

MOCAP MOTION CAPTURE



By

**AMAD JUNAID
MOHAMMAD ALI
AZKA AMIN
QASIM KHAN**

**Submitted to the Faculty of Computer Science, Military College of Signals,
National University of Sciences and Technology, Rawalpindi,
in partial fulfillment for the requirement of a
B.E. Degree in Software Engineering.
August 2010**

CERTIFICATE

Certified that the contents and form of project report entitled “**MOCAP (Motion Capture)**” submitted by 1) NC Amad Junaid 2) NC Mohammad Ali 3) NC Azka Amin, 4) NC Qasim Khan have been found satisfactory for the requirement of the degree.

Supervisor: _____

Col Naveed Sarfaraz Khattak

Abstract

The Human Motion Capture or MOCAP are terms used to describe the capturing of actual human motion and then translating it on to CG models. In simple words, Human Motion is captured through multiple cameras in a controlled environment and then this very motion is transferred onto 3D models. The world of Human Motion Detection has progressed and done wonders. The systems available in the market are complex and very expensive as they are dependent on hardware requirement. Most of these systems range into the tens of thousands of dollars. Our aim is not to reinvent the wheel, rather to create a cheaper system, which laid emphasis on the systems software to compensate for the hefty hardware requirements.

The Project is divided in to two parts, i) Motion Detection ii) 3d Scene Reconstruction.

For motion detection, we used two to three cameras, and tracked the motion of an actor, in a controlled environment: The actor is to wear white markers on black apparel in front of a black background. This was achieved using the techniques of neighborhood matching and region growing. The reconstruction process is subdivided into two parts: first, calibration of cameras being used; second, computing the points in space that project to the tracked image points. Calibration is done by using a chessboard pattern, corner identification, square count and then relating the real world units to the computer system units. The tracked sequence of points is used to reconstruct corresponding set of points in 3d space representing the object surfaces on the scene using method of stereoscopic reconstruction and linear triangulation.

Our system captures the motion and builds the 3d representation in **real time**. In addition to being much less expensive our system is also very **portable**. We can setup our studio anywhere we want which satisfies the goal of Nex Gen gaming console as well. In comparison to other similar projects such as Microsoft Kinect and PS3 motion controller which employ different kinds of special sensors and cameras, our system equals in reconstructing the motion but in overall performance lags by a few milliseconds as video capture rate by our simple webcams is only 15 fps.

Table of Contents

1. Introduction	1
1.1. Introduction.....	1
1.2. Scope and Objective.....	2
1.3. Conceptual Framework of MOCAP.....	2
2. Related Work	4
2.1. History of Motion Capture.....	4
2.1.1. Early Attempts.....	4
2.1.2. Advent of Digital Motion Capture.....	6
2.2. Types of Motion Capture.....	7
2.2.1. Mechanical.....	8
2.2.2. Optical.....	8
2.2.3. Electromagnetic (magnetic).....	9
3. Requirement Analysis	11
3.1. Introduction.....	11
3.2. System Overview.....	11
3.3. Assumptions and Dependencies.....	12
3.4. Operating Environment.....	12
3.5. Product Features.....	13
3.6. User Problem Statement.....	13

3.7. Specific Requirements.....	14
3.7.1. Functional Requirements.....	14
3.7.2. Non-Functional Requirements.....	15
3.8. Target Environment.....	16
3.9. System Requirements.....	16
3.9.1. Hardware Requirements.....	16
3.9.2. Software Requirements.....	16
4. System Design and Architecture.....	17
4.1. Introduction.....	17
4.2. System Overview.....	17
4.3. System Architecture.....	18
4.4. Decomposition Description.....	20
4.4.1. Motion Tracker.....	20
4.4.2. Motion Analyzer.....	20
4.4.3. Motion Modeler.....	20
4.5. Data Design and Data Description.....	20
4.6. External Interface Requirements.....	23
4.6.1. Hardware Interfaces.....	23
4.6.2. Software Interfaces.....	23
5. System Implementation.....	24
5.1. Introduction.....	24
5.2. Calibration ToolBox.....	25
5.2.1. Overview.....	25

5.2.2. Detailed Explanation.....	25
5.3. Client.....	29
5.3.1. Overview.....	29
5.3.2. Detailed Explanation.....	30
5.4. Server.....	36
5.4.1. Overview.....	37
5.4.2. Detailed Explanation.....	37
6. Comparative Analysis.....	42
6.1. Introduction.....	42
6.2. Test Results.....	42
6.2.1. Introduction.....	42
6.2.2. Unit Testing.....	43
6.2.2.1. Tracking.....	43
6.2.2.2. 3D Reconstruction and Rendering.....	44
6.2.3. Integration and System Testing.....	45
6.2.3.1. Data Transfer.....	45
6.2.3.2. Rendering.....	46
6.3. Systems to be compared.....	48
6.3.1. Optical Motion Capture by Phase Space.....	48
6.3.2. Optotrak Certus Motion Capture System.....	48
6.3.3. Vicon – Motion Capture Services.....	48
6.4. Comparison.....	49
6.5. Conclusion.....	50
7. Conclusion.....	51

7.1. Introduction.....	51
7.2. Conclusion.....	51
7.3. Future Work.....	52
8. Bibliography.....	54

List of Figures

2.1 Mahomet Running, Eadweard Muybridge, 1879.....	5
2.2 Advent of Digital Motion Capture.....	6
4.1 The Architectural Framework of MOCAP.....	19
4.2 Data Flow Diagram at Client Side.....	21
4.3 Data Flow Diagram at Server Side	22
5.1 Calibration ToolBox.....	25
5.2 Images Read.....	26
5.3 Clicking on Corners.....	27
5.4 Corners Extracted(Red).....	27
5.5 Calibration Results.....	27
5.6 Extrinsic Parameter Image.....	28
5.7 Extrinsic Parameters.....	29
5.8 Client Module.....	30
5.9 Reading Calibration Data.....	31
5.10 Projection Matrix.....	31
5.11 Communications Between Client and Server.....	32
5.12 Acquired Video on Right and Original on Left.....	34

5.13 Feature Initialization.....	35
5.14 Searching.....	36
5.15 Labeling.....	36
5.16 Multi Threading Structure.....	37
5.17 Skeletal Model Moving According to Tracked Data.....	40
6.1 Testing Path.....	42
6.3 Tracking Initialization.....	42
6.4 5 Seconds into the video.....	42
6.5 Tracking Initialization.....	43
6.6 2 Seconds into the video.....	4.3
6.7 Motion Representation is Accurate Enough.....	44
6.8 Calibration Data in Client(up) and Server(Lower).....	45
6.9 Figure Shows Server Allowing Only One Client to Initialize.....	46
6.10 Environment Used While Testing.....	47
6.11 Motion Being Generated in Real time.....	47

List of Tables

6.1 Comparative Analysis	49
--------------------------------	----

INTRODUCTION

1.1 Introduction

Motion Capture MOCAP is sampling and recording motion of humans, animals and inanimate objects as 3D data. The data can be used to study motion or to give an illusion of life to 3D computer models. Since most of applications today require special equipment there are still a limited number of companies that are utilizing MOCAP technology. Most people even the children have seen the films and games etc for which MOCAP technology is used. In that sense, MOCAP is in everyday life.

Motion Capture MOCAP is in its first release by the students of NUST. The objective is to capture human motion, understand it and translate it into 3D Graphical models using Computer Vision and Computer Graphics. A lot of work has been done in the field of capturing human motion with great dependence on hardware requirements and with less emphasis on the software. The purpose of this project is not to reinvent the wheel but rather to go about using well developed software to compensate for the systems hardware requirements which are quiet expensive for the ordinary user and market. The overall system is quite compact with three to four cameras which can be setup easily.

1.2 Scope and Objective

MOCAP is a project on which considerable amount of work has been done. The challenge taken up in this project by the team is to reduce the cost that comes with the expensive hardware with the usual MOCAP technology. The MOCAP technology in market uses expensive hardware e.g. the use of Hi-tech digital cameras, usually using 12 cameras at a time. This expenditure makes it hard for the normal market to work with MOCAP technology. Emphasis is laid on the underlying software in this project, with less dependence on the hardware.

The aim is to capture general human motion, simply capturing the basic joint movements and working progressively from there on. The captured motion would then be translated on graphical models.

1.3 Conceptual Framework of MOCAP

The basic concept of MOCAP was to build a system based on client-server architecture where the cameras serve as clients and the image processing is done at the server side. System will use 3 to 4 web cameras of resolution 320x240 to capture the motion of the actor.

Camera calibration will be done by the user for each camera. User will calibrate the cameras using a chessboard pattern for each client by taking snaps of the pattern (20 from each cam using own software) and give the system path of the pictures. System will then calculate the intrinsic parameters. The user will then hold the pattern to take

one photo from each cam so that pattern is visible in each cam for extrinsic parameter calculation.

To start tracking user must initialize tracking on each camera by clicking on each trackable point first and then tell the program to start tracking. Mathematical relationships between motion of different human body parts like angles and translational values should be calculated from tracked data so as to build a smooth system.

Calibration will be used to calculate camera projection matrices. Calibration will give 1 intrinsic matrix ($K_{3 \times 4}$) and 2 extrinsic matrices (rotation: $R_{3 \times 3}$, translation: $T_{3 \times 1}$). These will be used to calculate camera projection matrices as $P = K \times [R|T]$ ('x' means matrix multiplication). Use data from tracking and calibration to reconstruct points in 3D.

The points in 3d space will be computed as: Suppose two cameras that see a point has projection matrices P and P' . In each image there is this measurement $x = PX$, $x' = P'X$, where x and x' are 2d coordinates found from tracking and X is the 3D coordinates. These equations will be combined into a form $AX = 0$, which is a linear equation in X . It will be solved by *SVD*. Thus the 3d point in space that project to these two image points will be computed.

The scene will be updated as per the received track data from the tracking clients so as to build smooth motion transition.

RELATED WORK

2.1 History of Motion Capture:

The development of modern day MOCAP technology has been led by the medical science, army and computer generated imagery (CGI) field where it is used for a wide variety of purposes. It seems that MOCAP technology could not exist without the computer. However, there were early successful attempts to capture motion long before the computer technology became available. [1]

2.1.1 Early Attempts

Eadweard Muybridge (1830 –1904) was born in England and became a popular landscape photographer in San Francisco. It is said that in 1872 Leland Stanford (California governor, president of the Central Pacific Railroad, and founder of Stanford University) hired Muybridge to settle a \$25,000 bet on whether all four feet of a horse leave the ground simultaneously or not. Six years later Muybridge proved that in fact all four feet of a trotting horse simultaneously get off the ground. He did so by capturing a horse's movement in a sequence of photographs taken with a set of one dozen cameras triggered by the horse's feet. Muybridge invented the zoopraxiscope, which projects sequential images on disks in rapid succession, in 1879. The zoopraxiscope is considered to be one of the earliest motion picture devices. Muybridge perfected his technology for sequential photographs and published his photographs of athletes, children, himself, and animals. His books, *Animals in Motion* (1899) and *The Human*

Figures in Motion (1901), are still used by many artists, such as animators, cartoonists, illustrators, and painters, as valuable references. Muybridge, who had a colorful career and bitter personal life, is certainly a pioneer of MOCAP and motion pictures. Born in France, in the same year as Muybridge, was Etienne-Jules Marey. Marey was a physiologist and the inventor of a portable sphygmograph, an instrument that records the pulse and blood pressure graphically. Modified versions of his instrument are still used today.



Figure 2.1 Mahomet Running, *Eadweard Muybridge, 1879*

In 1882 Marey met Muybridge in Paris and in the following year, inspired by Muybridge's work, he invented the chronophotographic gun to record animal locomotion but quickly abandoned it. In the same year he invented a chronophotographic fixed-plate camera with a timed shutter that allowed him to expose multiple images (sequential images of a movement) on a plate. A sample result is shown

in figure 2.1. The camera initially captured images on glass plates but later he replaced glass plates with paper film, introducing the use of film strips into motion picture.

2.1.2 Advent of Digital Motion Capture:

In Figure 2.2, an example of digital motion capture is shown.



Figure 2.2 Advent of Digital Motion Capture

In digital motion capture live motion is captured (upper left corner) as shown in the diagram and the motion is translated on a graphical model as on the lower right side of the figure 2.2.

Research and development of digital MOCAP technology started in pursuit of medical and military applications in the 1970s. The CGI industry discovered the technology's

potentials in the 1980s. Since some of this book's readers weren't born in the 1980s, let's recall the 1980s. In the 1980s there were floppy disks that were actually floppy and most computers were equipped with monochrome monitors; some with calligraphic displays. To view color images, for example rendered animation frames, images had to be sent to a "frame buffer," which was often shared by multiple users due to its cost. Large computers were housed in ice cold server rooms. The noise of dot matrix printers filled offices. Ray-tracing and radiosity algorithms were published in the 1980s. Renderers based on these algorithms required a supercomputer or workstations to render animation frames in a reasonable amount of time. Personal computers weren't powerful enough. (Ray-tracing and radiosity didn't become widely available until the computing power improved.) CPUs, memories, storage devices, and applications were more expensive than today. Wavefront Technologies developed and marketed the first commercial off-the shelf 3D computer animation software in 1985. Only a handful of computer animation production companies existed. Most of the animations that they produced were "flying logos" for TV commercials or TV program's opening sequences. These were often 15 to 30 seconds long per piece. The readers who saw "Brilliance" (also called "Sexy Robot") in the 1980s probably still remember the astonishment of seeing a computer generated character, a shiny female robot, moving like a real human being.

2.2 Types of Motion Capture

It is sometimes suggested that the roots of motion capture can be seen in the motion studies of Eadweard Muybridge and Etienne Jules Marey. In the form thought today,

MOCAP technology has been developing since the 1970s, when it was created for military use, and has been used in entertainment since the mid-1980s[2]. Over the years, MOCAP has taken many forms, each with its own strengths and weaknesses. Following is a summary of three types of MOCAP used in entertainment and the ways in which they work. Examples and more description of all these types of motion capture systems are available on the website at La Trobe University, "Introduction to Motion Capture in Music." [3] [4]

2.2.1. Mechanical

Performer wears a human-shaped set of straight metal pieces (like a very basic skeleton) that is hooked onto the performer's back; as the performer moves, this exoskeleton is forced to move as well and sensors in each joint feel the rotations. Other types of mechanical motion capture involve gloves, mechanical arms, or articulated models (like Monkey), which are used for 'key framing'. Its advantage is that there is no interference from light or magnetic fields. It has certain disadvantages such as: The technology has no awareness of ground, so there can be no jumping, plus feet data tends to slide, equipment must be calibrated often, unless there is some other type of sensor in place, it does not know which way the performer's body is pointing and absolute positions are not known but are calculated from the rotations.

2.2.2 Optical

Performer wears reflective dots that are followed by several cameras and the information is triangulated between them. Markers are either reflective, such as a system

manufactured by Vicon or Motion Analysis, or infra-red emitting, many of which have been developed for musical applications (such as conducting).It was developed primarily for biomedical applications (sports injuries, analysis of athletic performance, etc.).Its advantages include performer feels free to move due to no cables connecting body to the equipment, larger volumes are possible, more performers are possible and very clean, detailed data. Disadvantages include: It is prone to light interference, reflective dots can be blocked by performers or other structures, causing loss of data, or occlusion-this can be compensated for with software which estimates the position of a missing dot, rotations of body parts must be solved for and are not absolute, performer must wear a suit with dots and balls (20-30 for body, in 1995), which may be uncomfortable, information has to be post-processed or 'tracked' before viewing so performer cannot see his or her image and so cannot be as creative or identify potential problems (a hand hitting a giant nose, for example),higher cost than magnetic (a cost of US\$150,000 to 250,000 in 1995),tracking can take 1-2 minutes per captured second for straightforward data (complicated can take 15-30 minutes per second, according to 1995 data).

2.2.3 Electromagnetic (magnetic)

Performer wears an array of magnetic receivers which track location with respect to a static magnetic transmitter. One of the first uses was for the military, to track head movements of pilots. Often this type of motion capture is layered with animation from other input devices. The two main manufacturers of this type of motion capture equipment are Polhemus and Ascension. Advantages include: Positions are absolute,

rotations are measure absolutely; orientation in space can be determined, which is very useful, can be real-time, which allows immediate broadcast as well as the opportunity for performers to puppeteer themselves with instantaneous feedback (more spontaneity in the performance), relatively cheaper than optical (1995 price under US\$40,000 for a typical system). Disadvantages include: Magnetic distortion occurs as distance increases, data can be noisy - it's not as good as optical, prone to interference from magnetic fields - cement floors usually contain metal, so stages must be built; performers wear cables connecting them to a computer, which limits their freedom, and in 1995, sampling speed too low for many sports applications.

REQUIREMENT ANALYSIS

3.1 Introduction

This section of the document contains a structure for a software requirements specification (SRS) document. It describes the services and functions which the system should provide, the constraints under which the system must operate, overall properties of the system. Often misunderstandings arise between the customer and the developer and maintenance operator due to miscommunication. So this section of document will help in clarifying the requirements of the proposed system. Some of the intended audiences of this section include system customer, project manager, system engineer, system test engineer and system maintenance engineer.

3.2 System Overview

This system was developed to capture human motion through cameras and use 3d reconstruction to model that motion in computer graphics. The first part i.e. tracking was done by neighborhood matching and region growing. A number of marked points on the human body can be tracked through this technique. The reconstruction process can be divided into three parts: first, calibration of cameras being used; second, compute the camera matrices from camera matrices; in the end, compute the points in space that project to these two image points. This system can reconstruct a point in 3d from two

views of an object. Also the system has been employed with a Matlab calibration toolbox for calibration to compute the camera matrix. The resulting sequence of points will be used later to reconstruct a set of points in space representing the object surfaces on the scene. In the last system rebuilds the extracted 3d motion frame by frame in a computer graphics application like OpenGL.

3.3 Assumptions and Dependencies

The MOCAP has certain limitations. All motion capture has to be done in a controlled environment i.e. a black background behind the actor. Large amount of external sunlight that might hinder the cameras from capturing the markers is also discouraged. The actor is required to wear black clothing so he becomes indistinguishable from the black background and for the easy detection of the white markers. The markers need to be considerably small to allow speedy computation and avoid any errors in tracking and translating motion. At least 2 and not more than 4 cameras would suffice for the MOCAP system with resolution 320x240. All cameras are to be calibrated exactly as per the instructions so that scene geometry can be constructed accurately.

3.4 Operating Environment

Motion Capture, MOCAP, should run on Operating Systems such as Windows XP, Windows Vista, Windows 7. Client PC Requirements include Pentium 2 GHz Core 2 Duo Processor, 1 GB Ram or more. Server PC Requirements include Pentium 2 GHz Core 2 Duo Processor, 1 GB Ram or more. System also requires laptop or PC for each client, 320x240 resolution webcams, one for each client, and room of 13 by 20 by 7 feet.

For faster and better tracking, it was sought to place the actor and the background in similar rather than contrasting colors, i.e., black background and black apparel for actor. System requires white markers made of any cloth. The reason for using white markers is that the team has worked on very basic human motion tracking. Even with different colored markers their aims can be achieved but when they would consider a more advanced and detailed tracking markers of different colors might start to produce redundant data.

3.5 Product Features

Product features include Motion Capture/Recording that is done by using 3 to 4 cameras in a controlled environment so that the motion of an actor would be captured who would be wearing a black suit with white markers in front of a black background. The motion would be captured using the assistance of all present cameras so that cases of occlusion could be avoided. The motion can be recorded to be analyzed later for study. Another feature is to analyze human motion. The recorded human motion can be analyzed to find the necessary requirement to translate that very motion on to graphical models. Another feature is for calculating 3d correspondences of motion. Using camera calibration data and tracking from each camera 3d motion in the real world scene will be computed and motion will be modeled on a virtually created object.

3.6 User Problem Statement

Motion capture for use in animations is too much expensive around the world. A single motion capture studio takes 10,000 USD / day rent and whole equipment is sold around

200,000 USD. In Pakistan not many animation houses use this technology due to its expensiveness. The developed system can hopefully provide them a cheaper system.

NexGen gaming consoles like Microsoft XBox 360, Sony Play Station 3 and Nintendo Wii are using motion capture techniques to provide immersive gaming environment. This system can be used to build applications to compete even these systems as this solution is both robust and real time. A lot of work will be needed to done but it is achievable at a much cheaper price than the above mentioned systems.

Virtual Reality systems need to be real time and it is also a small industry in Pakistan so this system also tends to resolve this problem as well.

3.7 Specific Requirements

This section includes the functional requirements as well as the non-functional requirements.

3.7.1 Functional Requirements

System will use web cameras of resolution of 320 x240 to capture the motion of actor. Camera calibration will be done by the user for each camera to find extrinsic and intrinsic parameters. To start tracking, user must initialize tracking on each camera by clicking on each track-able point and then tell the program to start tracking. Calibration will be used to calculate camera projection matrices. Calibration will give 1 intrinsic matrix ($K_{3 \times 4}$) and 2 extrinsic matrices (rotation: $R_{3 \times 3}$, translation: $T_{3 \times 1}$). Use the camera projection matrices and tracking data for 3D scene reconstruction. The video is

transferred in the form of frames one by one and motion is reconstructed with every new coming frame.

3.7.2 Non-Functional Requirements

The performance of the system is directly dependant on the number of cameras being used, i.e., the more the cameras, the better the parameter calculation as more data would be available and it would drastically reduce cases of occlusion, processing speed of server, i.e., the better the procession power, the faster the calculations etc. , image background, i.e., the background is an important constraint in the system as it leads to easier tracking with a darker background or background of same color as actor's apparel.

The quality attributes of the system include: The system is flexible as it can adapt to more changes when more clients are added. System would be easy to maintain as cheap hardware is used which would be replaceable. A lot of work has been done on Motion Capture in the market already which helps in better understanding of this field. MOCAP would be quite reliable in its results as it would be tested with various data sets. Test cases from the Brown University's dataset "Human Eva" are going to be used to test its workings. The system can be reused to enhance it for other applications like physio therapy, video games etc. It is easy to test as only actor is required in the working environment. This system can withstand changes occurring over time as it can be equipped with better cameras if the expenses are available.

3.8 Target Environment

The target environment for MOCAP would be the animation development studios or 3D simulation studios in Pakistan. MOCAP also has its applications in the field of VR simulations, NexGen gaming consoles, automatic animation of CG models, automation of physical therapy, gesture driven applications, immersive trainings and many more.

3.9 System Requirements

Technical requirements cover Hardware and Software requirements implementing the proposed system.

3.9.1 Hardware Requirements

Presently, the hardware components and interfaces that are needed include Pentium 2 GHz Core 2 Duo Processor, 1 GB Ram or more , computers to create client server network, and 2 webcams to capture video but more can be required if advance level support/functionality is desired of the system.

3.9.2 Software Requirements

The functional modules of this system require MATLAB, OpenCV, OpenGL, Visual C++ Runtime Environment, Windows XP, Windows Vista, Windows 7 to properly work.

CHAPTER 4

SYSTEM DESIGN AND ARCHITECTURE

4.1 Introduction

This section of the document contains a structure for a software design specification. Most software projects fail because of the flaws in the design, so the design phase can be referred to as a very crucial stage in the software development lifecycle. The purpose of writing this document is to describe the design of the system in detail. This document contains a detailed description of the design of the system, and helps in clearing any doubts which might have been left in the specification of the requirements of the system. Misunderstanding between the users of the system and those developing it can further be clarified if the user can see the design of the system. So this document will help in clarifying the requirements as well as design of the proposed system. Some of the intended audiences (readers) of this document include system customer, project manager, system engineer, system test engineer and system maintenance engineer.

4.2 System Overview

This part of the document describes the design relationship between the system, its components and the external environment of the system. The complete system network includes both hardware (the web cameras) and the software (motion tracker, motion

analyzer and the motion modeler). The system is a stand-alone system and does not depend upon any other existing system.

4.3 System Architecture

Architectural Diagrams shows the basic architectural layout of the system being designed. In the initial stages of this project, the team started off with a simple single machine based architecture where all processing and modules were implemented using a single program. Even though this would have given satisfactory results but it was decided to opt for a different scheme so as to employ techniques for faster processing and better results. This subsequently led them towards the Client Server Architecture.

The project is based on the client server architecture .There are various reasons for this requirement .The team could have used a simpler approach by not adopting the client server architecture but they wanted this system to be both fast and robust.

The client server architecture has many advantages over a single machine based system. The following are the reasons that lead the team to the adoption of the client server architecture in this system.

They needed to implement complex functions of image processing for tracking and calibration and for that they required fast processing and the client server architecture gives the required output. Allowing different modules to be created for and implemented by separate machines and thus giving more processing power per module. The system becomes more scalable by using the client server architecture. The number of clients can

be increased to accommodate more sophistication and allowing the user with even more processing power.

System is a TCP client server based where tracking is part of client's functionality and 3D reconstruction is done by the server. Architecture is represented in figure 4.1.

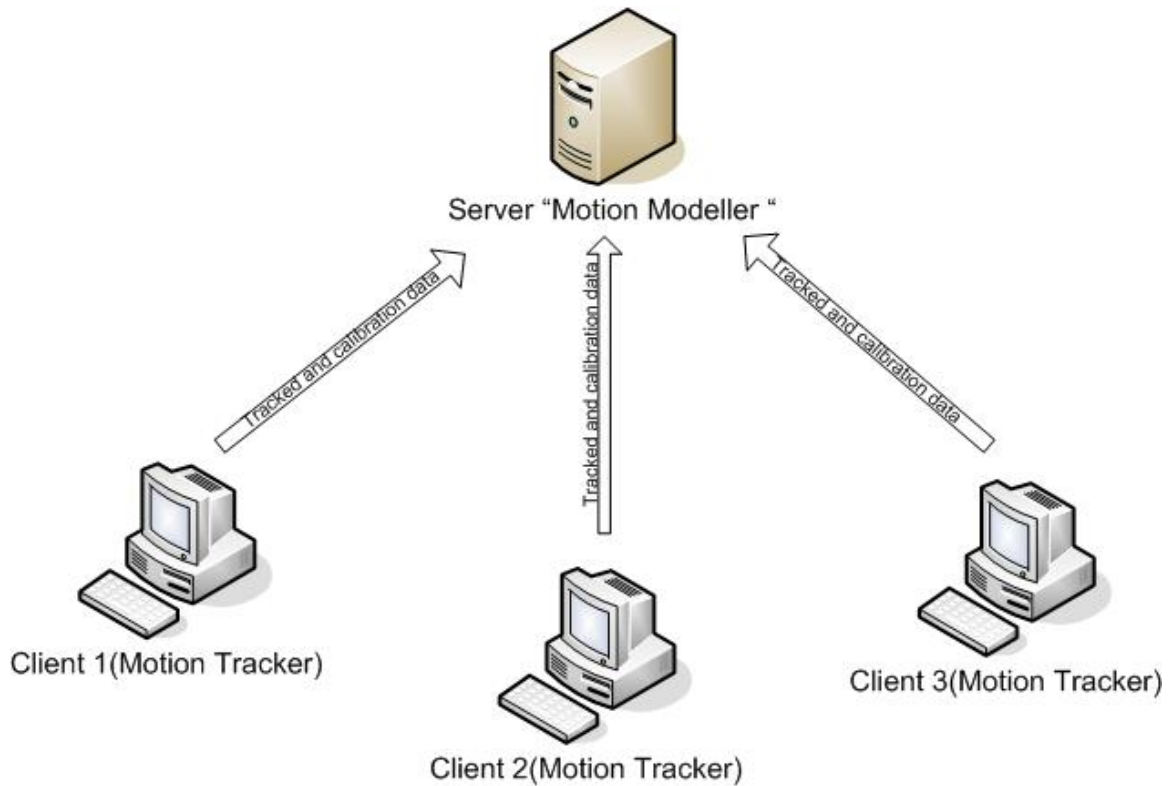


Figure 4.1 The Architectural Framework of MOCAP

The architecture diagram shows that multiple clients are communicating and working in collaboration with a single server. Client 1, Client2 and Client 3 are all working simultaneously with their own cameras and would be handling their own functions and working. Each client does its own tracking and calibration which is associated with its own camera. The server used the data fed by all three clients for 3D Reconstruction.

4.4 Decomposition Description

Under this heading, all the functionalities of all modules present at both the client and server sides are discussed in detail.

4.4.1 Motion Tracker

The actor wearing black apparel stands in front of webcam with black background and white markers attached at his joints. The actor will act in front of webcam and the video is captured and tracked by motion tracker. This module is present at the client side and sends the tracked video frames to the server.

4.4.2 Motion Analyzer

This module receives the tracked data send by the motion tracker and analyzes it at the server side. It analyzes the motion i.e. it finds the relation between objects, their orientation and most importantly their mathematical angles with each other.

4.4.3 Motion Modeler

The analyzed data is received by motion modeler from the motion analyzer. The server used this input and uses that data to translate that motion onto a 3D graphical model.

4.5 Data Design and Data Description

The data items to be used by the system are real time video, number of tracked video frames and tracked data and projection matrices for 3D scene reconstruction. The figure 4.2 is a data flow model of the system on client side.

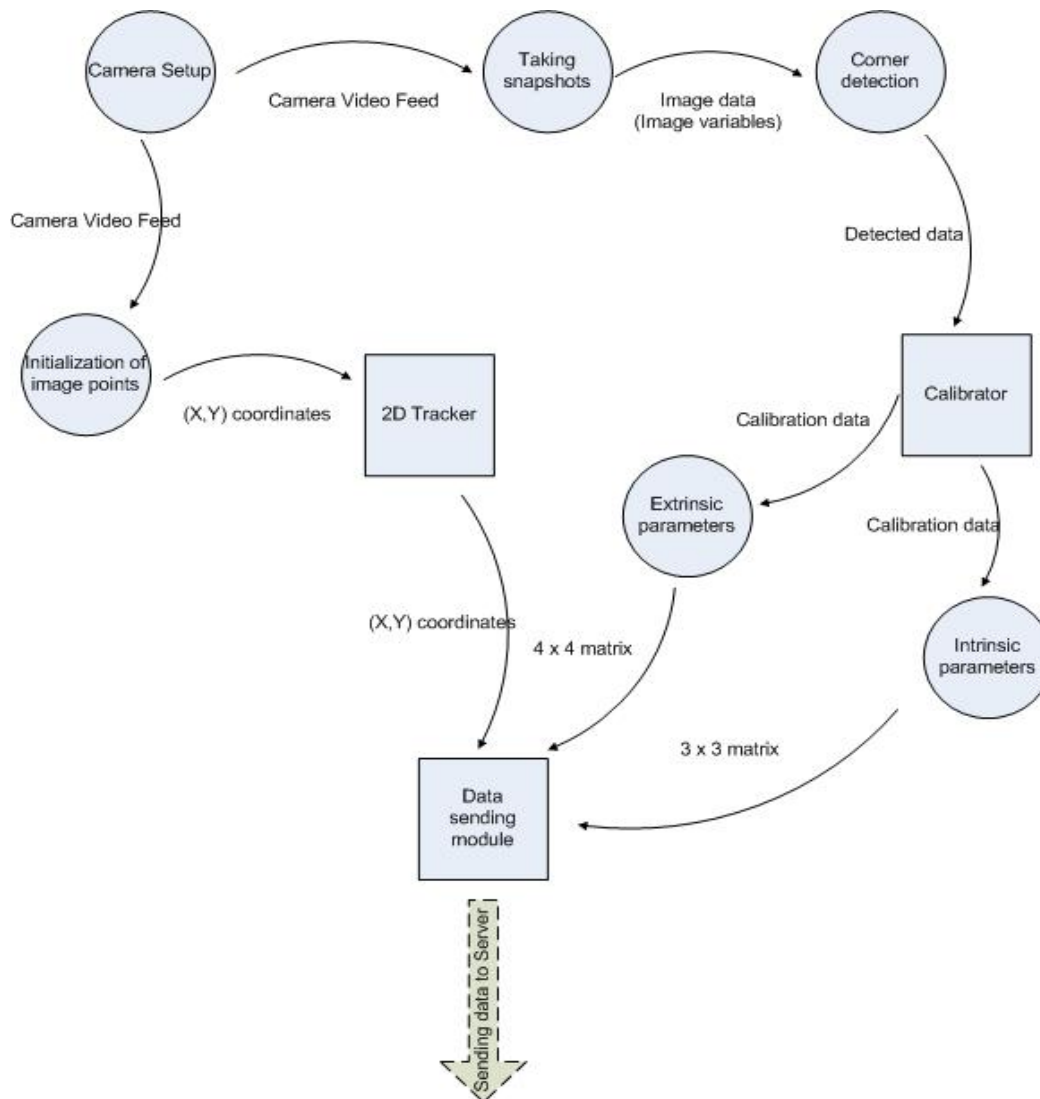


Figure 4.2 Data Flow Diagram at Client Side

The cameras will give two kinds of data. Video feed will be used by the tracking module for tracking initialization leading to object recognition which will be used for tracking in subsequent frames. This tracking is performed by the 2d tracker which then sends the data to data sending module which sends the data across the network. The other kind of data from the cameras, calibration data is calculated by still snapshots of the calibration object. These stills are used by the corner detection process to detect corners which are

then used to do the calibration. Calibration gives two calibration matrices. Both the intrinsic and extrinsic matrices are sent to the data sending module.

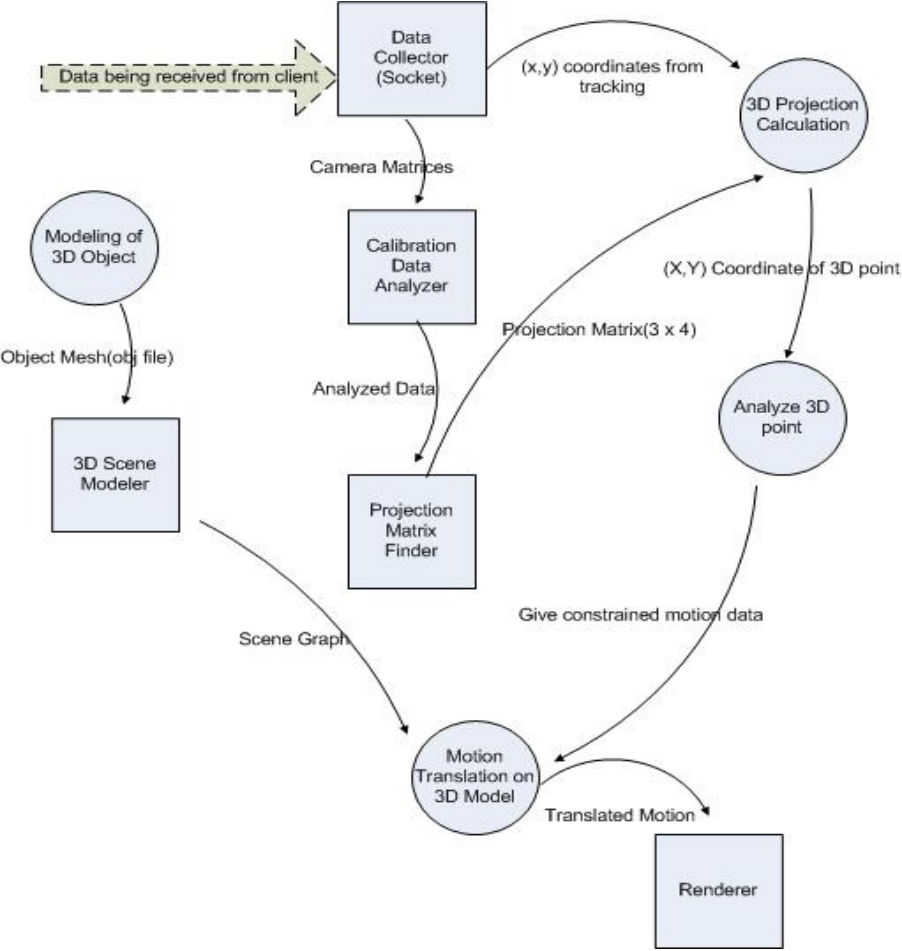


Figure 4.3 Data Flow Diagram at Server Side

In the figure 4.3, the socket entity collects data from the client. The calibration data is sent to the projection data calculator process and tracking data to the projection calculating module. The projection data calculator computes projection matrix. This matrix is then sent to projection calculating module. Projection then computes 3d projections and sends the 3d point data to 3d point analyzer. 3d object modeler sends the mesh data 3d scene modeler which sends the data to motion translator. Motion translator

translates the mesh data according to the calculated projections. This data is sent to renderer who renders the system.

4.6 External Interface Requirements

These are the interface requirements of the system.

4.6.1 Hardware Interfaces

First are the webcams which use USB interface to connect to PC. The client-server architecture is supported by standard LAN cables.

4.6.2 Software Interfaces

Data from Webcam is acquired through standard USB port by OPENCV inbuilt function “cvCaptureFromCAM(int portno)”. The client server architecture between different modules is handled by C++ Winsock’s TCP functions.

SYSTEM IMPLEMENTATION

5.1 Introduction

This section of the document contains the implementation details of the system. It describes the functionality of the system and also a complete explanation of the system from the implementation point of view. The basic idea is to make the reader familiar with the implementation details of the system so that he can have some idea about the actual working of the system.

The System comprises of a Calibration tool in MatLab, a server, a client (both implemented in C++) and the environment itself. Client acts as the eyes of the system and acquires video stream through web cam attached to the system. The video is then processed upon by the client and the required motion is tracked. The client connects with the server in the meantime. Calibration data is exchanged first. Then comes the tracking initialization part where user specifies the points to be tracked that will be used eventually for 3d reconstruction. The client then starts to send the tracked data frame by frame to the server. The Server then using the calibration and tracked data from two clients reconstruct the scene in 3d for which OpenGL is being used. The environment consists of an actor in black apparel against any black background with white markers on the joints to be tracked. These can be of any material (cloth, tape, plastic etc).

5.2 Calibration Toolbox

Following is the description of calibration toolbox.

5.2.1 Overview

This is used for calibration, it's a free open source code written in MatLab. It uses a number of images of the calibration object and based on user data on real world units it provides the camera parameters. Camera parameters are exported to a file named "Extrinsic.txt" which is then used by the client to read calibration data.

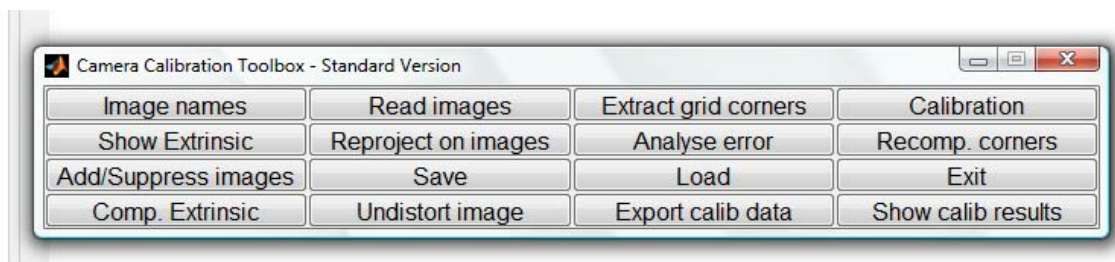


Figure 5.1 Calibration ToolBox

The figure 5.1 is the snapshot of calibration tool box GUI. Functionality is explained below.

5.2.2 Detailed Explanation

a) Calibration itself

Reading the images: First user clicks on the **Image names** button in the **Camera calibration tool** window. Then the base name of the calibration images (**Image**) and the image format (**jpg**) are entered. All the images (the 20 of them) are then loaded in

memory (through the command **Read images** that is automatically executed) in the variables **I_1**, **I_2**, ..., **I_20** as in figure 5.2. The number of images is stored in the variable **n_ima** (=20 here).

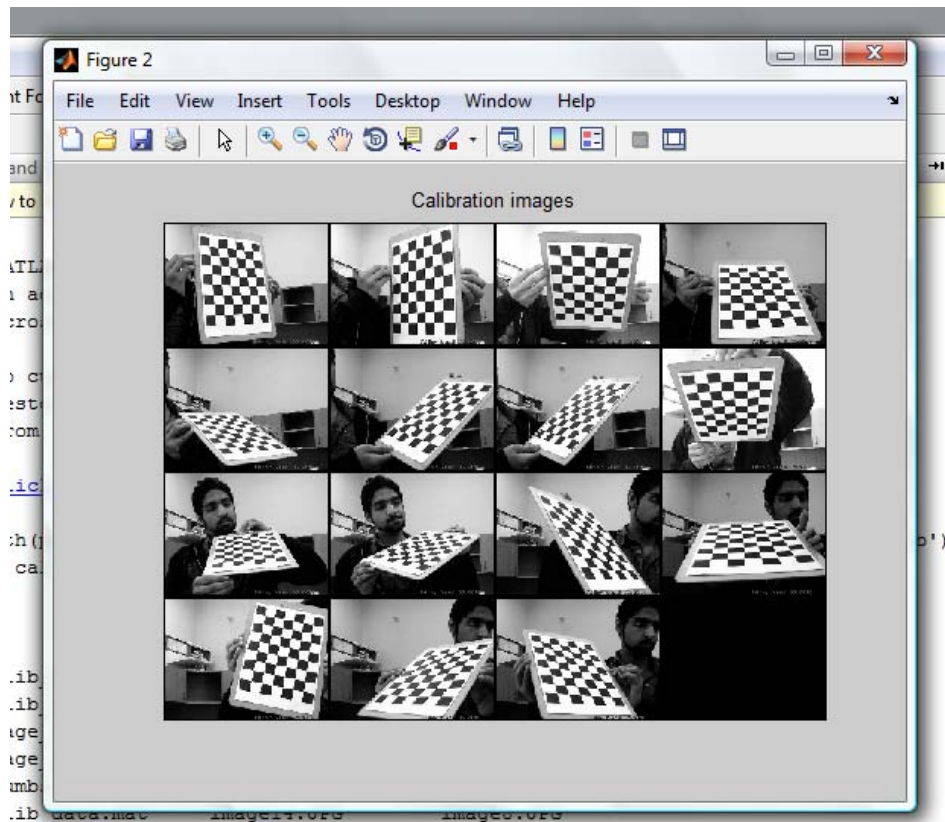


Figure 5.2 Images Read

Extracting Corners: The **Extract grid corners** button is clicked. The corners are extracted by clicking on them one by one on each image. It's a lengthy process but once camera is calibrated it need not be repeated for intrinsic parameters.

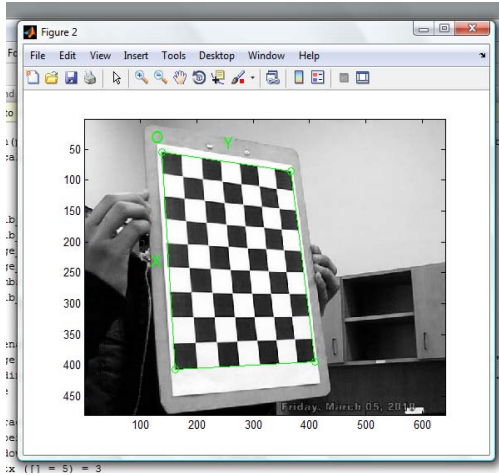


Figure 5.3 Clicking on Corners

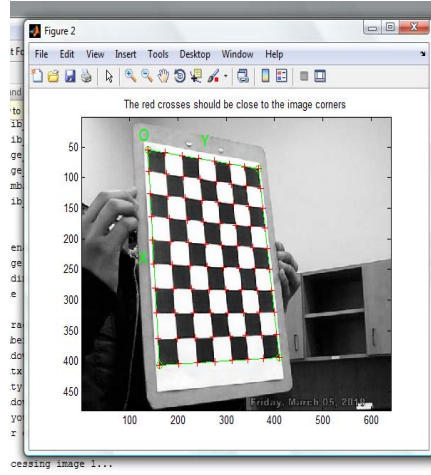


Figure 5.4 Corners Extracted (Red)

First figure 5.3 shows the result of clicking on the tool box and when the required inputs are given i.e. number of boxes and measurement in real units the corners extracted are shown in figure 5.4. The green lines show the XY axis as indicated and red points are showing the corners.

Calibration: Now **Calibration** button is pressed which computes the intrinsic calibration parameters as shown in the figure 5.5.

Calibration results (with uncertainties):

Focal Length: $fc = [893.97550 \ 903.17247] \pm [5.07640 \ 6.91673]$
 Principal point: $cc = [308.96719 \ 218.50977] \pm [14.56654 \ 12.67625]$
 Pixel error: $err = [0.60609 \ 0.71381]$

Note: The numerical errors are approximately three times the standard deviations (for reference).

Figure 5.5 Calibration Results

This data is saved using the **Save** button and for another run of the system this data will only be needed to load using the **Load** button.

For Extrinsic Calibration: It is done whenever the environment is changed or specifically camera position is changed. Intrinsic parameters are loaded first. Then the button Comp. Extrinsic is pressed which prompts user to enter the name of the image. Corners are clicked same as in corner extraction part but this time position and orientation of camera with respect to the environment is computed.

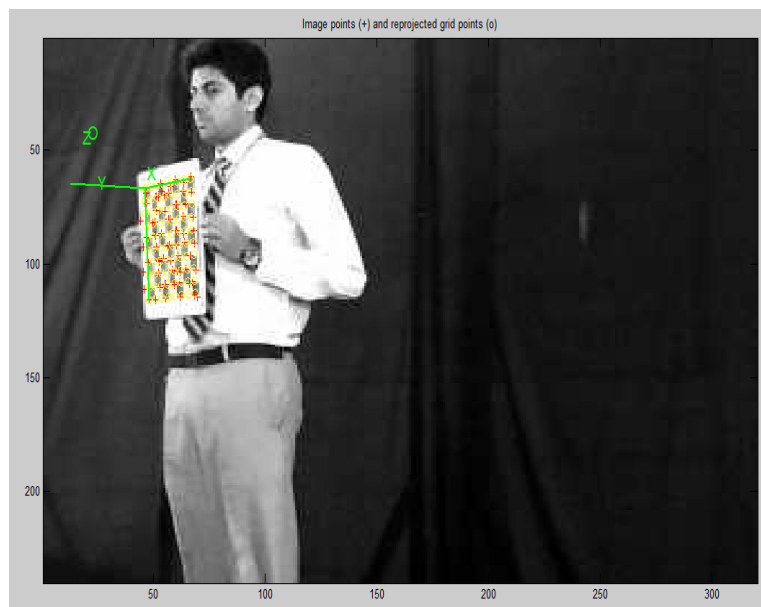


Figure 5.6 Extrinsic Parameter Image

The actor held the calibration object as snapshots from each camera was taken for extrinsic parameter calculation. The three green lines show the XYZ axis as indicated and red points are showing the corners. The extrinsic parameters are calculated in figure 5.6 and are listed as in the figure 5.7:

Extrinsic parameters:		
Translation vector:	$Tc_ext = [$	$-366.205029 \quad -158.439767 \quad 1064.895319]$
Rotation vector:	$omc_ext = [$	$-2.198348 \quad -2.161748 \quad 0.011205]$
Rotation matrix:	$Rc_ext = [$	$0.017612 \quad 0.998781 \quad -0.046109$
		$0.999205 \quad -0.015933 \quad 0.036531$
		$0.035752 \quad -0.046715 \quad -0.998268]$
Pixel error:	$err = [$	$0.70663 \quad 0.61109]$

Figure 5.7 Extrinsic Parameters

Parameters Exported: This is done by clicking the **Export Calib Data** button. User is required to enter the name of the file which should be “Extrinsic.txt” and after its creation; it is copied with the client executable.

5.3 Client

This section includes the description for client module.

5.3.1 Overview

The client is responsible for handling the camera, tracking through it and sending the data to the server through a TCP connection. It comprises of the following modules.

1. Calibration Reader
2. TCP Networking module
3. Video Acquisition Module
4. Tracking

It was developed completely in Visual Studio 2008 in C++ using OpenCV.

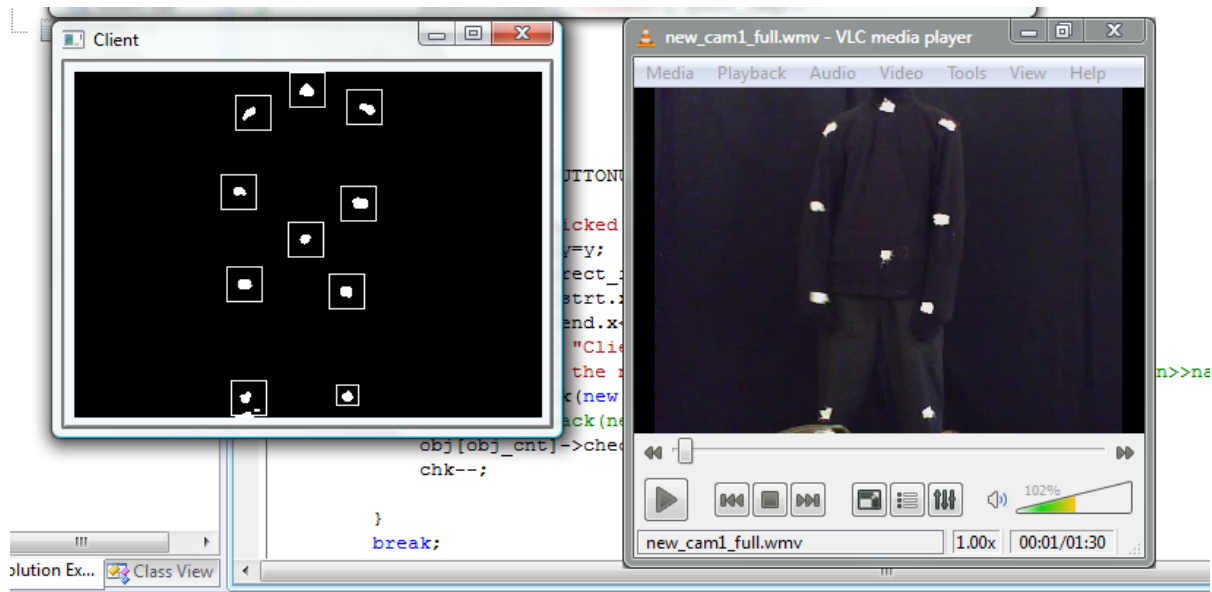


Figure 5.8 Client Module

In the figure 5.8, on the left is shown result of tracking module and on the right is shown the video.

5.3.2 Detailed Explanation

1. Calibration Reader

The calibration from MatLab is read here into the client program. This is done by file I/O operations in C++. A separate class Camera is used for handling the camera. Whose function doCalibration() extracts the intrinsic and extrinsic data from the file and stores it in the Intrinsic_mtrx[] and Extrinsic_mtrx[] respectively.


```
Reading Calibration Data from Extrinsic.txt
0: -366.205
1: -158.44
2: 1064.9
3: 0.0176124
4: 0.999206
5: 0.0357522
6: 0.998781
7: -0.0159334
8: -0.0467153
9: -0.0461086
10: 0.0365314
11: -0.998268
12: 893.976
13: 308.967
14: 903.172
15: 218.51
```

Figure 5.9 Reading Calibration Data

Figure 5.9 shows the calibration data being read from Extrinsic.txt. Then the projection is calculated as Intrinsic x Extrinsic Matrix by the function calc_Projection() in Camera class as shown in figure 5.10:

```
Projection matrix:
26.7913
878.452
-349.652
1639.33
910.267
-24.5983
-185.137
89591.5
0.0357522
-0.0467153
-0.998268
1064.9
```

Figure 5.10 Projection Matrix

Figure 5.10 shows the projection matrix obtained by multiplying extrinsic and intrinsic matrix.

2. TCP Networking module

The library utilized for TCP networking is visual studio's winsock 2.0 library. The client establishes the connection with the server and then sends the data one by one. First it converts the float projection matrix to char values and then sends it across the stream. Then tracking initialization data is sent i.e. names of the tracked objects and in the last when tracking is performed in a loop the data from tracking is also sent in the loop. This last data is in the form of X and Y coordinates of tractable objects in a float array which is converted to char for network streaming. This Communication is shown in figure 5.11

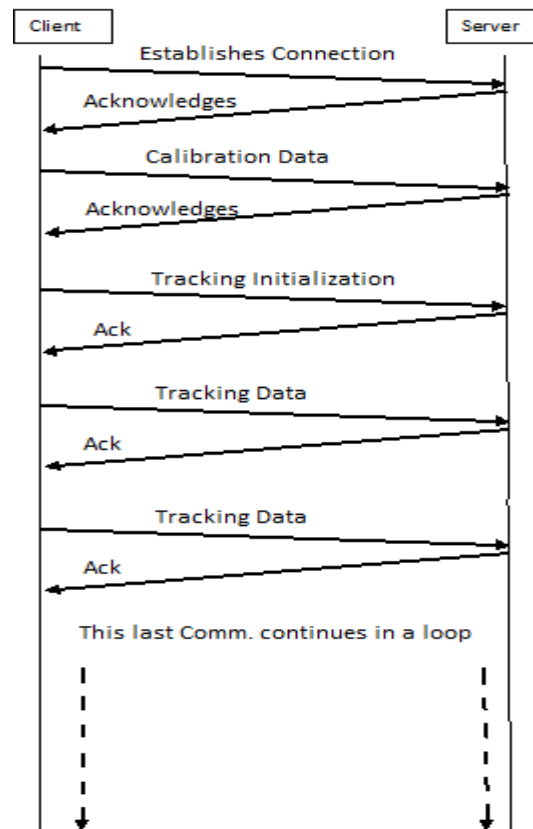


Figure 5.11 Communications between Client and the Server

Figure 5.11 shows communication between client and server. Client sends request to server for connection establishment and server acknowledges it. Then client sends calibration data, initializes tracking, and sends tracking data which are acknowledged by the server and this communication continues repeatedly until all data is sent.

3. Video Acquisition Module

This part is responsible for handling the video feed: connection with the webcam, acquisition of the video, getting frames out of it. Functions and variables used for this were from Intel's OpenCv library.

Explanation of the code elements used is shown in the following table.

Name	Nature	Functionality
CvCapture	data type	Used as the video container
IplImage	data type	used as the frame(image) container
cvCreateCameraCapture()	Function	Used to capture video from cam
cvNamedWindow()	Function	Used to create video window
cvThreshold()	Function	Used to convert image to black and white
cvQueryFrame()	Function	Get frame from video
cvShowImage()	Function	Show the frame from video

As shown in figure 5.12 it shows the video in black and white. On the right is the original video on WM player.

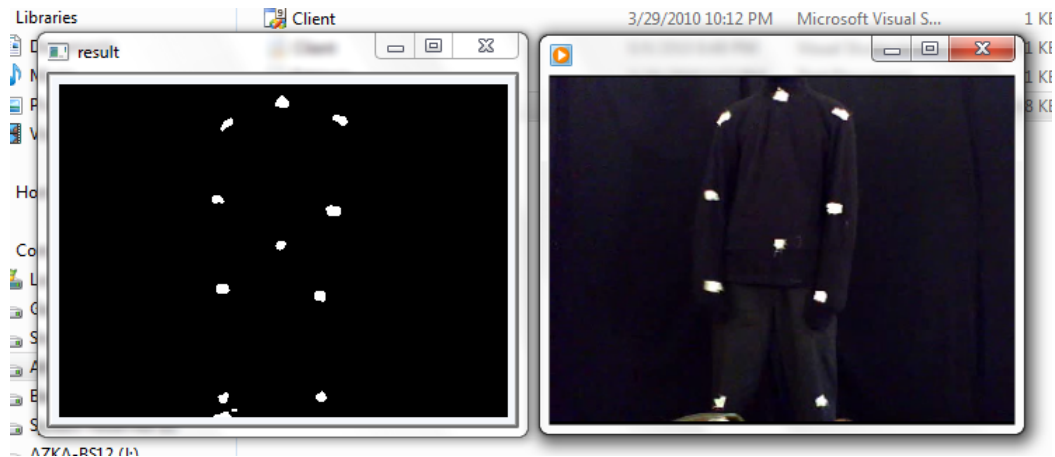


Figure 5.12 Acquired Video on Right and Original on Left

Figure 5.12 shows a comparison between original video and acquired video detecting the markers.

4. Tracking

Tracking as already explained in the design chapter is accomplished by neighborhood search. The feature class is used to instantiate the individual features. This class contains the tracking code. There are two parts to tracking.

a. Initialization

This was done by giving the initial coordinates of the feature to the system by drawing a marquee around it, the system then asks for the name of the feature which is then provided. The mouse listener function used for this part is `my_mouse_callback(int event, int x, int y, int flags, void* param)`. This function draws the marquee and then searches for the object in this marquee. Then it creates a separate feature object and initialize it with the name and the coordinates. As shown in the figure 5.13 the red circled

objects are in the squares indicated by the mouse. Rest of the objects (in white) are waiting to be initialized.

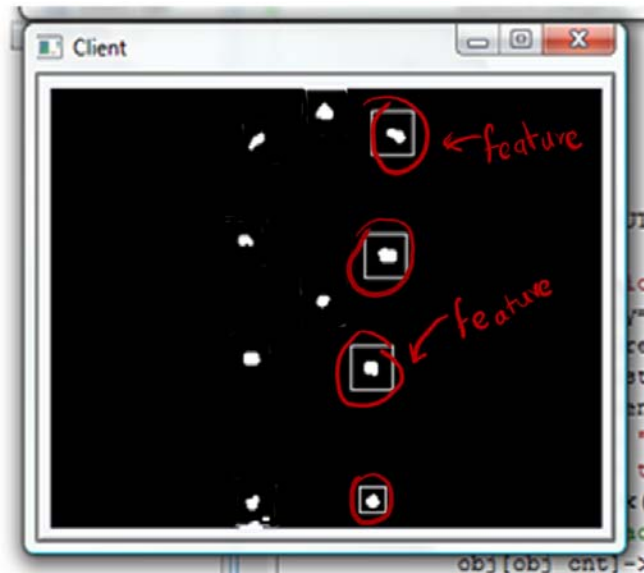


Figure 5.13 Feature Initialization

Figure 5.13 shows feature initialization in which the circles marked red are objects that are initialized and the unmarked white markers are the objects waiting for initialization.

b. Tracking

This was done by a modified neighborhood search. The searching function starts to search from the last mean i.e. centre and searched in concentric circles outwards until it finds the object. There is no limit for the neighbors which increase the chance of object getting found. Once found it labels the object in only a single horizontal and vertical line and mean is found in x and y direction. As using small markers are being used so correct centre is found anyways. In this way it saves time and helps in achieving the goal of a real time system. Labeling is done in the label () function in the feature class. It gets an

initial index and labels the surrounding connected white pixels. Search is done in the search_concentric(int) function which takes the index from last frame and searches in circles outwards until it finds the object. The figures 5.14 and 5.15 show this algorithm at work.

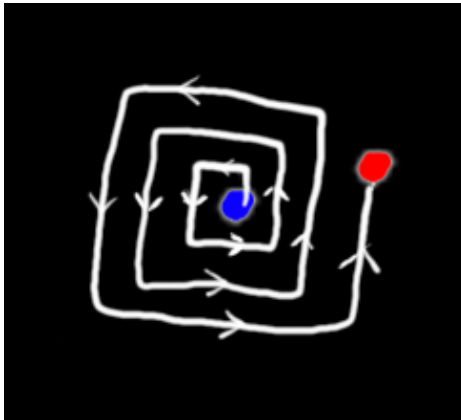


Figure 5.14 Searching

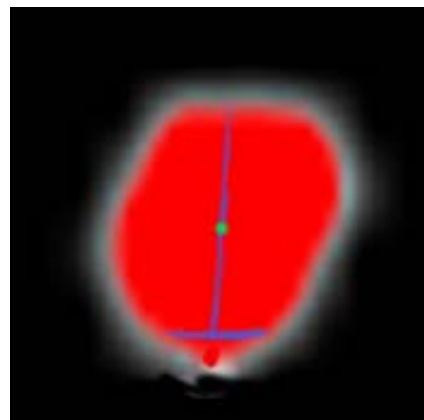


Figure 5.15 Labeling

In Figure 5.14 the object in blue represent the object's location in frame 1 and red one in frame 2. Objects are always white in the system, colors are used here for understanding purpose. The white path shows the concentric search and how it finds the moved object in frame 2 starting from last position index. In Figure 5.15 corner labelling is shown.

5.4 Server

This section includes the description of server module.

5.4.1 Overview

The server is responsible for handling the clients i.e. cams, 3d reconstruction, and rendering the scene. It comprises of three modules, a multi-threaded Server, 3d reconstructor, renderer.

It was developed completely in Visual Studio 2008 in C++ using OpenCV and OpenGL.

5.4.2 Detailed Explanation

1. A multi Threaded Server

It controls n number clients. It was implemented using winsock 2.0 library. One accomplishment at this level was multi-threading. It was needed to run project's renderer separately from the comm. server so that both can work independently. Not one hinders the efficiency of the other. Option for multi threading was chosen. Threading was achieved by using inheritance with the class thread .The structure of threading is given in the figure 5.16.

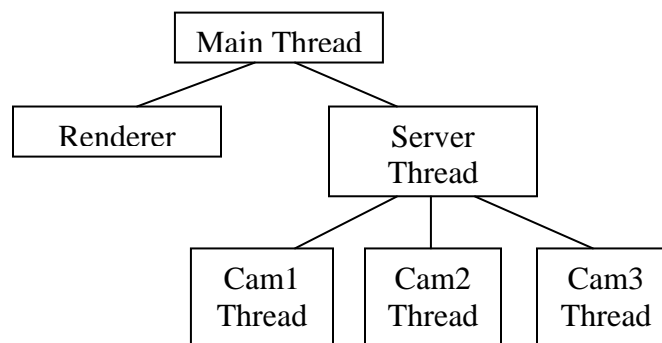


Figure 5.16 Multi Threading Structure

In Figure 5.16 it is shown that renderer runs separate from the Server. The data exchange is explained later when 3d reconstructor is explained. Different clients interact independently with the server. Synchronization is necessary in this case which is handled by the server and hence no data is misplaced.

A new thread for each client is created each time a client connects. The communication between server and client is explained below.

The server starts and wait for the client to communicate. When client connects and sends the calibration data, it is saved in separate camera instances associated with each client held in a vector camera_list. Initialization data is used to create instances of the feature_server class which acts as objects on server side .

When another client connects, server doesn't allow it to initialize when one has already initialized. This is to ensure that features get initialized correctly. Same data from two different sources can include redundant feature points in the list. Once the server starts receiving tracking data it updates the Feature server accordingly.

This data is then used by the 3d Reconstructor which when completes reconstruction, signals the server by setting the projection_flag = 0 so that it can update the tracking data. Now data from at least two clients is needed to find projection so it is the server's job to maintain synchronization between clients. One more issue here was to restrict clients to update tracking when projection is being calculated so that only fresh and in-sync data reaches the reconstructor.

This synchronization is attained by using a locking methodology. The piece of code that updates keeps a check. Only one client accesses it at a time. Other clients are locked outside by using a locked loop. When one by one, the data is updated by at least any two clients the reconstructor is signaled which project the 3d point and updates the scene.

2. 3d Reconstructor

Two classes Projection and FeatureServer collaborate to perform the strong matrix computations to calculate the projections whenever following requirements are met.

1. Tracking data from at least two clients is received.
2. Tracking data is fresh and sync-ed.

FeatureServer::setProjectionParams() saves the tracking data and keeps a check by the variable int f_no. Whenever it gets two sync-ed tracking data, it calls the Projection::calc3DProjection() which calculates the 3d Projection for the current feature. A loop has been used in the myClient side to ensure every feature's projection gets calculated..

3. Renderer

This is done by the Rendering class. The rendering thread waits for the bool Rendering_flag to render the update projection data. Server signals the renderer once projection gets calculated by the 3d Reconstructor. Renderer does the following jobs.

It creates an openGl window for rendering titled MOCAP: A Cheap Approach (Rendering::displayGl ()). It draws the scene with a horizontal grid in the middle

(grid()). It draws a skeletal model to represent human motion and then move it according to the data (make_lines()).

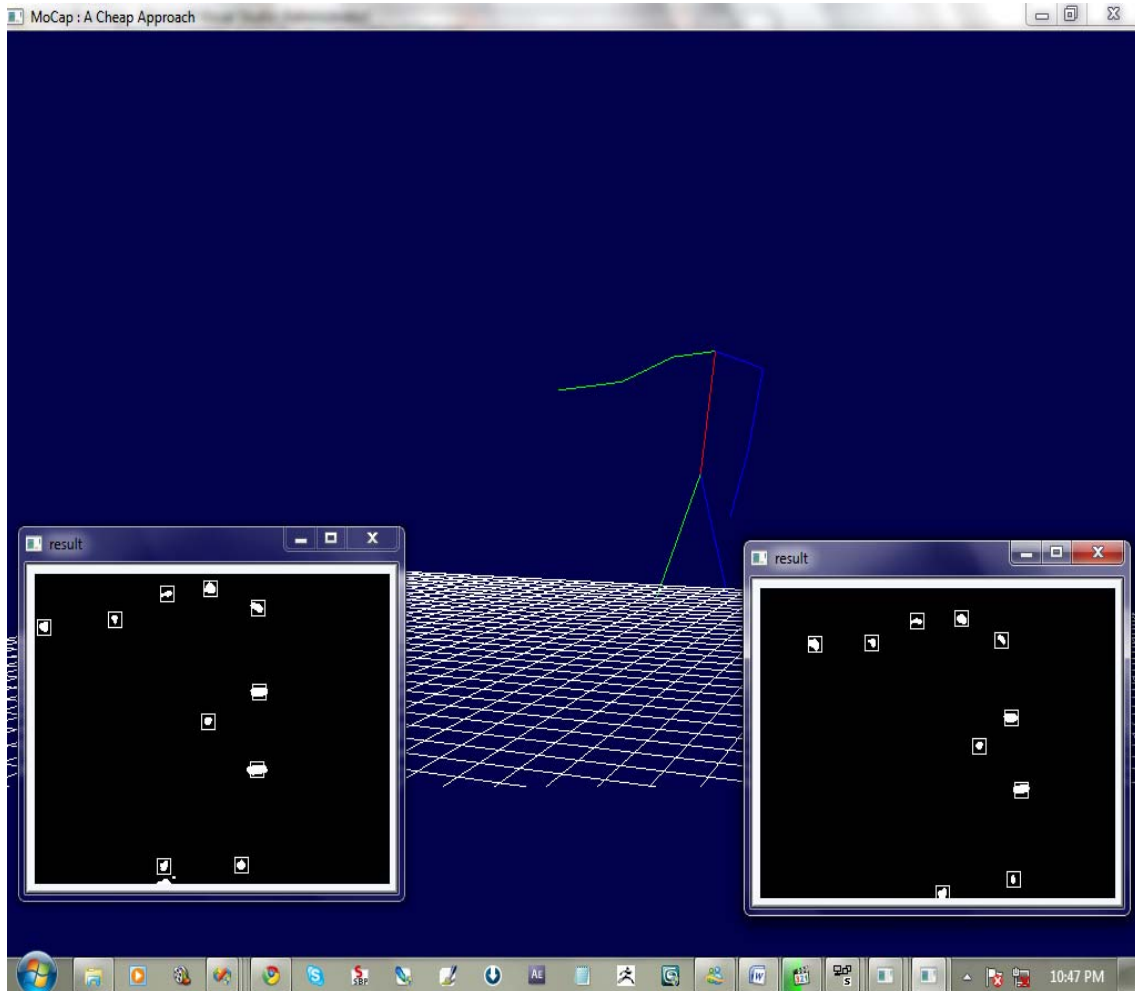


Figure 5.17 Skeletal Model Moving According to Tracked Data

The figure 5.17 shows 3d rendering being done by the server from the input from the two clients on the lower corners.

COMPARATIVE ANALYSIS

6.1 Introduction

In this chapter a comparison of this system with state of the art systems in the world has been given. Now this comparison has been done keeping in mind necessary qualitative, quantitative and technical attributes of a motion capture systems. First, test results of this system have been presented then description of the systems around the world that have been compared with this system. After that actual comparison has been given followed by a conclusion.

6.2 Test Results

This section gives the testing methodology and the results for the system.

6.2.1 Introduction

An extensive amount of testing has been performed on this system. Since the system contains a lot of modules which share and modify data and produce results based on that data. Testing path as shown in figure 6.1 was chosen.

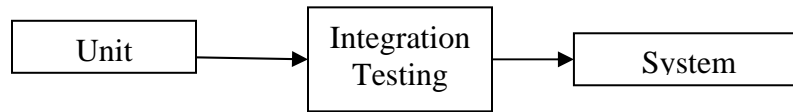


Figure 6.1 Testing Path

Figure 6.1 shows that testing was first performed on modules independently, once satisfied the modules were integrated and integration testing was performed which ended up in system testing.

6.2.2 Unit Testing

Details about results of unit testing are given below.

6.2.2.1 Tracking

Tracking algorithm was tested by running it through one CG video (created automatically for testing) and several recorded videos.

CG video

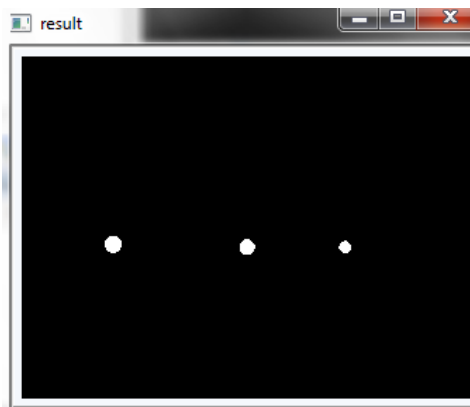


Figure 6.3 Tracking Initialization

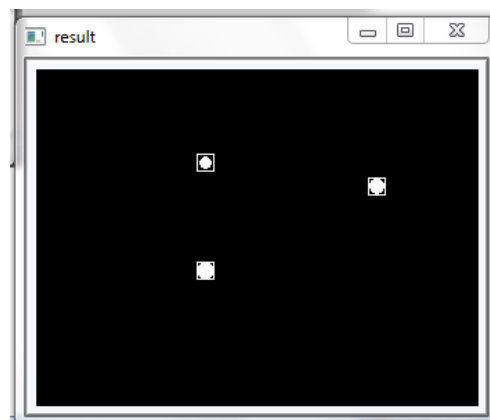


Figure 6.4 5 Seconds into the Video

Real Video

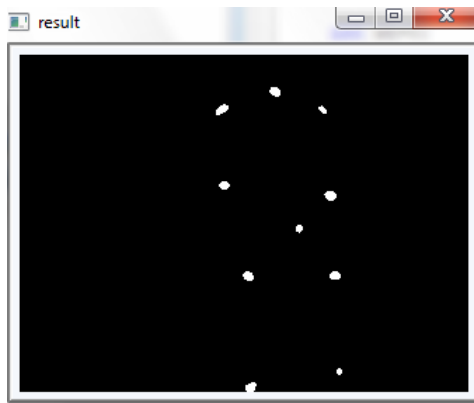


Figure 6.5 Tracking Initialization

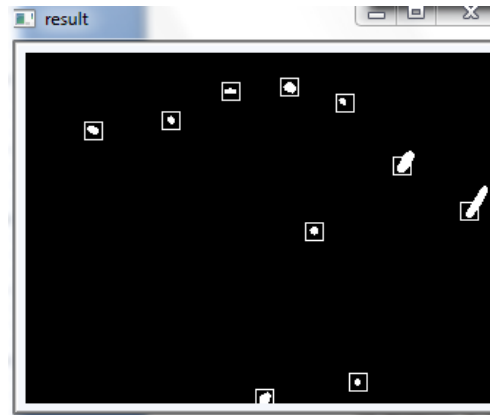


Figure 6.6 2 Seconds into the Video

The test results shown in the figures 6.3-6.6 show both initialization and tracking in (a) and (b) parts. The squares around the objects show that they are getting tracked.

There were too properties to check in this testing. One to check if tracking loses any object during the video due to intricate motion paths, second if tracking is fast enough to be real time.

First that the tracking does not lose any object unless the motion is very fast which is not the short coming of tracking algorithm itself but due to slow fps of webcams?

Second that the Tracking is fast enough as its average speed was 120 frames/sec.

Tracking from a live video test was passed successfully.

6.2.2.2 3D Reconstruction and Rendering

It has been performed on the 3d Reconstruction and Rendering modules as they could not be tested correctly by unit testing as they needed real data and other modules to work. They could be tested independently with hypothetical data but it would have taken a lot of time and it still would not have known if all the computations are correct. E.g. If 3d Reconstruction gives some 3d coordinates without rendering them it's hard to make out where they exactly lie. One can only make an intuitive guess. So it was better to test these at integration level.

The best way to test this was to use a real video of some motion so that it can be clearly seen if reconstruction is accurate. Only there was a little jitter in the movement which indicated in some percentage of error in the 3d data.

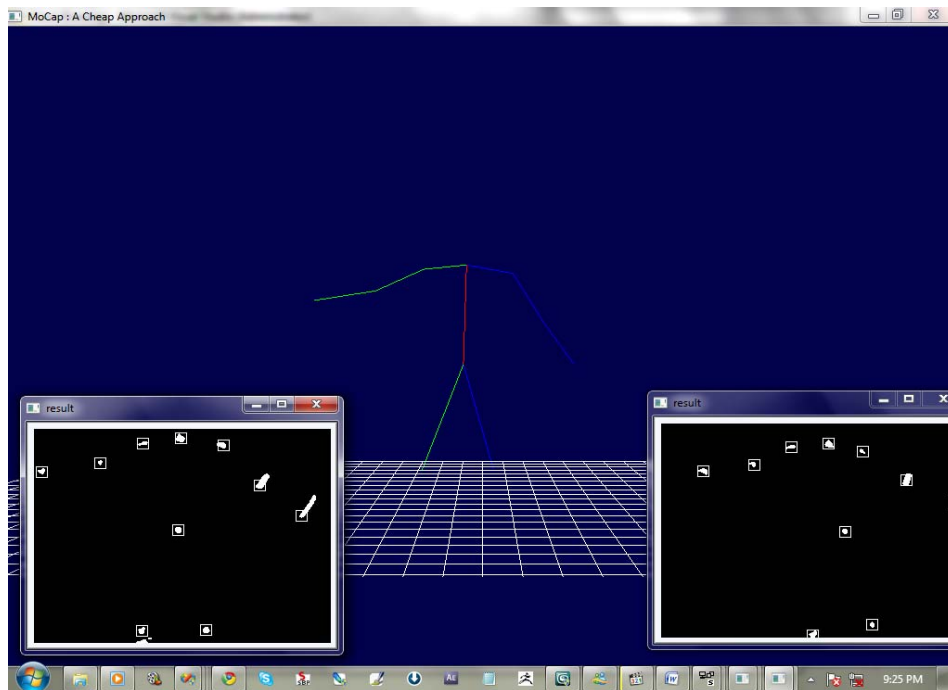


Figure 6.7 Motion Representation is Accurate Enough

The figure 6.7 shows 3d rendering performed at the server side. As is evident from the figures the 3d representation is right enough.

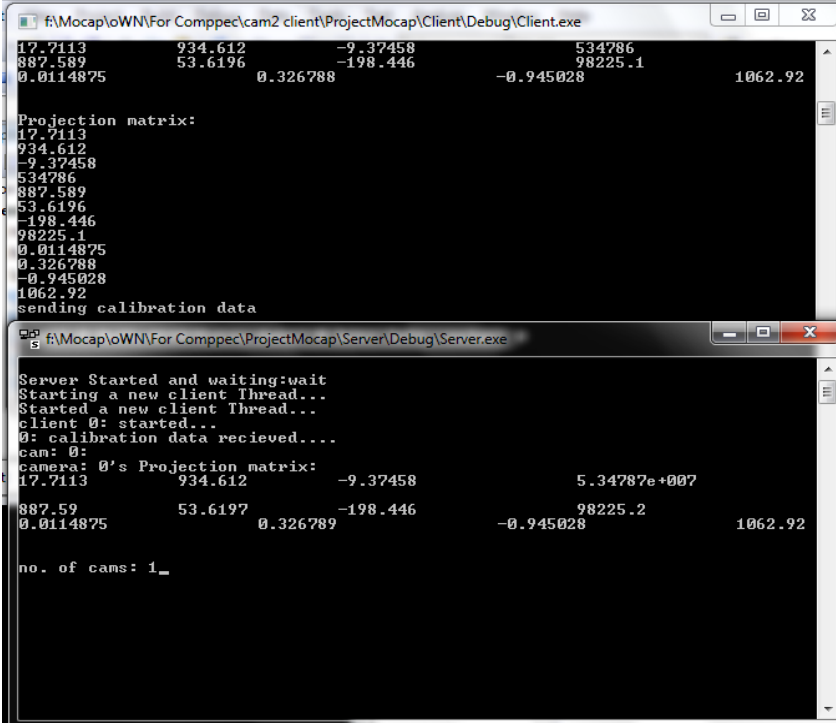
6.2.3 Integration and System Testing

Details about integration and system testing are below.

6.2.3.1 Data Transfer

The issue of faulty data transfer was expected to occur between server and the client as a lot of data transfer has to occur and it should be accurate.

Calibration Data: A float array of 16 elements is sent across networks. By displaying data on both sides it was tested successfully as in figure 6.8.



The image shows two overlapping terminal windows. The top window is the client's terminal, and the bottom window is the server's terminal. Both display a 4x4 projection matrix of 16 float values. The client window shows the matrix being sent, and the server window shows the matrix being received, with a slight difference in the second column's second element (53.6196 vs 53.6197).

```
f:\Mocap\oWN\For Compec\cam2 client\ProjectMocap\Client\Debug\Client.exe
17.7113      934.612      -9.37458      534786
887.589      53.6196      -198.446      78225.1
0.0114875    0.326788    -0.945028    1062.92

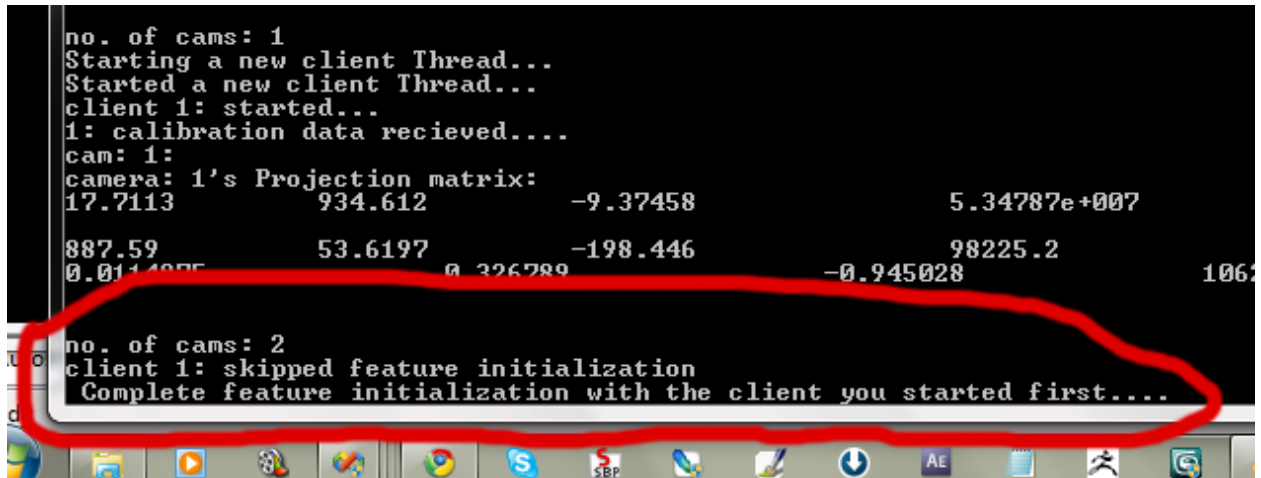
Projection matrix:
17.7113
934.612
-9.37458
534786
887.589
53.6196
-198.446
78225.1
0.0114875
0.326788
-0.945028
1062.92
sending calibration data

f:\Mocap\oWN\For Compec\ProjectMocap\Server\Debug\Server.exe
Server Started and waiting:wait
Starting a new client thread...
Started a new client thread...
client 0: started..
0: calibration data recieved...
cam: 0:
camera: 0's Projection matrix:
17.7113      934.612      -9.37458      5.34787e+007
887.59      53.6197      -198.446      78225.2
0.0114875    0.326789    -0.945028    1062.92

no. of cams: 1_
```

Figure 6.8 Calibration Data in Client (up) and Server (Lower)

Initialization Data: This was tested in the same way. An extra thing to test at this point was that server should allow initialization from any and only one client. Other clients should just start sending tracking data once initialization is complete as in figure 6.9.



```
no. of cams: 1
Starting a new client Thread...
Started a new client Thread...
client 1: started...
1: calibration data recieved....
cam: 1:
camera: 1's Projection matrix:
17.7113      934.612      -9.37458      5.34787e+007
887.59      53.6197      -198.446      98225.2
0.0114075      0.326789      -0.945028      1062

no. of cams: 2
client 1: skipped feature initialization
Complete feature initialization with the client you started first....
```

Figure 6.9 Shows Server Allowing Only One Client to Initialize

Tracking Data: Data was a float array of 24 values it was tested just by comparing the values on both sides and was successful.

6.2.3.2 Rendering

This was performed in real environment with three PCs. A server and two clients. Connection was on LAN.



Figure 6.10 Environment Used While Testing

Environment for the actor is as per requirement as shown in the figure 6.10 and final result in figure 6.11

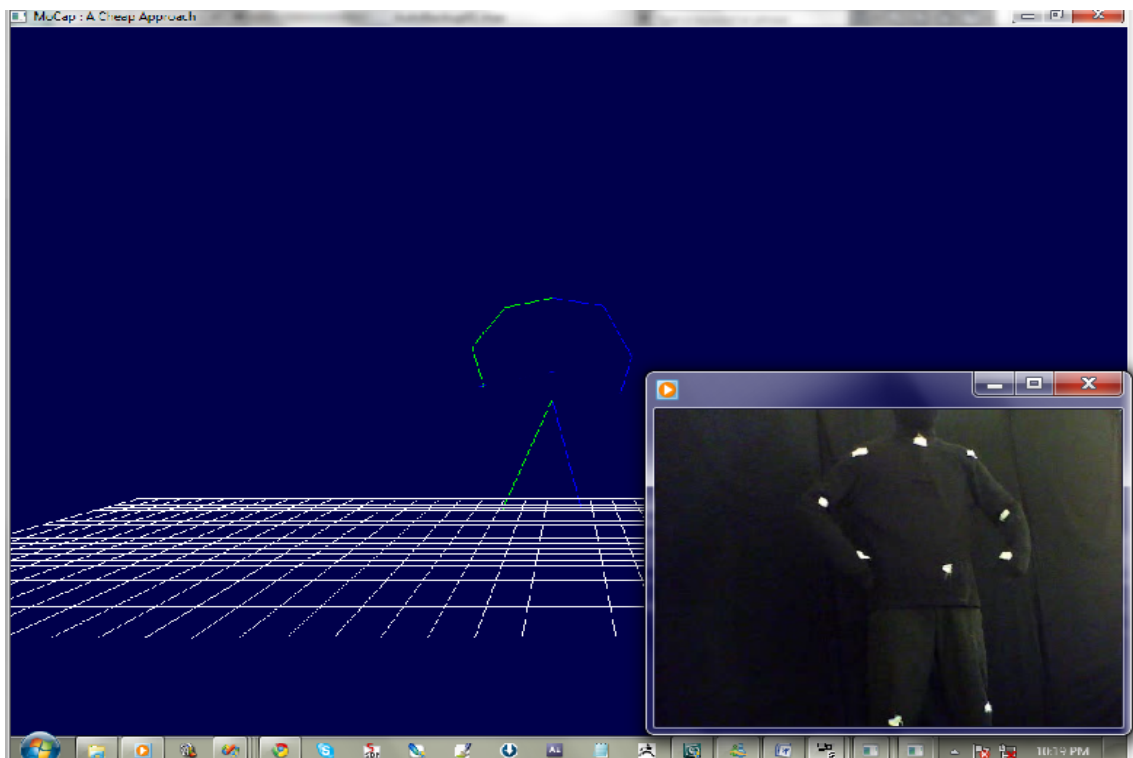


Figure 6.11 Motions Being Generated in Real Time

6.3 Systems to be compared

Only Optical MOCAP systems to be considered with the system.

6.3.1 Optical Motion Capture by Phase Space

This is a motion capture system produced by Meta Motion. Meta Motion was founded in 1999 by Lee Dickholtz, an animator whom has worked professionally with 3D animation and production since 1986. Prior to founding Meta Motion Mr. Dickholtz was the Vice President of Multi-Media development for CADCrafts / ID8 Media / Ideate Inc. Mr. Dickholtz helped take CADCrafts from verging on loosing their Kinetix dealership to being the world's second largest dealer for 3D Studio MAX.

6.3.2 Optotrak Certus Motion Capture System

This is produced by Northern Digital Inc. specializes in designing and manufacturing **research-grade motion capture systems** as opposed to animation-grade motion capture systems. The Optotrak Certus is a research-grade motion capture system.

6.3.3 Vicon – Motion Capture Services

Vicon is the new name for the combined MOCAP services provided by Vicon Motion Systems and Peak Performance Inc.

Vicon was established in Oxford, UK in 1984 as a privately owned company and grew from organically generated profits, establishing itself as a world leader in its core business of motion capture and analysis. Peak Performance Inc was established in

Colorado, USA in 1984. The combined Vicon offers an integrated solution for both digital and video based motion tracking.

6.4 Comparison

Table 6.1 Comparative Analysis

Attribute	Phase Space	Opto Track	Vicon	The System
Markers	Active LEDs	Opto Track Smart Markers	Active LEDs	White Passive circles(any material)
Camera Resolution	12 Megapixels	10 Megapixels	16 Mega Pixels	5 Megapixels
Capture Rate	480 fps	460 fps	120 fps	15 fps
Camera Price	\$230	\$118	\$5000	\$29
Learning Curve	1 day training	1 hr	Few minutes	Instant
Environment Flexibility	Can be setup anywhere	Indoor Studio	Indoor	Indoor Studio
Scalability	Yes	Yes	No	Yes
Multiple Object Tracking	No	No	Yes	No
Occlusion Handling	Yes	Yes	Yes	no
Direct Link to Graphics Renderer	No	No	No	Yes
Real Time	No	Yes	No	Yes
Compatible with VR	No	Yes	No	Yes
Accuracy	95%	93%	97%	92%
Max. Cameras	up to 24	Up to 3	Up to 244	Up to 2
Max Markers	Unlimited	8	Unlimited	9
Price	Around \$100,000	Around \$15000	Around \$250,000	Around \$150

As shown in the table, the system is not up to competition according to hardware aspects like camera resolution and capture rate but that was one of the goals i.e. to use minimal hardware costs. This makes the system to suffer on the software side but still the system gives a stiff competition in the results of the overall system performance.

6.5 Conclusion

The presented system is simple yet effective and true to its goal that is enabling real time motion capture. Considering low development costs and small development time, qualitatively it is not that marvelous but it will get the job done none the less and in much less price than the systems it is up against.

CONCLUSION

7.1 Introduction

Here a conclusion is presented on the effort put into this project linking the effort to the results followed by future prospects of this project.

7.2 Conclusion

Motion capture system was developed in seven months with the features described below. Object motion tracking through markers placed at motion points of a body was implemented successfully. System can perform tracking of up to ten individual motion points per frame. Tracking occurs in real time (up to 120 fps).

System is implemented on a client server architecture thus allowing it to render and track on separate machines hence causing resource distribution which effects in real time motion translation. It also 3d scene rendering which represents actual motion performed by the actor in real time. System has easy to use interface and presently no learn ability issues.

Issues related to the system are as follows. More cameras cannot be added hence environment is restricted. Occlusion handling has not been implemented.

System is completely implemented in OOP/C++ so it carries the qualitative features of OOP and integrity of a strong language such as C++. It is integration ready in a VR motion simulation. It is much less expensive than its competitor systems in the market. The system is totally **portable**. It can be setup anywhere.

7.3 Future Work

With the basic frame work developed in this project, many different directions can be taken and quite wonderful projects can be envisioned. Following are the future prospects of Motion Capture system hence developed.

A system similar to Nintendo Wii gaming station employing optical markers can be developed by optimizing tracking a little. It can create a market for itself and be a good addition to Pakistani industry.

It can be used to create VR simulations but tracking needs to be heavily optimized and good fps cameras will also be needed. This VR system can be used for virtual training in almost every institution. It can be a huge budget saver for organizations like Army, Air Force, Navy, Police, Heavy machinery complexes, flying clubs etc. which rely heavily on good trained individuals.

Using this motion capture system, gesture controlled systems can be developed which means machines be controlled by human gestures. This will increase usability and machine-friendliness. The field of human computer interaction will certainly benefit from an endeavor like this.

Applications can be developed using this system which will enable doctors to study their patients more accurately. Those will be patients suffering from some physical impairment.

Microsoft is currently developing an add on for its Xbox 360 called kinect. It will be used to create an immersive gaming experience using a special kind of camera. Similar kind of entertainment system can be developed using this project, but cameras will have to be upgraded.

BIBLIOGRAPHY

1. History of MOCAP Pg 2-8. MOCAP for Artists Workflow and Techniques for Motion Capture by Midori Kitagawa and Brian Windsor.
2. Maria Matzer, "Animation's New 'Toon Advances Mean 'Motion Capture' is about to Make a Splash," Los Angeles Times (8 Sept 1997), n.p. (business section)
3. La Trobe University, "Applications of Motion Capture"
4. Motion Capture by Maureen Furniss
5. Arne Henrichsen. 3D Reconstruction and Camera Calibration from 2D Images
6. R. Hartley and J.L. Mundy. The relationship between photogrammetry and computer vision. In E.B. Barrett and D.M. McKeown, editors, *SPIE Proceedings*, volume 1944 of *Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision*, pages 92–105. SPIE Press, September 1993.
7. O. Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? *Computer*
8. *Vision-ECCV'92*, Springer Verlag, *Lecture Notes in Computer Science*, 588:563–578, 1992.
9. Z. Zhang. Determining the Epipolar Geometry and its Uncertainty: A Review. *The International Journal of Computer Vision*, 27(2):161–195, March 1998. Also Research Report No.2927, INRIA Sophia-Antipolis.

10. L. Kitchen and A. Rosenfeld. Gray-level corner detection. *Pattern Recognition Letters*,1(2):95–102, December 1982.
11. C.J. Taylor and D.J. Kriegman. Structure and Motion from Line Segments in Multiple Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1021–1032, November 1995.
12. M. Pilu. Uncalibrated Stereo Correspondence by Singular Value Decomposition. Technical Report HPL-97-96, Digital Media Department, HP Laboratories Bristol, August 1997.
13. Z. Zhang, R. Deriche, O. Faugeras, and Q.-T. Luong. A Robust Technique for Matching Two Uncalibrated Images Through the Recovery of the Unknown Epipolar Geometry.
14. *Artificial Intelligence Journal*, 78:87–119, 1995. Also Research Report No.2273, INRIA Sophia-Antipolis.
15. O. Faugeras. Stratification of 3-D vision: projective, affine, and metric representations.
16. *Journal of the Optical Society of America*, 12(3):465–484, March 1995.
17. D. Liebowitz and A. Zisserman. Metric Rectification for Perspective Images of Planes.
18. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 482–488, 1998.
19. M. Pollefeys. *Self-Calibration and Metric 3D Reconstruction from Uncalibrated Image Sequences*. PhD thesis, ESAT-PSI, K.U. Leuven, 1999.

20. R. Hartley. Kruppa's Equations Derived from the Fundamental Matrix. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):133–135, February 1997.
21. M.I.A. Lourakis and R. Deriche. Camera Self-Calibration Using the Singular Value Decomposition of the Fundamental Matrix: From Point Correspondences to 3D Measurements.
22. Technical Report 3748, INRIA Sophia Antipolis, Project Robotvis, 1999.
23. C. Zeller and O. Faugeras. Camera Self-Calibration from Video Sequences: the Kruppa Equations Revisited. Technical Report 2793, INRIA Sophia Antipolis, Project Robotvis, 1996.
24. B. Caprile and V. Torre. Using Vanishing Points for Camera Calibration. *International Journal of Computer Vision*, 4:127–140, 1990.
25. D. Liebowitz, A. Criminisi, and A. Zisserman. Creating Architectural Models from Images. In *Proc. EuroGraphics*, volume 18, pages 39–50, September 1999.
26. D. Liebowitz and A. Zisserman. Metric Rectification for Perspective Images of Planes.
27. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 482–488, 1998.
28. D. Liebowitz and A. Zisserman. Combining Scene and Auto-calibration Constraints. In *Proc. 7th International Conference on Computer Vision, Kerkyra, Greece*, pages 293–300, September 1999.
29. Z. Zhang. A Flexible New Technique for Camera Calibration. Technical Report MSRTR-98-71, Microsoft Research, December 1998.

30. R. Mohr and B. Triggs. Projective Geometry for Image Analysis. In *International Symposium of Photogrammetry and Remote Sensing*, Vienna, July 1996.
31. M. Pollefeys, R. Koch, and L. van Gool. A simple and efficient rectification method for general motion. In *Proc. 7th International Conference on Computer Vision, Kerkyra, Greece*, pages 496–501, September 1999.
32. S. Roy, J. Meunier, and I. Cox. Cylindrical Rectification to Minimize Epipolar Distortion.
33. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 393–399, 1997.
34. Fusiello, E. Trucco, and A. Verri. Rectification with unconstrained stereo geometry.
35. In A.F. Clark, editor, *Proceedings of the British Machine Vision Conference*, pages 400–409. BMVA Press, September 1997. Also Research Memorandum RM/98/12, 1998,
36. Department of Computing and Electrical Engineering, Heriot-Watt University, Edinburgh, UK.
37. Fusiello, E. Trucco, and A. Verri. A compact algorithm for rectification of stereo pairs.
38. *Machine Vision and Applications*, 12(1):16–22, 2000.
39. R. Koch. Automatic Reconstruction of Buildings from Stereoscopic Image Sequences.
40. *Proceedings of Eurographics '93*, 12(3):339–350, September 1993.

41. R. Koch. 3-D Surface Reconstruction from Stereoscopic Image Sequences. In *Proc. 5th International Conference on Computer Vision, Cambridge, MA., USA*, pages 109–114, June 1995.
42. Fusiello, V. Roberto, and E. Trucco. Efficient stereo with multiple windowing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 858–863, June 1997.
43. http://en.wikipedia.org/wiki/Kalman_filter#Overview_of_the_calculation
44. Wei Zhang, Sui Wei: A Simple Method for 3D Reconstruction from Two Views
45. Zhang: Flexible Camera Calibration by Viewing a Plane from Unknown Orientations
46. <http://www.metamotion.com/about/meta-motion-history.html>
47. <http://www.ndigital.com/lifesciences/certus-motioncapturesystem.php>
48. <http://www.vicon.com/company/>