REAL TIME IMAGE AND VOICE PROCESSING INPUT FOR ONLINE MULTIPLAYER COMPUTER GAMES

A Thesis

Submitted to the Department of Computer Science and Engineering

of

BRAC University

by

Syed Shaiyan Kamran Waliullah

Student ID: 09301019

Md. Risul Karim

Student ID: 09301002

In Partial Fulfillment of the

Requirements for the Degree

of

Bachelor of Science in Computer Science

December 2012

DECLARATION

I hereby declare that this thesis is based on the results found by myself. Materials of work found by other researcher are mentioned by reference. This thesis, neither in whole nor in part, has been previously submitted for any degree.

Signature of Supervisor	Signature of Author

ACKNOWLEDGMENTS

Special thanks to Insan Arafat for helping us with the idea and also Imran Farid for helping us in graphical elements. We would also like to thank Dr. Khalilur Rahman for helping us in our robotic project. We also would like to thank Nayak Amir Arup for helping us with OpenCV.

ABSTRACT

Real Time image processing and voice processing in computer gaming is a very recent topic as companies like Microsoft and Sony have researched and implemented this technology recently in their gaming consoles XBOX-360 and Play Station 3 respectively. As these consoles have high performance processor likes CELL or XENON with POWER (Performance Optimization with Enhanced RISC) PC Architecture unlike desktop computers which uses x86 and legacy architecture processor which is slower than console's processors. Gaming with "Natural User Interface" such as gestures and voice was the main focus of the research of this project which is named as "Project: Control Scheme Revolution". To materialize this challenge, several approaches were carried out. An open source game engine JMonkey Engine 3 (JME3) was customized with natural user interface. To measure the goodness of the system which is being developed by this project, Frame per Second (FPS) of the game was kept in account. All the approaches were taken to integrate different libraries to the core of the Game Engine ensuring a playable FPS for good user experience. Besides that, this research is also focused on implementing this system on the Multiplayer and Massively Online Multiplayer games as the participation of the more than one user in gaming produces an extra over head on the overall processing of the computer. As not all computers are powerful enough to support all of these systems at real time effortlessly, this research also focused on developing a system to render and stream a game from high performance computer to a moderate or low performance device such as note book or android device. Performance analysis of all of these approaches to develop a system which will bring the feeling of gaming in a console to the gaming in an ordinary computer is the main goal of this project.

TABLE OF CONTENT

Contents:
Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
INTRODUCTION

1.1 Real time image and voice processing overview

Real time image processing and voice processing (also known as speech recognition) has been used separately for long time in different fields of science. As subcategory of the field of Digital Signal Processing, digital image processing has broad advantages over analogue image processing. It permits mathematicians to apply wider range of algorithms to apply on the digital data and can avoid problems such as the buildup of noise and signal distortion during processing. This research used the information from images that are taken in real time for detecting various objects by which we can perform multi-modal operation with computer. This creates a Natural User Interface where a human can be physically part of a virtual world. Voice processing or Speech recognition in real time has been also used in several fields such as health care, battle management, vehicles etc. Integration of image processing and speech recognition in gaming consoles such as XBOX 360 and Play Station 3 is considered as revolution as this integration has been well accepted by users. Sony Play station 3 uses a high speed camera called PS3 Eye and XBOX 360 offers Kinect Sensor.

For image processing in real time, a camera is required to grab the image of the real world and process it to find out the object of interest. Based on the existence and location of the object of interest, changes in the command for controlling the character in gaming world are done.

Speech recognition in real time is also involves the processing of digital signal that are found from the microphone and extract the subjected language model. Three main challenges of speech recognition system are Speaker Independence, Continuous Speech and Large vocabulary. A good speech recognition system must deal with all of these issues.

Combination of all of these components together is the main goal of this project. For this research paper, this project will be named as Project Control Scheme Revolution or Project C.S.R.

1.2 System Integration overview

The system that this project is willing to produce is a customized game engine which we will be enabled to produce games with feature of speech recognition and image processing at the same time and will also support the networking feature. As the personal computers are not as strong as consoles and with limited processing capabilities, the speech recognition and image processing libraries are required to be as optimized as possible. The three main part of this system are a game Engine, Computer Vision Library and Speech Recognition Library.

While developing this system, it was focused on choosing a game engine first that is easy to customize and open source. JMonkey Engine is an open source project, programmed in java and build on OpenGL which was selected to customize.

For image processing, several ways were tried to detect several object of interest such as human face, hands, shoulder etc. Open Computer Vision (OpenCV) library was chosen to merge with the game engine as it is highly optimize and has a java wrapper called JavaCV.

For speech recognition library, CMU Sphinx was chosen as it is entirely written on java, which is also optimized enough to be processed with the game and the computer vision library at the same time. Therefore the components of the system are as follows:

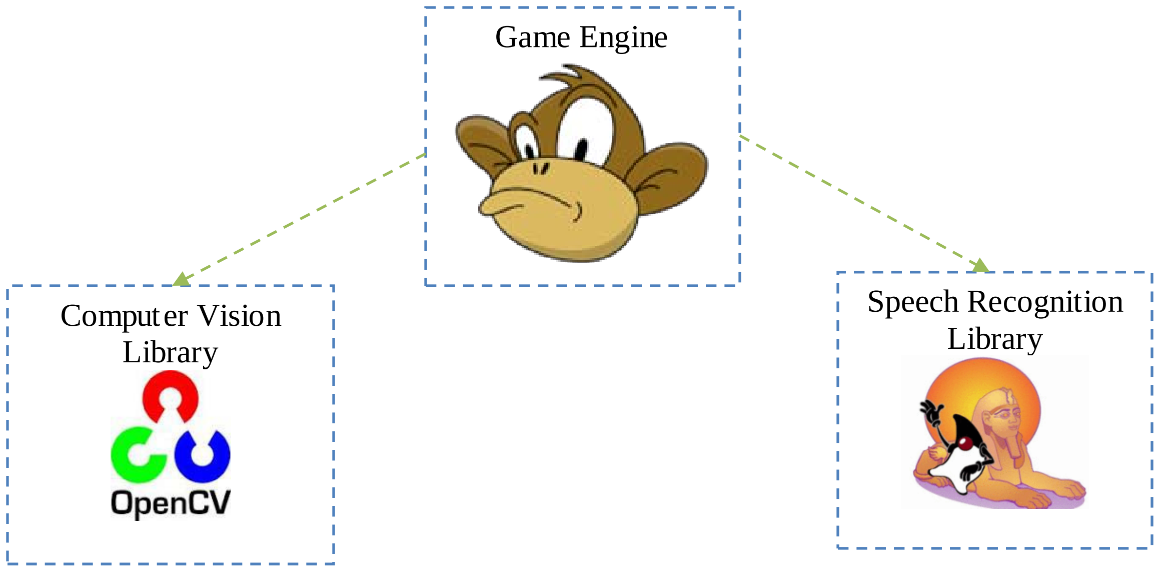


Fig 1.1 System Integration overview

1.3 Existing research and implementation

The two existing technology that has similarity with the project: Control Scheme Revolution. XBOX 360 Kinect by Microsoft and PlayStation 3 Move by Sony, both released in 2010.

Kinect is a motion sensing input device by Microsoft for the Xbox 360 video game console. Based around a webcam-style add-on peripheral for the Xbox 360 console, it enables users to control and interact with the Xbox 360 without the need to touch a game controller, through a natural user interface using gestures and spoken commands. The project is aimed at broadening the Xbox 360's audience beyond its typical gamer base . The device features an "RGB camera, depth sensor and multi-array microphone running proprietary software", which provide full-body 3D motion capture, facial recognition and voice recognition capabilities.

PlayStation Move is a motion-sensing game controller platform for the PlayStation 3 (PS3) video game console by Sony Computer Entertainment (SCE). Based around a handheld motion controller wand, PlayStation Move uses the PlayStation Eye camera to track the wand's position, and inertial sensors in the wand to detect its motion. The PlayStation Move motion controller features an orb at the head which can glow in any of a full range of colors using RGB light-emitting diodes (LEDs). Based on the colors in the user environment captured by the PlayStation Eye camera, the system dynamically selects an orb color that can be distinguished from the rest of the scene. The colored light serves as an active marker, the position of which can be tracked along the image plane by the PlayStation Eye. The uniform spherical shape and known size of the light also allows the system to simply determine the controller's distance from the PlayStation Eye through the light's image size, thus enabling the controller's position to be tracked in three dimensions with high precision and accuracy. The sphere-based distance calculation allows the controller to operate with minimal processing latency, as opposed to other camera-based control techniques on the PlayStation 3.

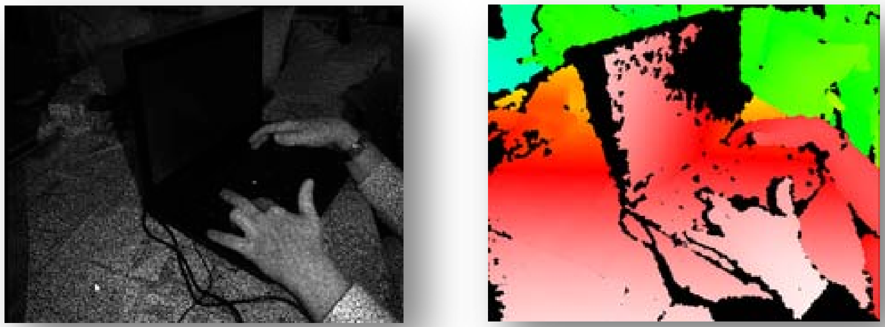


Fig1.2 Depth Sensing using Kinect and PS3 Move Wand

CHAPTER II

REAL TIME IMAGE PROCESSING APPROACH

2.1 Overview of Technologies

Before merging with Game Engine for this project, two image processing technologies were tested before implementation. They are Java Media Framework (JMF) and Open Computer Vision (OpenCV) library by intel and later Willo Garage.

The Java Media Framework (JMF) is a Java library that enables audio, video and other time-based media to be added to Java applications and applets. This optional package, which can capture, play, stream, and transcode multiple media formats, extends the Java Platform, Standard Edition (Java SE) and allows development of cross-platform multimedia applications. As this framework allowed us to interface webcam with it, the first initiative to integrate computer vision with the game engine was carried out with this library.

The second approach was undertaken for image processing part of the system using the library OpenCV. OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate it.

In the section 2.2, section 2.2.1 and section 2.1.2, performance analysis of these libraries with the implemented algorithm are discussed and in section 2.3 will discuss the consistency of the suggested technology.

2.2 Performance analysis of different libraries

In this section we discuss about the performance of two libraries, JMF and OpenCV.

2.2.1 Java Media Framework 2.1.1

Java Media Framework is a java library which enables to use real time images or videos in java applet. For image processing in Project-C.S.R, JMF was first used. The rendered real time image is taken from the JMF and that image is used for image processing. For finding a specific object in the image the shape, size and color of the object is used. Blue-LED lights were first used to be detected in Project-C.S.R. Now, for detecting the light the image had to go through some processes for finding the light. The first process that the image goes through is that all of its green and red pixels are removed. The reason for this is to detect the blue color in the Blue-LED light by differentiating it from other colors and to remove these red and green pixels a threshold value is used for both the colors. Once the red and green pixel crosses the threshold value the pixel at that point is converted to black. Below is the pseudocode for removing green and blue pixel values.

```
removeRedGreenPixel(){
    1.  TRed = Threshold Value of Red
    2.  TGreen = Threshold Value of Green
    3.  Height = height of the image
    4.  Width = width of the image
    5.  x = 0 ← Current Value of pixel in x coordinate
    6.  y = 0 ← Current Value of pixel in  y coordinate
    7.  While x < Width
    8.     While y < Height
    9.          if getGreen (x, y) > TGreen &  getRed(x, y) > TRed
    10.           imageSetColor(x, y) = 0 ← making that pixel black
    11.         else
    12.           Nothing
    13.         y = y + 1
    14.    x = x + 1
    }
```

This process ensures that most of the green and red colors are removed but it still does not remove all of them. So, after the remove of the red and green pixel process the image then goes through another process which only keeps pixels with blue value. There is a threshold value for finding the blue value of the pixel. If the value of the pixel is below the threshold value then that pixel is turned to black by replacing its value with 0. Below is the pseudocode for keeping blue only.

```
blueOnly(){
   1.  TB = Threshold value of blue.
   2.  Height = height of the image.
   3.  Width = width of the image
   4.  x = 0 ← Current value of pixel in x coordinate
   5.  y = 0 ← Current value of pixel in y coordinate
   6.  While x < Width
   7.      While y < Height
   8.          If getBlue(x, y) > TB
   9.              imageSetColor(x, y) = 0 ← making that pixel black
   10.         else
   11.             imageSetColor(x, y) = 1 ← making that pixel white
   12.         y = y + 1
   13.     x = x + 1
   }
```

After this process it has been ensured that almost all of the other colors except for the Blue-LED light blue has been removed. Now after the all these processes had been done a small recursive algorithm starts which searches for the edge of the blue light. The algorithm scans the entire image pixel by pixel. During scanning whenever it finds the blue color value it then starts the algorithm. Below is the pseudocode for finding the edge of the blue light.

```
findBlueEdge(x, y, Width, Height){
    1.  If x < 0
    2.    Return
    3.  If y < 0
    4.    Return
    5.  If x > Width
    6.    Return
    7.  If y > Height
    8.    Return
    9.  If getPixelValue(x, y) equals to 2
    10.   Return
    11. If getPixelValue(x, y) equals to 0
    12.   Return
    13. If x > maxX
    14.   maxX = x
    15. If x < minx
    16.   minx = x
    17. If y > maxY
    18.   maxY = y
    19. If y < minY
    20.   minY = y
    21. getPixelValue(x, y) = 2
    22. findBlueEdge(x, y + 1, Width, Height)
    23. findBlueEdge(x + 1, y, Width, Height)
    24. findBlueEdge(x - 1, y, Width, Height)
    25. findBlueEdge(x, y-1, Width, Height)
    26. return
    }
```

The findBlueEdge finds the edge of the blue light and then marks a square around it. After marking it with square, the center of the square is the position of the blue light.

Technology Used: For taking the image a normal Web-Cam was used which had a FPS of 30.

2.2.2 Open Computer Vision 1.0

OpenCV library allows programmers to detect objects of interest during real time image processing. The good thing about OpenCV is the way to train to find a specific object and it can also be used in java applets. OpenCV was later used in this project. The algorithm that is used from OpenCV is ‘haar’. For using the OpenCV a lot of positive and negative images are required. For clarification, positive images are those images with the object of interest in the image. Negative images are those images without the object of interest in the images. Later these images are used in haartraining. The haartraining uses these images for making accurate detection. It compares all the positive images with all the negative images. Also in haartraining the number of stages must be specified. The stages represents how many times the comparison will occur. The more stages are specified the more accurate the detection will be but the down fall is that more time it will take to finish the training. Once the training has been finished an xml file is created. This xml file contains the result of the haartraining. This xml file is used with OpenCV to find the specific object.

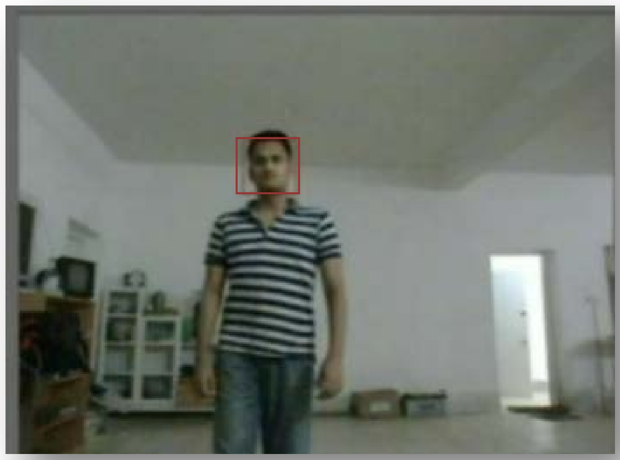


Fig. 2.1 Example of Showing Face Detection in OpenCV

Technology Used: For taking the image a Sony PS3Eye was used which has 60 – 120 Frame Per Second.

2.3 Consistency of the Proposed Technology

The library which Project-C.S.R is using is OpenCV. The reason for using OpenCV over JMF is plain and simple, OpenCV is faster and much more accurate than the JMF process. OpenCV saves a lot of processing power during image processing where as JMF does not. Due to this a lot of FPS are saved in games and other application related to games. The technology used in this project is Sony PS3Eye because PS3Eye has higher FPS than most of other Web-Cams.

CHAPTER III

REAL TIME SPEECH RECOGNITION APPROACH

3.1 Speech Recognition Library

For Speech Recognition system, CMU Sphinx Library was chosen. CMU Sphinx is a speaker independent, continuous speech recognition system with large vocabulary. It is founded on discrete hidden Markov models (HMM’s) with Linear Prediction Model (LPC) derived parameters. To provide speaker independence, knowledge was added to sphinx in several ways such as, (i) multiple code books of fixed width parameters and (ii) enhanced recognizer with carefully designed models and word duration modeling. To deal with co articulation in continuous speech, yet still adequately represent a large vocabulary, two new sub word speech units was introduced (i) functions-word dependent phone model and (ii) generalized triphone models. With grammar perplexity 997, 60 and 20, Sphinx attained word accuracies of 71, 94 and 96 percent on a 997 work task.

Sphinx 4.0 library was integrated in JMoneky engine to make it enabled with voice recognition system. While gaming, both discrete word and sentences were required. Sphinx comes with large knowledge base of English words of 129247 words. Among them, specific words that are needed for games were included. The uses of Sphinx speech recognition library was done for performing two tasks, (i) For voice activated Natural User Interface and (ii) Non playable character control. Details on this topic will be discussed on Chapeter VI, section 5.3.

3.2 Performance Analysis of the Proposed library in Game Engine

Before adding Sphinx Voice Recognition library in game engine, it was needed to be tested on several computers to realize it's usability on as an input scheme for gaming. For test cases we choose five computers with following specification:

Table 3.1

Computer Configuration

PC	**Brand**	**Processor**	**RAM**	**GPU**
PC1	HP Compaq Note Book	2.2GHz Dual Core	2GB	700MB: Built in
PC2	Dell Inspiron Note Book	2.12GHz Core i3	2GB	800MB: Built in
PC3	HP Pavilion Laptop	2.00GHz Dual Core	4GB	358MB: Built in

In blank Screen Test, without Sphinx Voice recognition is activated, the computers show the following Frame Per Second in the blank game world.

Table 3.2

JME3 Blank Screen Frame per Second

PC	**PC1**	**PC2**	**PC3**
FPS	600	500	700

With Sphinx Activated, the process shows the following FPS

Table 3.3

Sphinx Activated JME3 Blank Screen FPS

PC	**PC1**	**PC2**	**PC3**
FPS	156	150	170

Without the blank Screen, on a same game, developed for this project, when the sphinx is activated, it shows the FPS in the following number on average.

Table 3.4

Sphinx Activated in game

PC	**PC1**	**PC2**	**PC3**
FPS	89	70	95

The ideal Frame Per Second for playing a game in Computer is around 30. A game is playable if the FPS is around 22 however lower than this FPS causes bad user experience. In XBOX 360 the Frame Per Second is around 28.5 and for PS3 is 29.3
After comparing Frame Per Second in all of the PC’s above, it can be concluded that in games with low processing power, it may be possible to play game using sphinx however user experience might not be very satisfactory.

CHAPTER IV

GAME ENGINE

4.1 An overview of existing game engines

Most of the game engines and frameworks of today are compatible with high graphics processing. To develop the process, it was required to choose a game framework that is easy to customize and has ability to adopt the integration of sphinx and openCV. Several game engines were studied before the customization. In section 4.2 and Section 4.3, a competitive discussion on different game engines and framework will be shown and in section 4.4 will hold the reason for using JMonkey engine will be discussed.

4.2 Game Engine Criteria

To choose the game engine for customization, focus was given to the following points

1. Integration ability with OpenCV and CMU Sphinx
2. Optimization
3. Abstraction Level
4. Visual Editor Support
5. Scripting and Programming Language
6. GPU
7. License Cost

A brief result of little internet research is shown in the following sections below.

4.3 Comparison among existing Game Engines and Frameworks

To compare between the frameworks and Game Engines, brief introduction of them can be presented.

4.3.1 XNA Framework:

The XNA Framework is based on the native implementation of .NET Compact Framework 2.0 for Xbox 360 development and .NET Framework 2.0 on Windows. It includes an extensive set of class libraries, specific to game development, to promote maximum code reuse across target platforms. The framework runs on a version of the Common Language Runtime that is optimized for gaming to provide a managed execution environment. The runtime is available for Windows XP, Windows Vista, Windows 7, Windows Phone 7 and Xbox 360. Since XNA games are written for the runtime, they can run on any platform that supports the XNA Framework with minimal or no modification. Games that run on the framework can technically be written in any .NET-compliant language, but only C# in XNA Game Studio Express IDE and all versions of Visual Studio 2008 and 2010 (as of XNA 4.0) are officially supported. Support for Visual Basic .NET was added in 2011. Other than all of these features, XNA is not free for publishing. Beside, both Sphinx and OpenCV java Wrapper (JavaCV) are written in Java where XNA is C# based. So XNA was not chosen for the project.

4.3.2 Unity 3D:

Unity is an integrated authoring tool for creating 3D video games or other interactive content such as architectural visualizations or real-time 3D animations. Unity's development environment runs on Microsoft Windows and Mac OS X, and the games it produces can be run on Windows, Mac, Xbox 360, PlayStation 3, Wii, iPad, iPhone and Android. The programming language is basically javaScript for game developer. As for this project java plays an important role, so Unity 3D was not selected.

4.3.3 Unreal Engine 3:

The third and current generation of the Unreal Engine (UE3) is designed for DirectX (versions 9-11 for Windows and Xbox 360), as well as systems using OpenGL, including the PlayStation 3, Mac OS X, iOS, Android, Stage 3D for Adobe Flash Player 11, Play station Vita and Wii U.Its renderer supports many advanced techniques including HDRR, per-pixel lighting, and dynamic shadows. It also builds on the tools available in previous versions. In October 2011, the engine was ported to support Adobe Flash Player 11 through the Stage 3D hardware-accelerated

APIs. However Unreal Engine runs on Unreal Script and maintains a very high level of abstraction so integrating a new library would have been an issue.

4.3.4 JMonkey Engine 3:

JMonkeyEngine (jME) is a game engine made especially for modern 3D development, as it uses shader technology extensively. jMonkeyEngine is written purely in Java and uses LWJGL as its default renderer. OpenGL 2 throughOpenGL 4 is fully supported.

jMonkeyEngine is a community-centric open source project released under the new BSD license. It is used by several commercial game studios and educational institutions. The default jMonkeyEngine 3 download comes readily integrated with an advanced SDK.

JMonkey Engine Showed the best possibility to be integrated with libraries like OpenCV and CMU Sphinx as all of them were written in java. Beside, JMonkey Engine is free of cost, easily customizable so it was chosen for the project.

4.4 Effectiveness of the proposed Game Engine

To show the effectiveness of JMonkey Engine, features of this engine can be discussed. JME3 comes with the JME3 SDK which is a full featured java and xml code editor. It has numbers of plugin and as an open source project, number of it’s plugins are increasing frequently. It can be deployed into all major Operating Systems such as windows, Mac, Linux and Android. It has Bullet Physics library which enables it to support complex physics calculation such as rigid body control, rag doll, Inverse Kinematics and vehicle physics. Finally, it comes up with Spider Monkey Network API which can be used to create multi player games.

OpenCV and Sphinx is merged together with the libraries of game engine. We took two computers with the following configuration to figure out the performance of the system.

Table 4.1

PC Configuration

PC4	**Desktop**	**2.66GHz Quad**	**4GB**	**1GB nVidia**
PC5	Desktop	3.30GHz Core i5	8GB	1GB ATI

Before and after integration of the OpenCV and CMU Sphinx, the FPS of a game, developed for this project was following.

Table 4.2

Sphinx with OpenCV Activated in game FPS

PC	**PC4**	**PC5**
FPS Before addition of OpenCV and Sphinx in game	600(approximate Value)	800(approximate Value)
FPS after addition of OpenCV and Sphinx	300(approximate Value)	400(approximate Value)

This data clearly shows that the game runs with enough frames per second to play with a good user experience. Two games were developed on JMonkey engine to test all the approaches to integrate Natural User Interface with them. Games are

1. Flight Simulator Game (Massively Multi player online game)
2. Run (Re creation of “Temple Run” game, human racing game)

CHAPTER V

NETWOKING

5.1 Multiplayer System Overview

This project uses JME3's SpiderMonkey which is a multiplayer feature. SpiderMonkey is a multiplayer feature for games so it is quite fast. Multiplayer and Streaming for Cloud Based Games are used in this project.

5.2 Spider Monkey Network Integration

Spider Monkey is the networking feature or multiplayer feature of JME3. It can use both TCP and UDP protocol. Since it is made for games it is fast too. The use of Spider Monkey is simple. Spider Monkey allows customized messages to be passed around the server. The type of protocol that will be used can also be specified in Spider Monkey.

5.3 Network Feature of the system

Two types of features are used in this project. They are Multiplayer Network and Streaming for Cloud Based Gaming.

5.3.1 Multiplayer Network

There are two parts in the multiplayer network, the server side and client side. The server side is a headless application which means it has no GUI. The server in this project sends messages from one client to another and also from the database. It also stores the position of each client in the database. The client side of the network only sends message to the server. The client sends its position to the server.

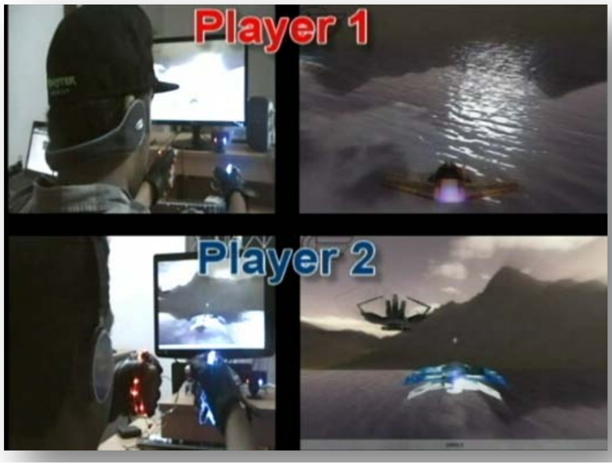


Fig. 5.1 Showing multiplayer using real time image processing

5.3.2 Streaming for Cloud Based Gaming

This feature is still in progress. In this project the video streaming works by sending snapshots from one computer to another. It breaks down the image by 25k bytes and sends each separate part separately. Then once after all the bytes have been received it then combines all the bytes and forms the image which is then rendered to be shown.

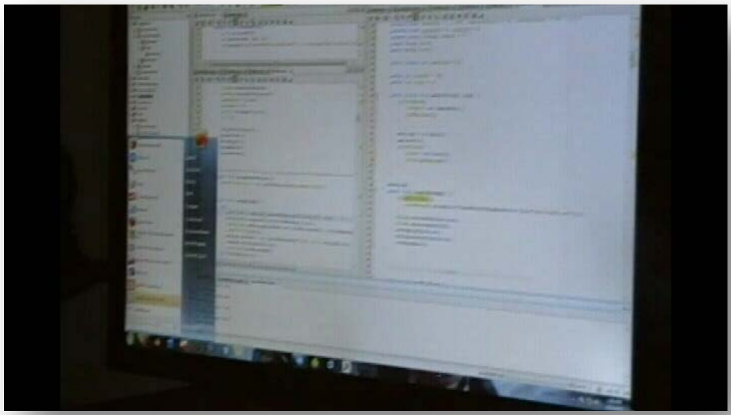


Fig. 5.2 Showing the state of a Computer

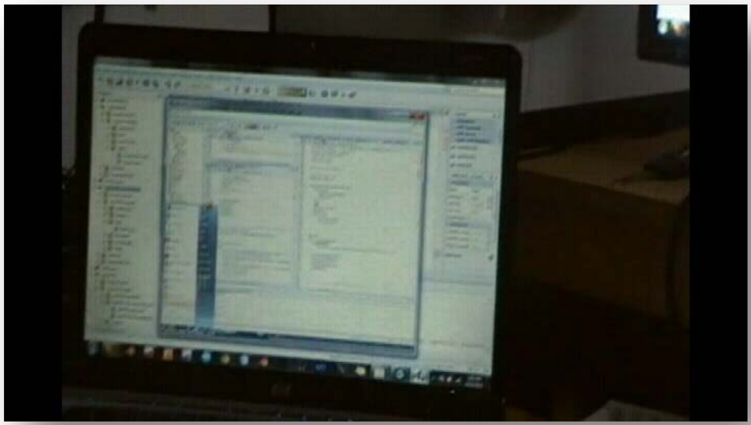


Fig. 5.3 Showing the state of the computer in this laptop

CHAPTER VI

NATURAL USER INTERFACE (NUI)

6.1 Image as input

In Project-C.S.R real time image processing is used as inputs. Three types of objects are used for image processing inputs they are fist, face and gloves with LED. Whenever these objects are found during image processing an event is specified to occur.

6.1.1 Fist Detection

The fist is used as an input. Through haartraining an xml file is created which is used to detect the fist part only. Around 1000 images in total were used to create this xml.

6.1.2 Face Detection

The face is used as an input. This is the default xml for the OpenCV.

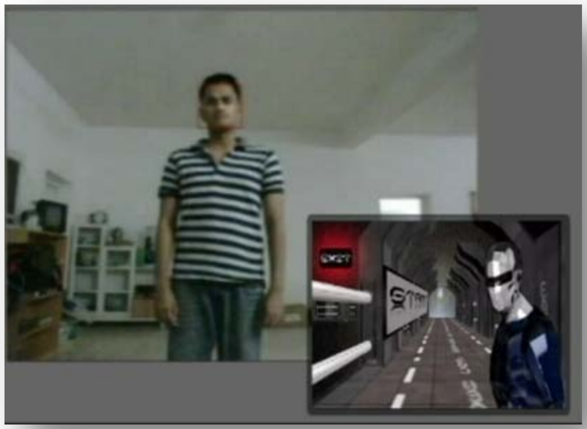


Fig. 6.1 Showing Face Detection in game

6.1.3 Gloves with LED Detection

Some LED are attached to gloves and that is used as an input. Through haartraining an xml file is created which is used to detect the object. Around 2500 images in total were used to create this xml.

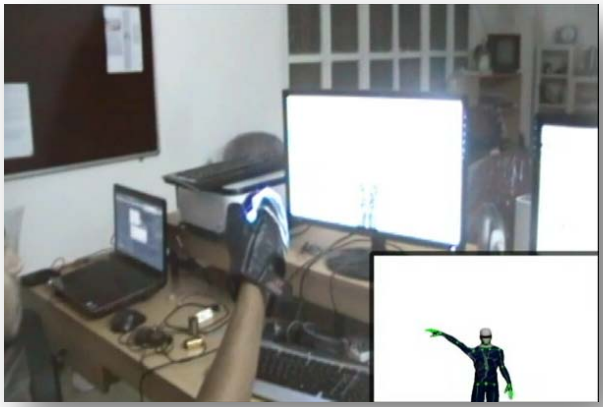


Fig. 6.2 Showing Gloves Detection in game

6.2 Algorithm for NUI

For different application different type of algorithm were used for creating an event from the NUI.

6.2.1 Avatar Control

For the Avatar Control the image taken is divided into three equal parts which are left, center and right. For this the Face Detection was used because to give a realistic feeling that the Avatar is looking at the user. When the user face happens to be at the left side of the image then the avatar rotates left, if the user happens to be at the right side of the image then the avatar rotates right and if at the center then it does nothing.

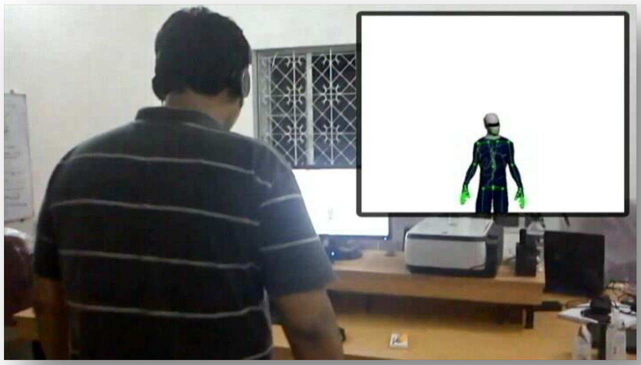


Fig. 6.3 Controlling the Avatar using Face Detection

6.2.2 Face detection Based Character Control

This uses face detection as input. The image is again divided into three parts, left, center and right. When the player happens to be at the left side the character then moves left, if at right then the character moves right and center for nothing. A threshold value for y coordinate is taken because if the player crosses that value then the character jumps. Here the size of the player’s face is also taken into account. There are two threshold values, maxFace which is the maximum size of a face and minFace which is the minimum size of the face. When the player face size crosses the maxFace then it means that the player is closer and moves the character forward. When the player face size is smaller than the minFace then it means that the player is further away and moves the character backward. Example of a game play detecting face is shown in figure 6.4. This game is called “Run” which was developed as part of this project.

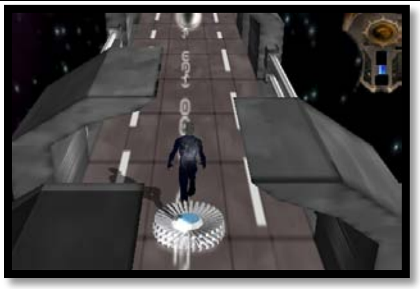
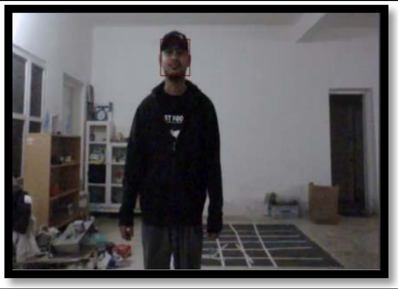
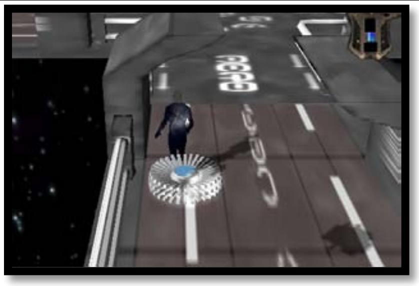
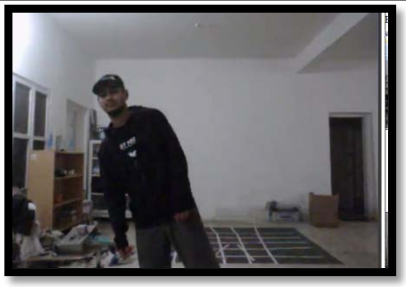
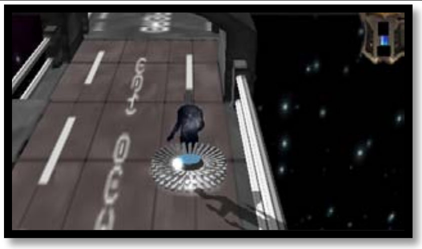
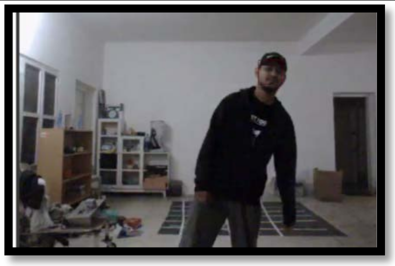
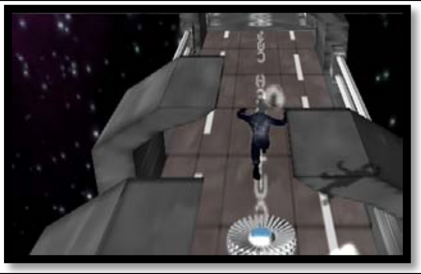
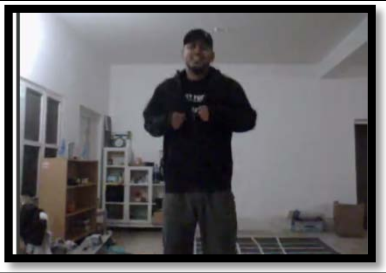
		The game character goes straight when the face is found at the middle of the screen and area is between minFace and maxFace.
		When player leans right, the face detected at right side of the screen, so game character goes left.
		When player leans left, the face detected at left side of the screen, so game character goes right.
		When player jumps, the face is detected at top part of the screen, the jump animation of the game character is triggered.

Fig 6.4: Face detection based character control and animation

6.2.3 Multi-Player Flight Simulation

This uses the gloves or fist detection as game inputs. It divides the image into five parts, left, right, top, bottom and center. When the gloves are at the center then it does nothing, when the gloves are at the left then the plane moves left, when the gloves are at the right side then it moves right, when the gloves are at the top then it moves up and when the gloves are at the bottom then it moves down.

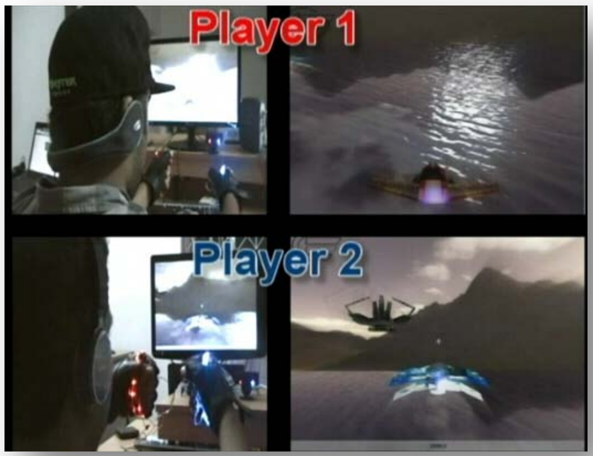


Fig. 6.5 Multi-player Flight Simulation

6.2.4 Mouse Cursor Control by Gesture

This uses gloves detection as input. Here it sees where the position of the mouse is at and the position of the gloves. The mouse is then moved to the direction of the gloves position.

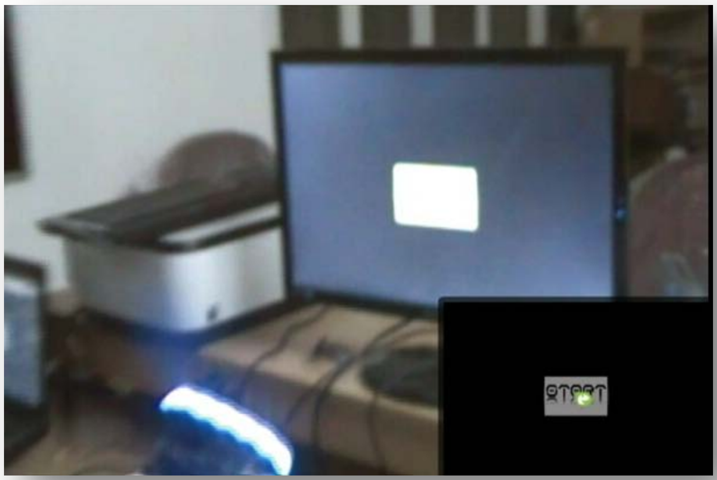


Fig. 6.6 controlling mouser cursor using Image Processing

6.2.5 Different Body Part Detection and Collaboration

To detect human movement, different human body part was detected by openCV and Haar Traning. Face, right Arm and Left Arm and Upper body was detected which was used in the games which were developed as part of the project. The detailed discussion of different movement of game character using these detected part are stated in the following segments.

6.2.5.1: Relative distance calculation between face and right hand for Flight Simulation Game

To control the game character of the Flight Simulation game, i.e a fighter plane, relative distance of the right hand and face was calculated. The figure 6.7 explains how this input works by the help of images from camera and game's screen shots.

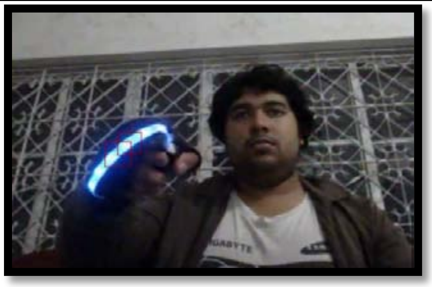
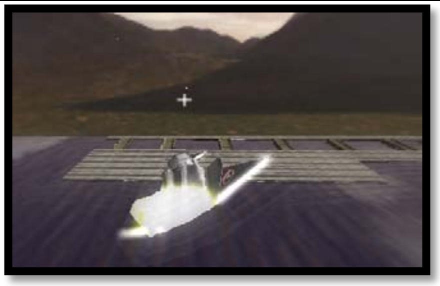
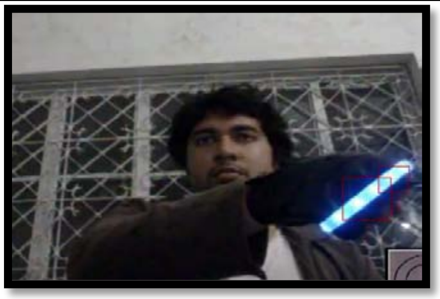
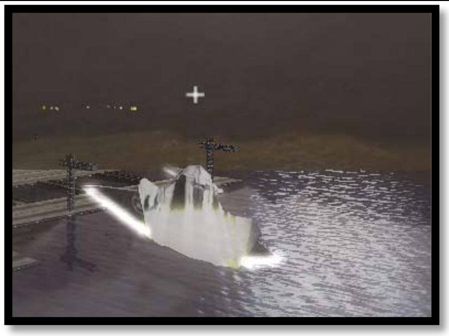
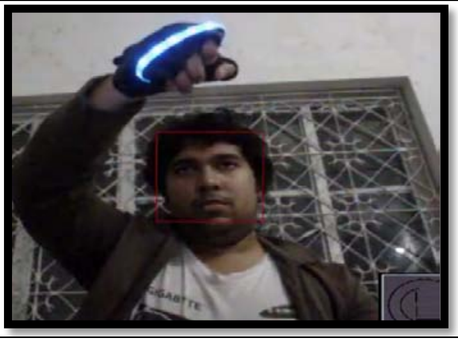
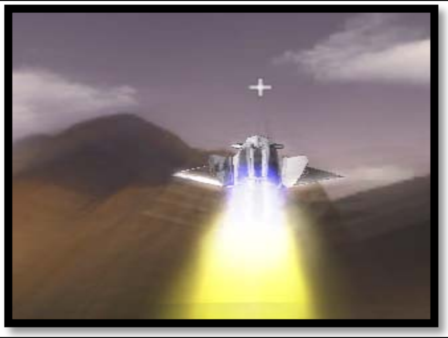
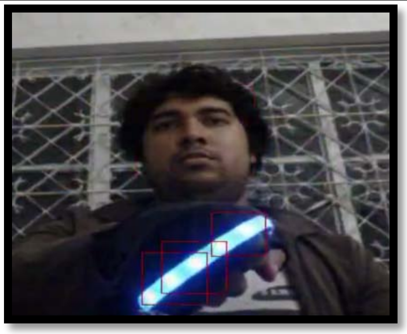
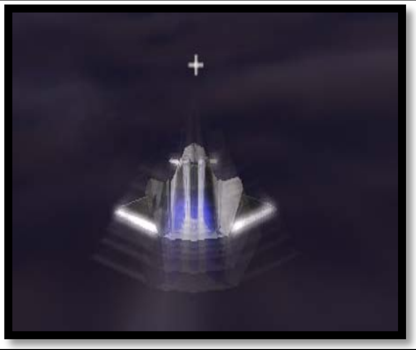
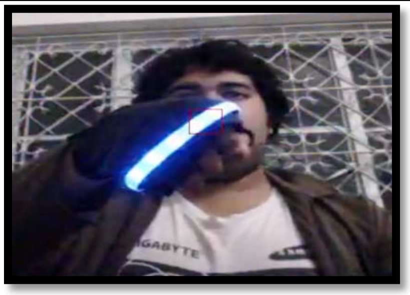
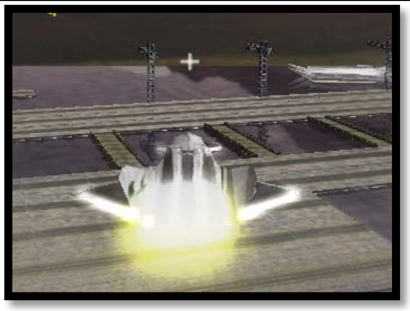
		When right hand is at the right side of the face, the plane moves in left direction.
		When right hand is at the left side of the face, the plane moves in right direction.
		When right hand is at the upward side of the face, the plane moves in upward direction.
		When right hand is at the bottom side of the face, the plane moves in downward direction.
		When the right hand is on the area covered by face, the plane goes forward.

Fig 6.7: Face and Hand's relative distance as input

6.2.5.2: Relative distance calculation between left and right hand for Flight Simulation Game

This input scheme detects both of the hands and by calculating their relative position, the game character is controlled. The figure below explains how the movement of game character synchronizes with hand gestures.

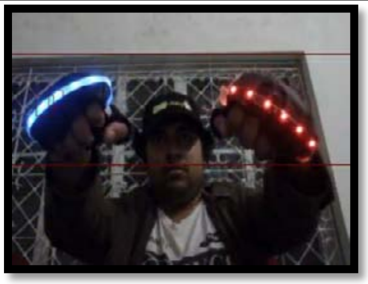
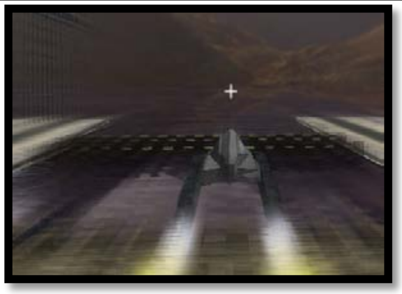
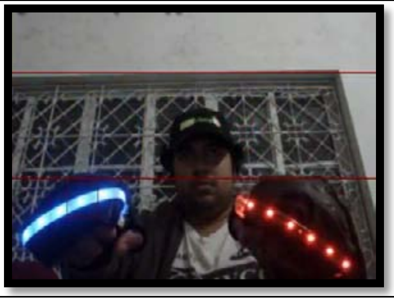
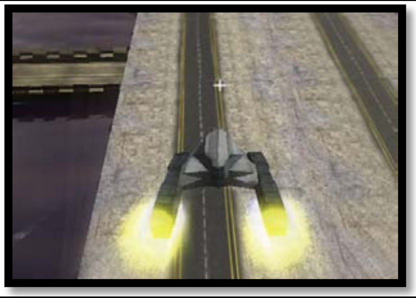
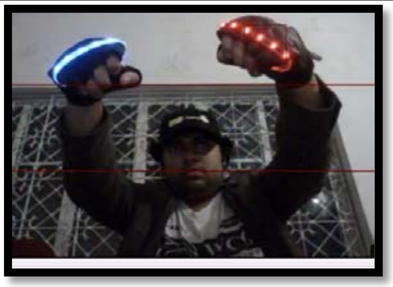
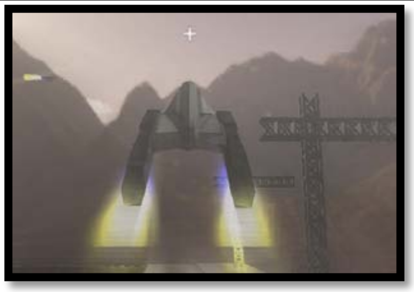
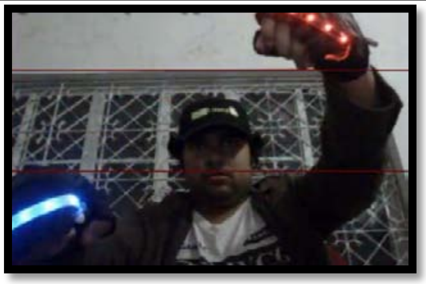
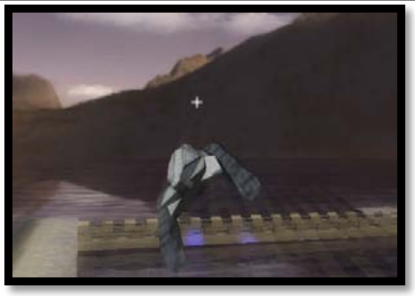
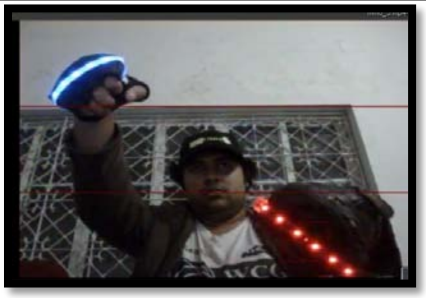
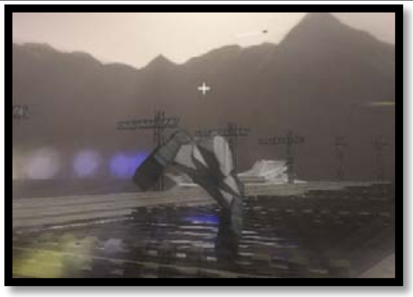
		When both of the hands are at the middle part of the screen as the screen is divided horizontally into 3 equal parts, the plane goes in forward direction.
		If both of the hands are at downward part of the screen, the plane goes downward.
		If both of the hands are at upward part of the screen, the plane goes upward.
		If left hand (gloves with red light) is at upward portion of the screen and blue is at downward portion, the plane goes left.
		If right hand (gloves with blue light) is at upward portion of the screen and red is at downward portion, the plane goes left.

Fig 6.8: Relative distance of both hands as input in game.

6.2.6 Rotation Detection

To rotate a model, distance of the detected object from the camera was calculated. As this project is not based on stereo vision, which means it uses only one camera for detecting object, so distance of detected object is determined from the size of the detected object.

As a square box is drawn around the detected object, the width and height of the detected object is taken in to account. Depending upon the game, a threshold of depth distance that is distance from the camera is set. If the area of the detected object is bigger than threshold value, then specific command is triggered. The figure below contains an example of using depth to rotate an object in game world.

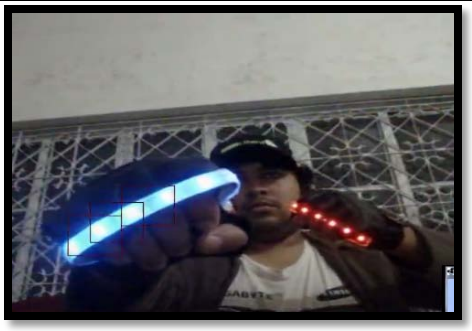
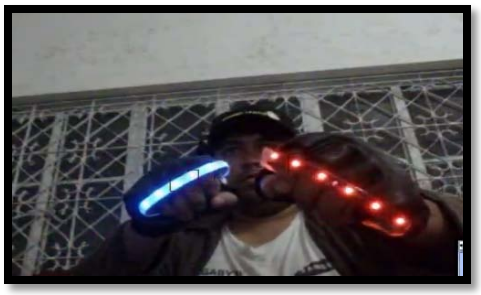
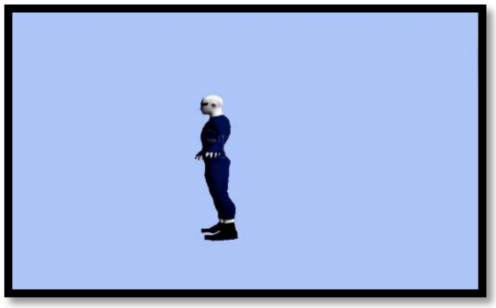
		Blue light is closer to the camera as it has bigger area of square than red light, thus the model is rotating clockwise.
		Red light is closer to the camera as it has bigger area of square than blue light, thus the model is rotating counter clockwise.

Fig 6.9 Rotating object by calculating depth.

6.2.7 Animation Trigger

Specific gesture or movement was used to trigger specific animation. For example, in figure 6.4, the jump animation is triggered when the player jumps. Otherwise the running animation keeps going on.

6.3 Voice as Input

The system was designed to recognize continuous speech for two main purposes which are voice input based user interface and voice as game input. Both of these are described in the section 6.3.1 and 6.3.2 respectively.

6.3.1 Voice input based User Interface

Project's one of the features was to create a natural user interface as part of multi-modal operation with computer. Voice recognition played a significant role in it. CMU Sphinx comes with a rich library of about 12000 English words. For every button on the game screen, a corresponding word is kept in the knowledge base of the Sphinx. One of the game that was developed as part of this project is a flight simulation game. To iterate the planes, the following commands with their language model were used. The accuracy of each word on out of 50 times of test is mentioned in the table.

Table 6.1

Sphinx Language model for UI

Word	Language Model	Accuracy times out of 50
NEXT	N EH K S	43
NEXT(2)	N EH K S T	43
PREVIOUS	P R IY V IY AH S	39
SELECT	S AH L EH K T	46

In the grammar file of the project's demonstration game, 20 different words were kept. To search a word from the grammar, the delay time was close to 0.5 seconds.

6.3.2 Voice as in- game input

This project included speech recognition as game input. In gaming world, speech recognition performs moderately as the delay of 0.5seconds can play an important role. The words are chosen in such a way that it has a good detection rate.

The flight simulation game had following commands.

Table 6.2

Sphinx Language Model

Word	Language Model	Accuracy times out of 50
GO	G OW	47
FIRE	F AY ER	45
FIRE(2)	F AY R	45

6.3.3 Strategic AI Control using Voice

Purpose of this part of the project is to control Non-playable Characters (NPC) using voice. As in most of the games, non playable characters are more or less static in their behavior and Artificial Intelligence. This means they keep doing certain tasks over and over again. As Project Control Scheme Revolution integrates voice input, so game scene was created as part of this project where two game characters were placed on the game screen named “Comrade One” and “Comrade Two” and four base models was created name “East”, “West”, “North”, and “South”.

:

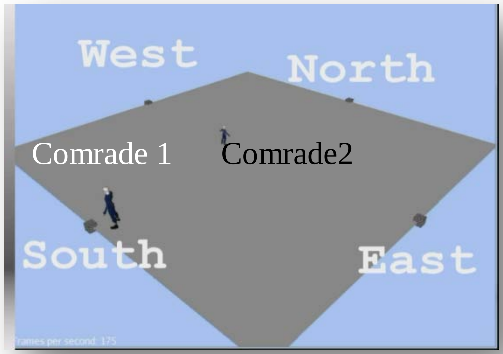


Fig 6.10 Voice command for Non Playable Character control

To send Comrades to specific base, voice command was used. The algorithm is as follows

```
Boolean comrade1= false
Boolean comrade2 = false
Void send_Comrades()
{
    If voice.message=”Comrade1”
        comrade1= true;
        comrade2=false;
    else if voice.message=”Comrade2”
        comrade2=true;
        comrade1=false;
    endif;

    if (comrade1)
        if(voice.message=”south”)
            send comrade1 to south base;
        else if(voice.message=”north”)
            send comrade1 to north base;
        else if(voice.message=”east”)
            send comrade1 to east base;
        else if(voice.message=”west”)
            send comrade1 to west base;
        else
            do nothing
    else if  (comrade2)
        if(voice.message=”south”)
            send comrade2 to south base;
        else if(voice.message=”north”)
            send comrade2 to north base;
        else if(voice.message=”east”)
            send comrade2 to east base;
        else if(voice.message=”west”)
            send comrade2 to west base;
        else
            do nothing
    else
        do nothing
    endif
}
```

This shows that controlling NPC gives a gamer wider and better experience in playing game which is possible by Project C.S.R

Chapter VII

Performance Analysis

7.1 Test Result on a computer

Multi-player Flight simulator game was taken as the subject game for the final test run of this project and was run on PC4 from Table4.1. Frame per second of the game in every ten seconds was taken into account for the first 15 minutes of the game play. The results have been plotted into following graph.

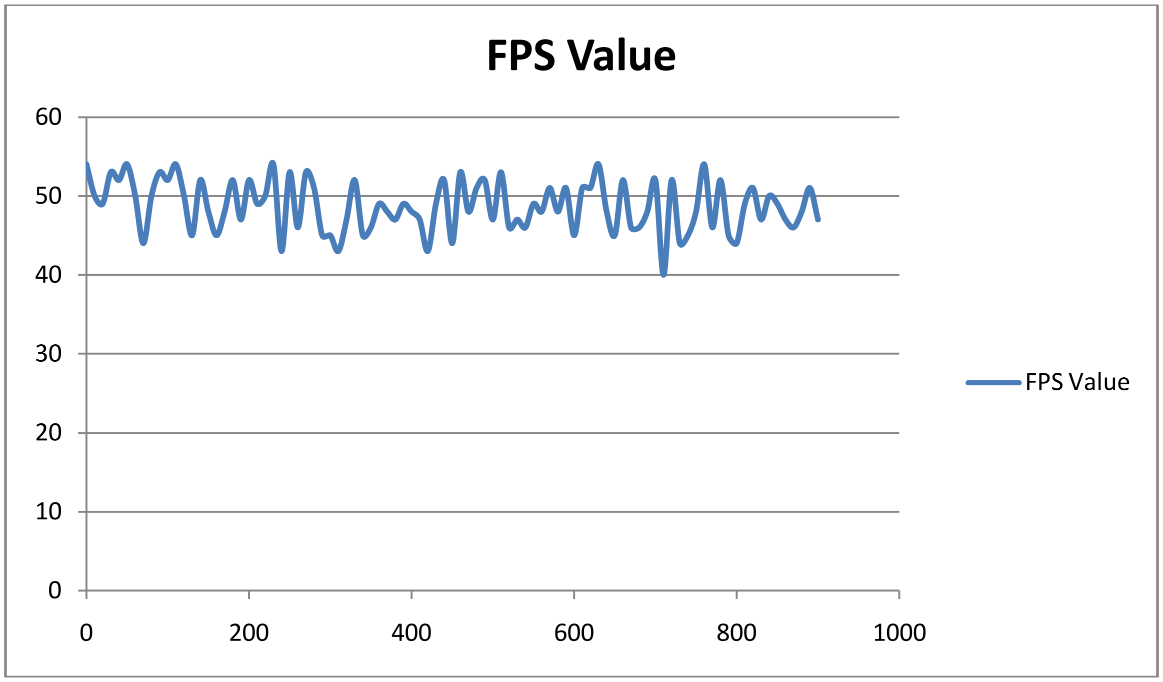


Fig 7.1: Frame per second of the Flight Simulator Game

As we can see from the graph that with all the features included into the game, the average FPS is around 40-45. As we know that for playing a game smoothly, the FPS is required to be around 30-35. Which clearly shows that, using elements of these projects, games which will be developed will have a playable fps.

7.2 Practical Implementation and Gaming Experience

Both of the games developed as part of this project were displayed in three national events where people said these two games are fun to play and they were interesting. As these games did not show any latency while playing, they were well accepted by people.

LIST OF REFERENCES

[1] L.Rainer, M. Jochent(2002) An Extended Set of Haar-like Features for Rapid Object Detection. IEEE ICIP 2002, Vol. 1, pp. 900-903, Sep. 2002.http://www.lienhart.de/ICIP2002.pdf

[2] J.Boye, J. Gustafson, M. Wirem (2006) Robust spoken language understanding in a computer game. Speech Communication(ISSN 0167-6393), Vol. 48, pp. 335-353, 2002.http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-15468

[3] L.Kai-Fu, H.Hsiao-Wuen, R. Raj(1990) An Overview of the Sphinx Speech Recognition System. IEEE Transection Of Acoustic Speech andSignalProcessing,Vol38. http://www.ri.cmu.edu/pub_files/pub2/lee_k_f_1990_1/lee_k_f_1990_1.pdf

[4] A. Amit (2011) How to make your own HaarTrainied .xml files.http://nayakamitarup.blogspot.com/2011/07/how-to-make-your-own-haar-trained-xml.html

[5] V. Paul, J.J. Michael (2001) Rapid Object Detection using a Boosted Cascade of Simple Features. IEEE CVPR. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.7597