connections are required, .NET Remoting is an ideal solution cutting the overhead caused by object and data serialization through XML. It is an automatic notification and data messaging mechanism between the robot server where an event is occurring and a client where the notification is required. It does not require client-side components to be registered on the server machine thus breaking the interdependency in the development phase. Process variables like real-time sensor data and robot-states are relayed to the client-side using implicit inter-component communication. In our framework, we are using the .NET Remoting for calling functions remotely, and hides the details of network transfers.

C# .NET is a very good object-oriented language that lets the programmer to easily create programs for their functions. Programming languages of the form of C and C++ can also be used in C#. C# solves many problems from C++, such as a garbage collector (to solve memory problems). In addition to the Garbage Collector and the use of references, in some cases, unsafe code can also be used to directly access the memory. Also, variables in C# are automatically initialized by the environment, and are type safe. Thus, C# is becoming a good choice for creating components-from high-level business objects to system-level applications.

JAVA has also some high level language features like garbage collection. But although very cool, it is an interpreted language. JAVA and CORBA are intended to

be cross-platform environments thus requiring lot of JIT (just-in-time) compilation and virtual machines to interpret code on different operating systems. Although JIT compilation gets faster and more grunt work is handed off to the APIs, JAVA is still a very young language and still going through a lot of changes [55]. On the other hand, in the .NET, IL (Intermediate Language) code is compiled by JIT compiler to native machine code prior to execution. Compiled IL code executes on top of a portable API that enables future platform independence. Besides, the .NET has embedded type signatures which allows component debugging across different languages which is a missing feature in JAVA and CORBA. In addition, they provide no support for hardware-accelerated graphics APIs that are critical for live video visualization on PCs.

5.4.2 Comparing Direct3D with Other Graphics API

The main distinct feature of Direct3D API of DirectX is its capability to provide the developers more directly access the hardware features of a computer i.e. it allows to utilize the hardware acceleration [57]. In computing, hardware acceleration is the use of hardware (here video card or graphics card) to perform some functions faster than is possible in software running on the normal CPU. Examples of hardware acceleration include blitting (physically copying the contents of one image on top of the another image) acceleration functionality in graphics processing units (GPUs)

and instructions for complex operations in CPUs. Using this hardware acceleration is a wonderful thing for AR simulation, because we can go from dozens of frames a second to hundreds of frames drawn per second. We can even go to almost a thousand fps unoffending on hardware capabilities.

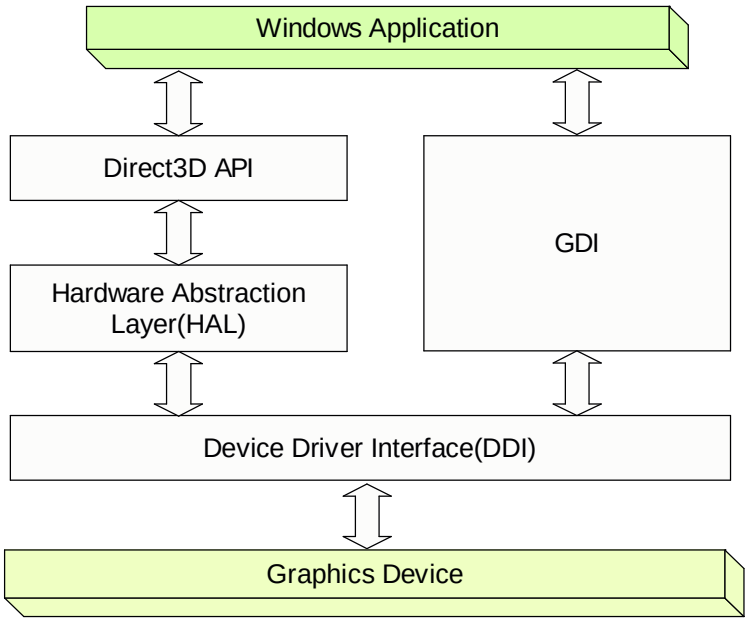

Figure 5.2: Dirct3D interaction to hardware.

The hardware acceleration capability of Direct3D is achieved through its utilization of the *HAL*. The *HAL* is a device-specific interface, provided by the device manufacturer. As shown in Fig. 5.2, *HAL* layer wraps around the *DDI (Device Driver Interface)*, abstracting it away and directly communicating to the graphics card (or graphics accelerator), more specifically to the GPU. A GPU is the microprocessor of a graphics card for a personal computer or game console. Modern

GPU's are very efficient at manipulating and displaying computer graphics, and their highly-parallel structure makes them more effective than typical CPUs for a range of complex algorithms. A GPU implements a number of graphics primitive operations in a way that makes running them much faster than drawing directly to the screen with the host CPU. GPU accelerates the memory intensive work of texture mapping and rendering polygons, and also accelerates geometry calculations such as mapping vertex into different coordinate systems. *HAL* takes requests for example, to *blit* surfaces and converts those into hardware instructions for the GPU to perform. The use of *HAL* also guarantees increased stability and portability of *DirectX* application.

Although it is not possible to use hardware feature that *Direct3D* does not support unlike OpenGL, it is updated once every two months allowing a standard interface to new features sooner than they appear in core OpenGL. Whereas, OpenGL has an extension mechanism allowing immediate access to new features as graphics manufacturers create them, rather than having to wait for a new API version to appear.

Direct3D is not an easy-to-code API. It is very difficult to use as it has large code overhead in comparison to OpenGL. But it is good for doing lower level operations. The vast majority of PC games are written using Direct3D. On the other hand, OpenGL has smaller code size, but it is more difficult for doing lower level operations

than Direct3D. Java3D has similar code overhead as OpenGL.

In consideration to multi-Platform support, *Direct3D* is very poor as it used only on only Windows, XBox, XBox 360 and foundation for *XNA*. But OpenGL can be used on Windows, Linux, MacOS, Gamecube, Playstation 3 (OpenGL ES subset), Silicon Graphics Workstations, BeOS and others. On the contrary, Java3D is platform independent.

Difference between Direct3D and Windows GDI+ as graphics programming tool is as follows. GDI+ is a technology to draw. It's a native 2D graphics library for working in windows. It is easy to work with as it has built in function for rounded rectangles, ellipses, circles, lines, n-shaped polygons and so forth. We can use Windows GDI+ to draw onto a Direct3D surface. But, it doesn't use any extended graphics or acceleration features, even when there is a hardware acceleration present; See Fig. 5.2. Therefore, it is so slow that it is better to write own optimized rectangle, line or circle functions oneself rather than resorting to the monstrous GDI+ [55]. It was actually designed to work on a myriad of different configurations, different resolutions, different bit depths, but not designed to be fast. Another benefit of GDI+ is that it is more easily ported to other devices (like Pocket PC). But this is not important in our telerobotic implementation.

The differences among the 3D graphics API's discussed so far can be summarized as illustrated in Table 5.5.

Table 5.5: Comparison between important 3D graphics APIs.

Features	Direct3D	OpenGL	Java3D
Speed	Excellent speed through Graphics acceleration	Debatable (little difference with Direct3D)	Slow
Ease of coding	large code overhead, but good for lower level operations	Small Code size,worse in lower level operations	Similar to OpenGL
Multi-platform support	Used on Windows, XBox, foundation for XNA	Used on Windows, Linux, MacOS etc	Platform independent
Ease of drawing text	Good	Poor in base standard - one has to use bitmaps. Good with GLUT	Fair
Ease of running in a window	Excellent	Poor	Not really applicable
Immediate mode	No	Yes	
Ease of learning	Hard to learn	easier to learn	Fair
Helper library	Extensive	Smaller	Fair

5.4.3 Comparing Our Approach to Others

Iqbal, A. [1] augmented his stereo-vision system by overlaying only a small red ball at the current position of the gripper in comparison to our whole graphical arm overlayed on the real video image. He used Faugeras [58] calibration method with affine frame of reference proposed by Kuno et. al. [44] which led him to noticeable error in the superimposition. Whereas, use of our computer vision-based calibration reduces error up to 1/50 of pixel size.

In [45], a high level web-based AR as well as VR user interface is developed using Java and CORBA. Hu et. al. [59] also proposed JAVA for network interfacing and video as well as the use of C++ for the robot controller for Internet-based telerobotic System. But in our system, .NET framework is directly used for all GUI development as well as the core system components making it a unified solution. This frees us from using middleware services like MS VM within the framework. JAVA and CORBA are intended to be cross-platform environments thus requiring lot of JIT compilation and virtual machines to interpret code on different operating systems.

In [1], GDI functionalities are used for drawing purposes. In [45], JAVA3D and in [60] OpenGL is used as 3D graphics API. These APIs are easy to work with but they provide no or little support for hardware-accelerated graphics APIs that are critical for live video visualization on PCs. Therefore, graphics rendering of our system is faster than these system for our use of Direct3D which uses hardware acceleration through HAL.

In [45], operation time reported for program launching, image acquisition, graphic robot initialization and robot movement as 5.06, 0.20, 3.32 and 2.29 seconds respectively. In our case, image acquisition time over a LAN is 59 ms (83 ms required from reading image from cameras to the display on HMD).

J. Vallino [61] reports in his PhD dissertation, refresh rate of 10 fps to be required

for AR. We have got above 11-17 fps after overlaying graphical arm with live stereo video and around 250 fps with local simulation.

Our system is comparatively cost effective due to the use of commodity hardware (PC) and software.

Chapter 6

Conclusion

In this chapter, a summary of the proposed work is depicted with outlines of the major contributions made. A number of recommendations and research directions are also mentioned.

6.1 Summary of the Work

A hierarchical design strategy and its implementation for augmenting a telerobotic stereo vision system comprising of a PUMA-560 industrial robot operating over a campus LAN is described in this work. At first, the geometric model of a six degree of freedom (DOF) robot arm is developed and based on that a three-dimensional (3D) graphical arm is modeled. Then, a computer vision based calibrations method and a registration method are used to superimpose graphics on real image creating

a simulation plan. Hardware accelerated graphics rendition is used through MS Direct3D API. A flexible and generalized data structure suitable for telerobotic visualization is used. Motion activation algorithms are developed and user-friendly graphical user interfaces are designed for facilitating telerobotic task simulation prior to give the real command to the telerobot.

6.2 Contributions

A brief account of the contributions made through this thesis work is given below:

1. Providing a simulation plan by overlaying graphic robot on real video for making rehearsal and correction before going for actual teleoperation and thus providing task safety.

2. Saving bandwidth by sending (less frequently) only the finalized or planned data.

3. Using hardware accelerated graphics rendition that provides us with excellent refresh rate of the output screen.

4. Improvement in accuracy of execution by using better calibration method giving accuracy up to 1/50 of pixel size.

5. Flexible and generalized data structure suitable for telerobotic visualization.

6. User-friendly graphical user interface for simple manipulation in the telerobotic AR system.

7. Use of low cost and commercially available hardware and software

8. Can be used as a cost effective and flexible visual tool for showing robot manipulation in the classrooms.

9. Can be used as a base framework for further research on virtual and augmented telerobotic manipulations.

6.3 Future Research Directions

Our future research work based on this thesis can be directed to the following areas:

1. Providing an intelligent system to switch between VR and AR modes of operation based on network delays to ensure QoS.

2. Using multi-processor system for processing video and graphics data more efficiently.

3. Developing a full fledged educational graphical tool by extending the VR manipulation sub-system to support all robot manipulations needed to be demonstrated in the classroom.

4. We will attach force feedback information to the graphical robot and use a multiplexer to provide options for using master arm in graphical arm manipulation as well as real robot manipulation. Thus, we will be able to reduce the effect of time delay even in force feedback.

5. Using a task-aware video compression techniques for compressing background data and transfer of uncompressed, small volume, higher resolution, region-focused, video data that is essential for the current task. The operator may specify a relevant tool region to be dynamically tracked and transmitted with higher resolution and refreshing rate. A tracking algorithm detects motion in the relevant region and guides a selective compression algorithm. Thus, by controlling the size of the uncompressed region and its resolution the user may set up a variety of scenarios between the above two extremes.

6. Using a resilient telerobotic flow control to ensure smooth performance degradation under severe load conditions. One approach is a flow-control that activates remote emergency agents, at the slave site, to ensure task safety and continuity under excessive delays. At the client station, the virtual environment may supply the operator station with kinesthetic and visual feedback to provide interaction continuity based on task locality, environment model, and history information.

7. Using commercial software available to extract exact 3D model of the workspace

objects which will facilitate more accurate task manipulation.

Bibliography

[1] A. Iqbal. Multistream Realtime control of a Distributed Telerobotic System. *M.Sc. Thesis, King Fahd University of Petroeum and Minerals*, June 2003.

[2] J.-Y. Herve, C. Duchesne, and V. Pradines. Dynamic registration for augmented reality in telerobotics applications. *IEEE International Conference on Systems, Man, and Cyberneticsn*, Vol.2:1348 – 1353, October 2000.

[3] A. Rastogi. Design of an Interface for Teleoperation in Unstructured Environments Using Augmented Reality. *M.A.Sc. Thesis, Department of Industrial Engineering, University of Toronto, Toronto*, 1996.

[4] Kris Gray. *Microsoft DirectX9 Programmable Graphics Pipeline.* Microsoft Press, 2003.

[5] A. Rastogi, P. Milgram, and D. Drascic. Telerobotic Control with Stereoscopic Augmented Reality. *SPIE*, Vol.2653: Stereoscopic Displays and Virtual Reality Systems III:135–146, Feb. 1996.

[6] Ronald T. Azuma. A Survey of Augmented Reality. *Presence: Teleopertaors and Virtual Environments*, Vol. 6(4):355–385, 1997.

[7] P. Milgram, A. Rastogi, and J. Grodski. Telerobotic Control Using Augmented Reality. *IEEE Inter. Workshop on Robot and Human Communication*, pages 21–29, 1995.

[8] R. Azuma, Y. Baillot, S. Behringer, R.and Feiner, S. Julier, and B. MacIntyre. A Survey of Augmented Reality. *Computer Graphics and Applications, IEEE*, 21(6):34–47, Nov.-Dec. 2001.

[9] Grigore Burdea. Invited Review: The Synergy between Virtual Reality and Robotics. *IEEE Transactions on obotics and Automation*, Vol.15, No. 3:400 410, June 1999.

[10] J. Abdullah and K. Martinez. Camera Self-calibration for the ARToolKit. *The First IEEE International Workshop on Augmented Reality Toolkit*, page 5, Sept. 2002.

[11] Z. Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.22, No. 11:1330 – 1334, Nov. 2000.

[12] Richard L. Holloway. Registeration Errors in Augmented Reality. *Ph.D. Dissertation, University of North Carolina at Chapel Hill*, 1995.

[13] Michael Deering. High resolution virtual reality. *Proc. of the 19th annual conference on Computer graphics and interactive techniques*, Vol.26(2):195–202, 1992.

[14] A. Janin, D. Mizell, and T. Caudell. Calibration of Head-Mounted Displays for Augmented Reality. *Proc. of IEE VRAIS '93*, pages 246–255, 1993.

[15] M. M. Wloka and B. G. Anderson. Resolving Occlusion in Augmented Reality. *Proc. of 1995 Symposium on Interactive 3D Graphics*, pages 5–12, 1995.

[16] D. Hearn and M. P. Baker. *Computer Graphics C Version*. Prentice Hall, 2nd edition, 1997.

[17] Xi Ning and T.J. Tarn. Action synchronization and control of internet based telerobotic systems. *Proc. of IEEE Inter. Conf. on Robotics and Automation*, Vol. 1:219–224, 1999.

[18] L. M. Strunk and T. Iwamoto. A linearly-mapping stereoscopic visual interface for teleoperation. *IEEE International Workshop on Intelligent Robots and Systems, IROS' 90*, pages 429–436, 1990.

[19] http://www.3dgw.com/Articles/articlepage.php3.

[20] G. Lee, S. Bekey and A. Bejczy. Computer control of space teleoperators with sensory feedback. *Proc IEEE Int. Conf. on Robotics and Automation*, pages 205–214, 1985.

[21] www.stereo3d.com/nuview.htm.

[22] www.demensional.com/manuals/eye3d 3-in-1 user's manula (e).pdf.

[23] S. Lee, S. Lakshmanan, S. Ro, J. Park, and C. Lee. Optimal 3D Viewing with Adaptive Stereo Displays for Advanced Telemanipulation. *International Conference on Intelligent Robots and Systems*, pages 1007–1014, 1996.

[24] S. Lee, S. Ro, J. Park, and C. Lee. Optimal 3D Viewing with Adaptive Stereo Displays: A Case of Tilted Camera Configuration. *ICAR '97*, pages 839–844, 1997.

[25] D. B. Diner and D. H. Fender. Human engineering in stereoscopic viewing devices. *Jet Propulsion Laboratory report (JPL D-8186)*, 1991.

[26] R. L. Pepper, R.E. Cole, and E. H. Spain. The influence of camera separation and head movement on perceptual performance under direct and tv displayed conditions. *Proceedings of the Society for Information Display*, pages 73–80, 1996.

[27] H. Loaiza, J. Triboulet, S. Lelandais, F. Chavand, and F. Artigue. A multi-configuration stereoscopic vision system for domestic mobile robot localization. *Proc. of the First Workshop on Robot Motion and Control, 1999. RoMoCo '99*, Vol. 5, No. 3:207 – 212, June 28-29 1999.

[28] M.N. Armstrong. Self-Calibration from Image Sequences. *PhD Dissertation, Department of Engineering Science, University of Oxford*, 1996.

[29] M. Malik, S. Mudar, and C. Florent. Automatic camera calibration based on robot calibration. *Proc. of IEEE Instrumentation and Measurement Technology Conference, 1999*, Vol. 2:1278 – 1282, 1999.

[30] J. Heikkila. Geometric camera calibration using circular control points. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.22, No. 10:1066 – 1077, Oct. 2000.

[31] N. Hollinghurst; R. Cipolla. Human-robot interface by pointing with uncalibrated stereo vision. *Image and Vision Computing*, 14(3):171–178, 1996.

[32] Zhengyou Zhang. Flexible Camera Calibration by Viewing a Plane from Unknown Orientations. *Proc. of ICCV99*, 1999.

[33] Y. Kuno, M. Sakamoto, K. Sakata, and Y. Shirai. Vision-based human interface with user-centered frame. *Proceedings of the IROS'94*, 3:2023–2029, 1994.

[34] R. Willson. Modeling and Calibration of Zoom Lenses. *PhD Dessertation, Carnegie Mellon University, Pittsburg, PA*, January 1994.

[35] W. Kim. Computer vision assisted virtual reality calibration. *IEEE T-RA*, pages 450–464, June 1999.

[36] *http://www.sop.inria.fr/robotvis/personnel/zzhang/sof twares.html*.

[37] F. Dornaika and R. Chung. An Algebraic Approach to Camera Self-Calibration. *Computer Vision and Image Understanding*, Vol. 83, No. 3:195–215, Sept. 2001.

[38] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analaysis and Machine Vision*. PWS Publishing, 2nd edition, 1999.

[39] The free encyclopedia, Wikipedia. 3D computer graphics. *http://en.wikipedia.org/wiki/3D_computer_graphics*.

[40] P. Milgram, S. Zhai, and D. Drascic. Applications of Augmented Reality for Human-robot Communication. *Proc. of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems, Yokohama, Japan*, July 26-30, 1993.

[41] J. Gu, E. Augirre, and P. Cohen. An augmented-reality interface for telerobotic applications. *Proc. of Sixth IEEE Workshop on Applications of Computer Vision,2002(WACV 2002)*, pages 220–224, Dec. 2002.

[42] C. Dushesne and J-H. Herv. A point-based approach to the interposition problem in augmented reality. *Proc. International Conference on Pattern Recognition,Barcelona, Spain*, pages 220–224, Sept. 2000.

[43] O.D. Faugeras and G. Toscani. The Calibration Problem for Stereo. *Proceedings of Conference on Computer Vision and Pattern Recognition, Miami Beach, FL*, Vol. 5, No. 3, June, 15-20 1986.

[44] Y. Kuno, K. Hayashi, K.H. Jo, and Y. Shirai. Human-robot interface using uncalibrated stereo vision. *International Conference on Intelligent Robots and Systems 95*, 1:525–530, 1995.

[45] R. Marin, P.J. Sanz, and J.S. Sanchez. A very high level interface to teleoperate a robot via Web including augmented reality. *Proc. IEEE International Conference on Robotics and Automation, 2002 ICRA '02*, Vol.3:2725 – 2730, May 2002.

[46] Mayez Al-Mouhamed, Onur Toker, and Nesar Merah. Design of an Intelligent Telerobotic System. *Technical Report AT-20-80, King Abdulaziz City For Science and Technology, Kingdom of Saudi Arabia*, 2004.

[47] I Beiderman. Human image understanding: Recent research and theory. *Computer Vision, Graphics and Image Processing*, Vol. 32:27–73, 1985.

[48] R.A. Browse and S. Little. The effectiveness of real-time graphic simulation in telerobotics. *Proc. of IEEE International Conference of Systems, Man, and Cybernetics, Charlottesville, Virginia*, pages 895–898, Oct 1991.

[49] G. Litern, K. E. Thomley-Yates, B.E. Nelson, and S.N. Roscoe. Content, variety, and augmentation of simulated visual scenes for teaching air-to-ground attach. *Human Factors*, Vol. 29(1):45–59, 1987.

[50] M. Al-Mouhamed, Onur Toker, and Asif Iqbal. A Multi-Threaded Distributed Telerobotic Framework. *the IEEE/ASME Transactions on Mechatronics(accepted).*

[51] Jean-Yvesw Bouguet. Camera Calibration Toolbox for Matlab. August 10, 2005.

[52] S. R. Hollasch. Four-Space Visualization of 4D Objects. *MS Thesis, Arizona State University*, August 1991.

[53] Microsoft. MSDN Library. *http://msdn.microsoft.com/default.asp.*

[54] F. Wolfgang and A. Geva. *Beginning Direct3D Game Programming.* Prima Tech, ed. Andre LaMother, 2001.

[55] Peter Walsh. *Advanced 3D Game Programming with DirectX 9.0.* Wordware Publishing, Inc., 2003.

[56] M. Al-Mouhamed, O. Toker, A. Iqbal, and Syed M.S. Islam. Evaluation of real-time delays for networked telerobotics. *Proc. of the 3rd International IEEE Conference on Industrial Informatics INDIN05, Perth, Western Australia*, August 10-12 2005.

[57] David Weller, Alexandre Santos Lobao, and Ellen hatton. *Microsoft DirectX9 Programmable Graphics Pipeline*. Apress, 2004.

[58] O.D. Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? *In Proc. 2nd European Conference on Computer Vision, LNCS 588, Springer-Verlag*, pages 563–578, 1992.

[59] Huosheng Hu, Lixiang Yu, Pui Wo Tsui, and Quan Zhou. Internet-based robotic systems for teleoperation. *International Journal of Assembly Automation*, 21(2), 2001.

[60] S.R. Gomez, J. Carro, E. Moreda, J.J. Escribano, and C. Cerrada. A Design Srategy for Low Cost Experimental Telerobotics Platforms. *Proc. IEEE International Workshop on Robot and Human Interaction, Pisa, Italy*, pages 273 – 278, Sept. 1999.

[61] J. Vallino. Interactive Augmented Reality. *PhD Dissertation, Department of Computer Science, University of Rochester, Rochester, NY*, April 1998.

Vita

Syed Mohammed Shamsul Islam was born in Gazipur, Bangladesh on February 28, 1979. He received his Bachelors Degree in Electrical and Electronic Engineering with Honors in November, 2000 from Islamic Institute of Technology, IIT (which is now renamed as Islamic University of Technology, IUT), Board-bazar, Gazipur, a subsidiary organ of the Organization of Islamic Conference (OIC). He also completed Post Graduate Diploma in Technical Education from the same university in 2002. He joined Integrated Control Equipment Ltd., Dhaka in 2000 as a Project Engineer. He also worked as a Lecturer in Computer Science in Manarat Dhaka International College. He joined Asian University of Bangladesh, Dhaka as a Lecturer-cum-Programmer in the Department of Computer Science and Engineering in July, 2001. With study-leave from there, he joined the Department of Computer Engineering, King Fahd University of Petroleum and Minerals (KFUPM), as a Research Assistant in March 1, 2003. He received the Master of Science degree in Computer Engineering from KFUPM in December, 2005.