IMPLEMENTATION OF VIRTUAL TABLE TENNIS

By

NC Rabia Fatima

NC Aamina Hassan

PC Mustafa Waqar

Project Supervisor: Lt. Col Dr. Muhammad Arif Wahla

Submitted to the Faculty of Electrical Engineering Department,
National University of Sciences and Technology, Islamabad in partial fulfillment for
the requirements of a B.E Degree in Telecommunication Engineering

JUNE 2013

ABSTRACT

IMPLEMENTATION OF VIRTUAL TABLE TENNIS

Implementation of virtual table tennis has been carried out through real time video processing. A realistic three dimensional virtual table tennis has been developed using existing video processing techniques. A detailed explanation of the various modules of the design and their implementations is provided.

The game is developed to replace the conventional controllers of the games such as key board, mouse or joy sticks. The trend is now shifting towards kinect and other sensor based games in which player is physically involved. To remove the conventional monotony of the games we have used a real paddle to play. The user uses a real paddle to play the game.

Our first step was to use a Kinect device that follows movement of hand to play the game. The kinect based games are widely deployed around the world but in Pakistan, it is not that common mainly because of the fact that it is very expensive. To cater for the expense we used an integrated digital camera to map the 3d movements of paddle in the two dimensional world of computer screen through video processing .Thus, making a standalone and cost effective table tennis.

CERTIFICATE OF CORRECTNESS AND APPROVAL

It is certified that the work contained in this thesis titled "Implementation of Virtual table Tennis", carried out by Rabia Fatima, Aamina Hassan and Mustafa Waqar under the supervision of Asst. Prof. Dr. Muhammad Arif Wahla in partial fulfillment of the Bachelors of Telecommunication Engineering, is correct and approved.

Approved By

____

Asst. Prof. Dr. Muhammad Arif Wahla

Project Supervisor

Military College of Signals, NUST

Dedication

Almighty Allah for His blessings,

Teachers and friends for their help,

And Our Parents for their support and prayers

ACKNOWLEDGEMENT

All praises for ALLAH All Mighty who enlightened us with the requisite knowledge on portion of this subject enabling us to accomplish this extremely challenging and gigantic task.

We would like to express our gratitude towards our advisor Dr. Muhammad Arif Wahla for all his help, invaluable guidance, critics and generous support throughout our final year project.

Special acknowledgements to all teachers at MCS, NUST for helping us. Their interest in this project was very beneficial and helped design many vital parts of the project.

We would also like to thank all the lab staff, our friends and all those, whoever has helped us either directly or indirectly, in the completion of our final year project and thesis.

LIST OF ABBREVIATIONS

GUI	Graphical User Interface
MCS	Military College of signals
MIC	Minimum inhibitory concentration
RGB	Red, green and blue
SDK	Software development kit
USB	Universal Serial Bus

Table of Contents

Table of Figures

CHAPTER 1

INTRODUCTION

1.1 Overview of Chapter

This chapter gives the basic information about the idea of developing the project. The chapter covers the project's background, its objectives, and scope. The problem statement along with its solution is also stated in it.

1.2 Project background

With the development in technology in almost every field of life, there is a tremendous revolution in the gaming industry also. The computer games have undergone a huge range of changes in the hardware, interactive devices, software tools, demographics of the players and design of the games. There is a new revolution in the gaming industry after the development of powerful processors and diversified sensors in the technical market. To keep oneself alive in the industry and to cope up with the upcoming challenges and new technology one needs to be geared up.

1.3 Problem Statement

From the word "computer game" we take it for granted that keyboards and mouse will be used as commanding devices. The frequent movement from keyboard to move and back could cause carpal tunnel syndrome. New types of games are introduced in the market every year. One need to remove the monotony of the games and involve the player more interactively to the games .In order to survive in the industry new ventures and innovations should be introduced to keep the technology revolving and keeping oneself to the safe side.

1.4 Solution

We aimed to develop a realistic game in which a real paddle is used to play the game rather than giving commands through mouse, keyboard. For this purpose we first used kinect device and interfaced it to our simulation made in flash, next we mapped the three dimensional movement of physical paddle into the virtual 2D world of the computer screen using an integrated digital camera and reached our goal.

1.5 Project objectives

1.5.1 Academic objectives

Academic objectives included the thorough study of video processing techniques, learning these techniques and applying in real time. Moreover project involves making a game interface and applying gaming logic to the processed content. Kinect for Windows supports movement, voice, and gesture recognition technology for application development. Regarding kinect we studied only gesture recognition technology, we used only hand object from its SDK which allows a user to use his hands to *control* the Windows mouse.

1.5.2 Industrial Objectives

Industrial objectives included provision of a cost effective solution for virtual gamming environment, to bring technology in everyday use in Pakistan and to provide a prototype that can also be used for a number of other virtual games such as combating, flying, virtual tracking etc.

1.6 Project Scope

This project provides a complete solution for replacement of keyboard and mouse as controllers for games. It provides an interactive and user friendly virtual environment for the players. It has increased the overall efficiency of the games.

We initially used kinect SDK for windows to achieve our goal. Kinect for windows has the capability to track the skeleton image of one or two people moving within the Kinect field of view for gesture-driven applications. We programmed it using its one object i.e. hands in our case and gave cursor control to it, finally we interfaced it with the game graphics.

We acquire video from the camera in our second approach, which then is subjected to video processing techniques. Required features i.e. position velocity and tilt of the paddle, are extracted. And this information is passed to the application interface. The same video processing techniques can be used in other games like combating and in other applications, like object tracking, by making changes in the software part of the system.

1.7 Project Limitations

In order to make the system cost effective, the team has used a single camera to input video, so it is quite possible that some of the angles and positions beyond a certain distance are missed out. This problem can be solved by using multiple cameras.

To use the same system for other applications, the software part needs to be redesigned.

Game requires powerful processor and in case of poor processor the delay of the ball impact is increased.

CHAPTER 2

Back Ground and Literature review

2.1 Overview of chapter

The chapter includes the development of computer games over the last half century, the boom in the different aspects of the games and our academic survey done in order to get the theoretical concepts of the techniques used in the project.

2.2 Brief history of computer games

To design the computer games of the present, one needs to understand the games of the past. A brief overview of the most important developments in computer games over the half century is stated below.

2.2.1 Chronology

The first computer game that was ever created was probably the game *OXO* by Alexander Douglas in 1952. It was a version of tic-tac-toe that he wrote for his thesis on human computer interaction. But most people consider the real first interactive computer game to be *Tennis for Two*, developed in 1958 by nuclear physicist William Higginbotham

as shown in the Fig. 2-1 for visitors of the Brookhaven National Laboratory.

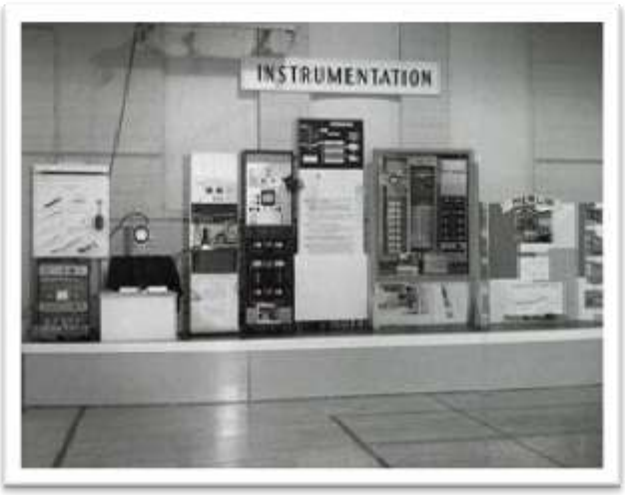


Figure 2-1: Tennis for Two, developed in 1958 by nuclear physicist William Higginbotham

1970 – 1979 is often referred to as the golden age for video arcade games. The first commercial exploitation of computer games came through the arcade machines as shown in the Fig.2-2. These machines were allowed to cost a considerable amount of money, making it commercially feasible to exploit computer games.

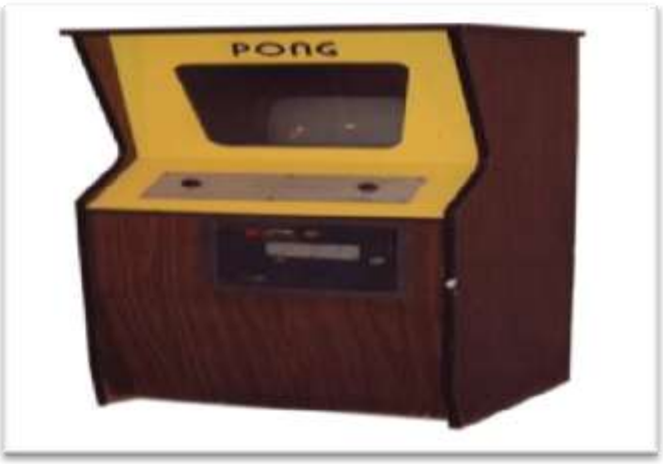


Figure 2-2: First Ping Pong Machine

In 1976 the Channel F system appeared that for the first time used cartridges for the games, making it possible to play different games on the same system, which was a very important innovation. From 1980 to 1989 the number of games increased rapidly. Many famous games were produced in this decade, like *Zork* (1980), *Donkey Kong* (1981), *Pitfall* (1982), *Mario Bros.* (1983), *Dragon's Lair* (1983), *King'Quest* (1984), *Elite* (1984), *The Legend of Zelda* (1986), *Metroid* (1986), *Mega Man* (1987), *Final Fantasy* (1987), *Metal Gear* (1987), *Popoulos* (1989) and *Prince of Persia* (1989).

Many companies appeared creating games for the various consoles, the game console marker crashed in 1984. Many companies in the U.S. went bankrupt or stepped out of the game business. As a result, the focal point of game production shifted to Japan. Reason for the crash was the introduction of new game computers. Cheap personal computers appeared that were particularly suited for games because of their memory, graphics and sound capabilities.

From 1990 to 1999 a huge increase in the power of the game consoles was seen, while at the same moment the popularity of the game arcade machines declined because they were no longer of superior quality. Between 1994 and 1996 the next generation of game consoles appeared. They used faster 32 or 64 bit processors and had much more memory (2-4 Mb). They also had special hardware for 3D graphics. Another advantage of PCs for gaming was that they had a mouse and keyboard. This allowed for games in which the player has to quickly point to locations on the screen, and for games that required many different commands. One of the problems of PC games was that it was initially very difficult to install them, in particular when most people were still using DOS.

And finally, in 2001, Microsoft entered the market with the Xbox. The Xbox was too expensive and too late. The PlayStation 2 had already taken the market and there were many more games available for it. Creating games for the new consoles also became a much more complicated and expensive task. Game budgets rose from one to five million dollars per game. In 2005 Microsoft started the most recent generation of game consoles with the Xbox 360 as shown in the Fig.2-3

Figure 2-3: Xbox 360

Nintendo continued this focus on casual games and physical interaction. With the introduction of *Wii Fit* that came with a balance board that measured your weight distribution. It was again a big success with close to 25 million copies being sold. Sony and Microsoft are reluctant because of the huge development and launch costs involved. Also, consoles are normally sold below the production price leading to an initial loss that will take a long time to recover.

Over the past two decades the gaming world has diversified tremendously. We play games on devices ranging from simple mobile phones to dedicated game consoles. In 2010 in the US alone people spend about 25 billion dollar on games.

2.3 Technology Change in the gaming industry

The changes in hardware, development software, budgets, and experience have changed the games considerably. Here are listed some of the areas where the gaming industry has boomed a lot.

2.3.1 Development in hardware of games

Developments in hardware had a huge effect on what was possible in computer games and hence, enabled the game designers to make different and more fascinating games. It went from devices with minimal computing power, memory, and graphics to the consoles of today with special 3D graphics cards, DVD disks to store game data, and Internet connectivity.

2.3.2 Development in interaction devices

Early game consoles had special rotating knobs or simple joysticks and a few buttons. Nowadays game controllers have multiple joysticks, and lots of buttons. And in recent years this has been extended with devices that measure movement, like the Wii controllers of Nintendo or the Kinect system of Microsoft as shown in the Fig.2-4. Obviously these had a strong impact on the game play.

Figure 2-4: Play Station, Game cube and The Nintendo Wii and the game Wii Fit. Development in the software tools

Initially game developers wrote every line of code them (often in assembly language) and drew every pixel of the artwork. Nowadays extensive game engines and other middleware packages are available allowing for much more sophisticated games. And artist, animators, and level designers use advanced tools that help them to create complicated artwork efficiently.

2.3.3 Boom in the game business

While initially games were developed primarily by individuals, now-a-days there are huge teams of specialists working on a single game. Development budgets have grown from a few thousand dollars to tens of millions of dollars per game as shown in the Fig.2-5. Also educational programs have appeared to train the professionals needed by the game industry.

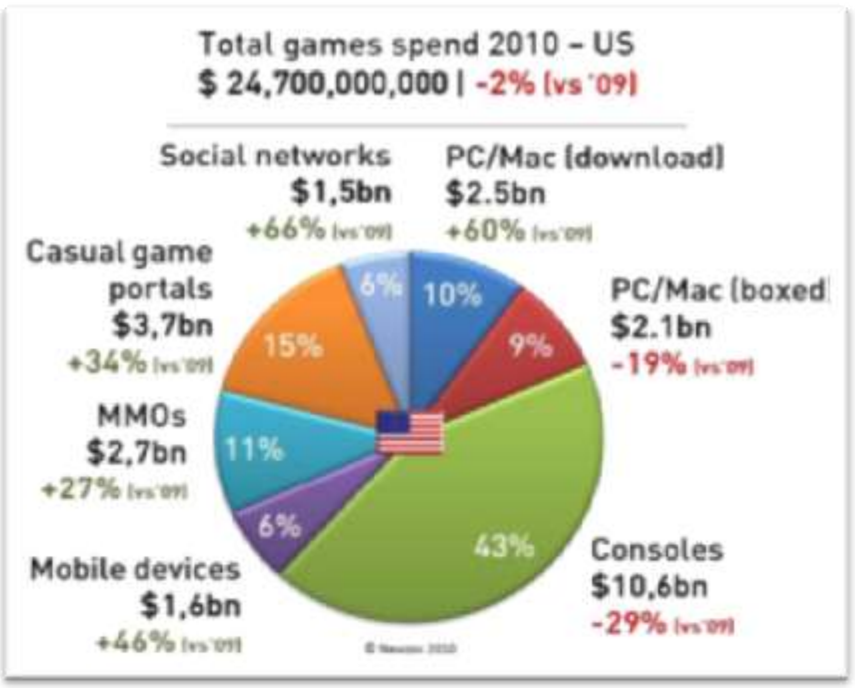


Figure 2-5: Game expenditure in the US in 2010 for the different types of games

2.3.4 Change in the demographics of the players

While initially games were primarily played by young males, this has considerably changed in recent years. The number of female players is close to the number of male players, and the age of players ranges from 4 to 100. This has led to new genres of games.

All the diversified features have made the gaming environment a really fascinating, tough and challenging world.

2.4 Academic survey

Our approach is to first develop our game with kinect and then to use an integrated digital camera to make it cost effective so we learnt kinect programming and video processing techniques for Matlab as well.

2.4.1 Programming for Kinect SDK

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. We used the hands object of kinect out of its five objects from its SDK and utilize it in our application. Kinect Mouse Cursor is an application that uses the Kinect for Windows SDK and its skeletal tracking features to allow a user to use their hands to control the Windows mouse cursor. We learnt initialization of Kinect sensor for use. This consists of two parts, first to attach Kinect sensor, then to initialize it and prepare to read data from it.

2.4.2 Video Acquisition and Video Processing techniques in Matlab

Here our main focus of study was to get theoretical concepts of Image and video Processing in Matlab. We learnt to interface camera with Matlab and to acquire video from it. As we are using red paddle to play the game so our aim was to remove the other colour pixels from the image so as to detect and proceed only with red colour. We ought to remove the noise as well, for this we studied and implemented a median filter to filter out noise. Median Filter is a type of filter for smoothening and reducing noise from an image while preserving strong edges as shown in Fig.2-6. Filter replaces the value of a pixel by the median of the values of surrounding pixels up to a defined range.

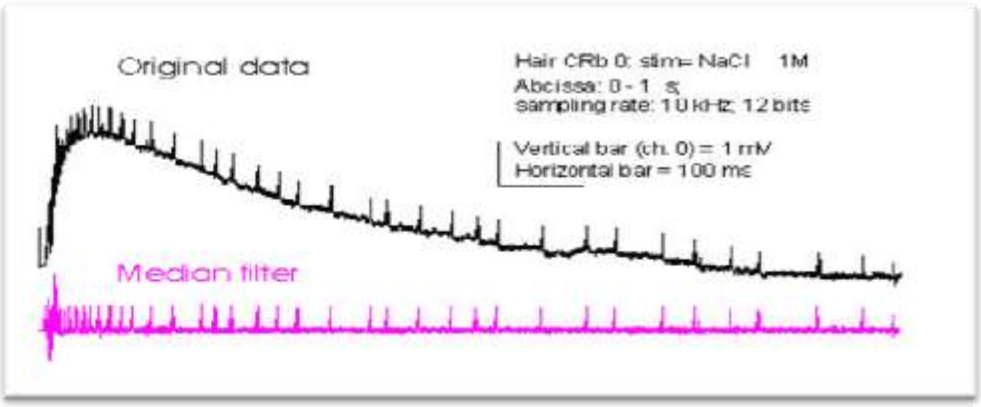
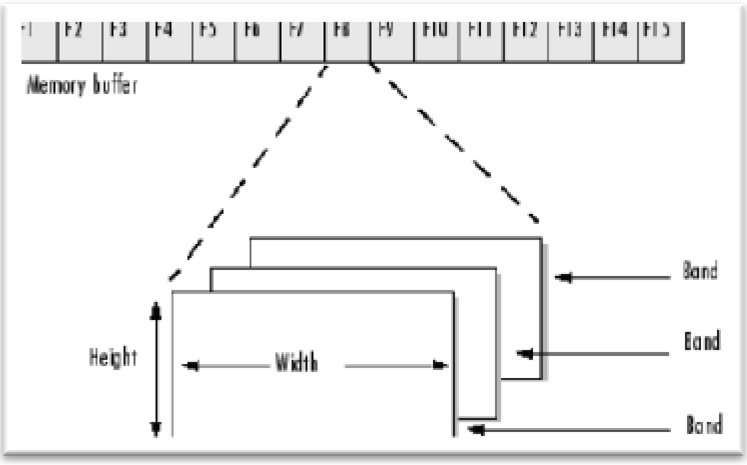


Figure 2-6: Median filter

Our required Video processing techniques further includes working on the individual frames for edge detection, measuring the velocity and other parameters. In reality, each frame is a multidimensional array. The following Fig.2-7 illustrates the format of an individual frame.



1

Figure 2-7: Format of an Individual Frame

Later on we run the snapshots at a frame rate of 25 and derived our own formulas for velocity and angle calculation.

2.5 Approach to our project

After our academic survey we were able to find out the route to our end goal. We found that our project will be completed in three development phases as shown in the Fig.2-8. First development phase is the development of game using Kinect device to have the idea of virtual control, second phase includes the development of algorithms for paddle parameter extraction on images and the third phase include the development of Matlab game using algorithms developed for phase two. The second and third phases are specifically done to develop the cost effective solution of our project.

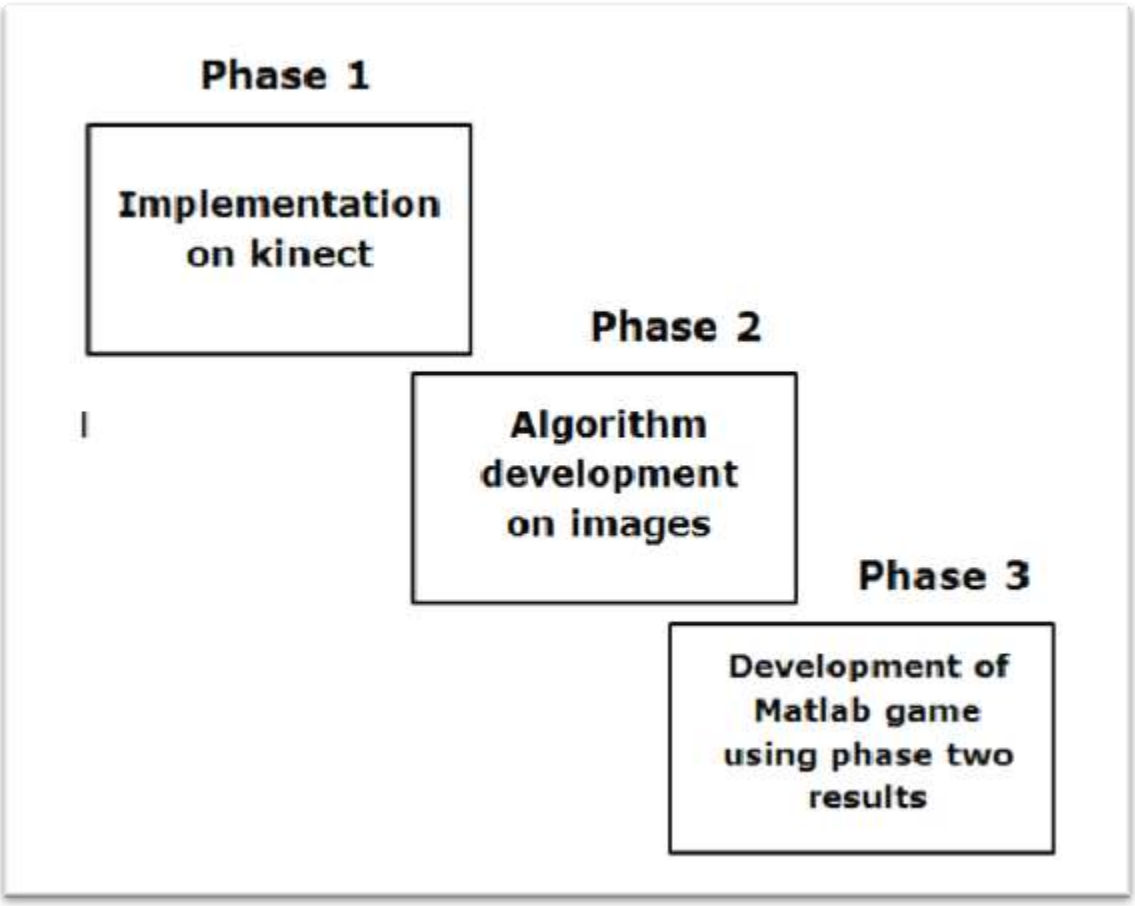

Phase 1
Implementation on kinect
Phase 2
Algorithm development on images
Phase 3
Development of Matlab game using phase two results

Figure 2-8: Our Project Approach

CHAPTER 3

Kinect Implementation

3.1 Overview of chapter

This chapter explains the first phase of our project and that is kinect implementation of virtual table tennis its hardware design, initialization, configuration, programing and interfacing with the game environment.

3.2 Introduction to Kinect for Windows SDK

This section consists of the hardware design of kinect for windows SDK and its features with detail. In kinect we can visually see is the 3D depth sensors, RGB cameras, multi array MIC, and motorized tilt as shown in the Fig.3-1

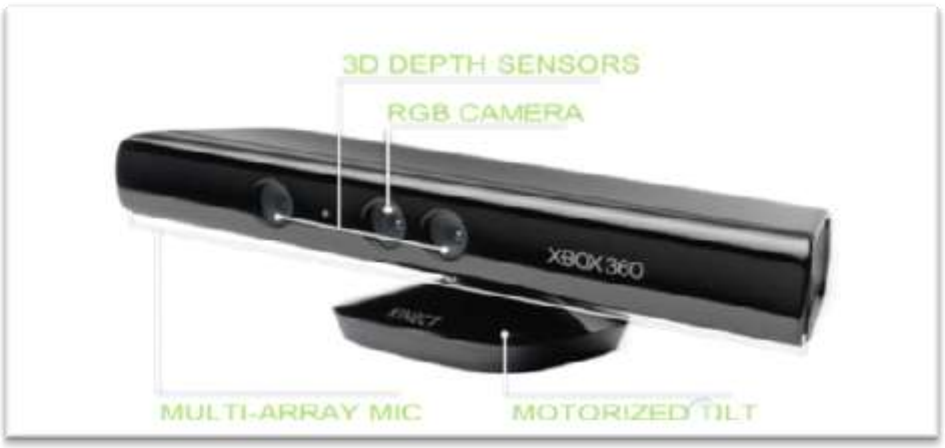


Figure 3-1: Kinect for Xbox

3.2.1 Kinect SDK architecture

Kinect SDK architecture includes the Kinect hardware, Kinect sensor and the USB hub through which the Kinect sensor is connected to the computer. The Windows drivers for the Kinect, which are installed as part of the SDK setup process. The fig.3-2 shows the architecture

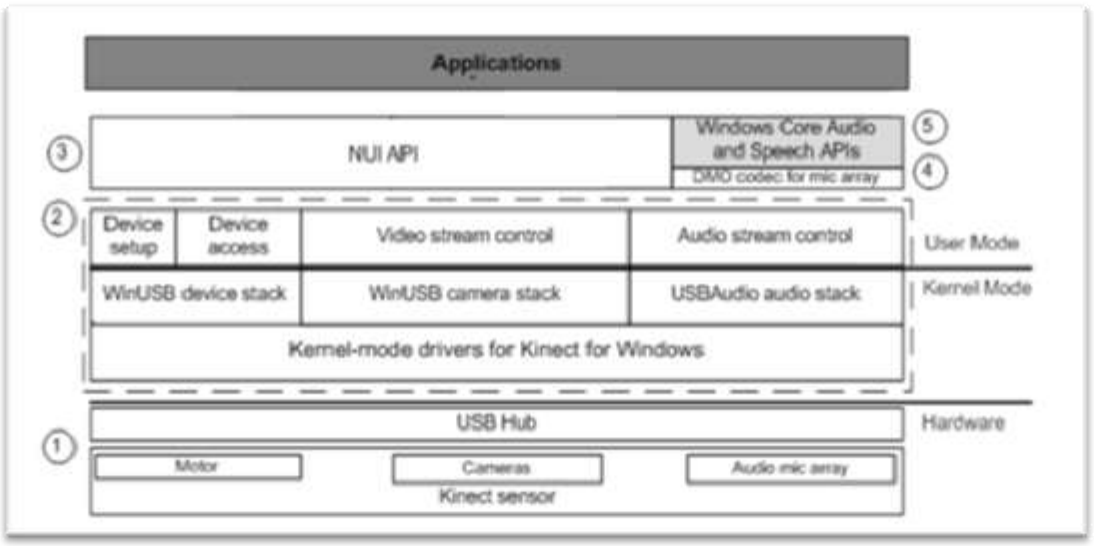


Figure 3-2: Kinect SDK architecture

The Kinect drivers support the Kinect microphone array as a kernel-mode audio device that you can access through the standard audio, APIs in Windows, Audio and video streaming controls for streaming audio and video (color, depth, and skeleton).

Audio and Video Components are also supported by kinect SDK. Kinect SDK provides natural user interface for skeleton tracking, audio, and color and depth imaging. DirectX Media Object (DMO) for microphone array beam forming and audio source localization.

It also provides Windows 7 standard APIs - The audio, speech, and media APIs in Windows 7. These APIs are also available to desktop applications in Windows 8.

3.3 Building blocks

There are two types of data streams in it, Data streams and Recognition Stream**.**

3.3.1 Data streams

They are basically the streams that the sensor can capture with its sensors like Color with the camera, Depth with the IR Emitter & receiver as shown in Fig.3-3

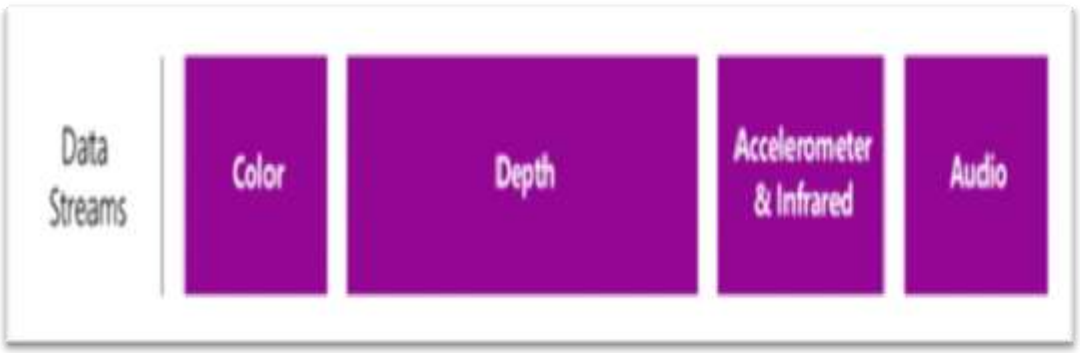


Figure 3-3: Data Stream

3.3.2 Recognition streams

The Recognition stream Diagram is shown in Fig.3-4

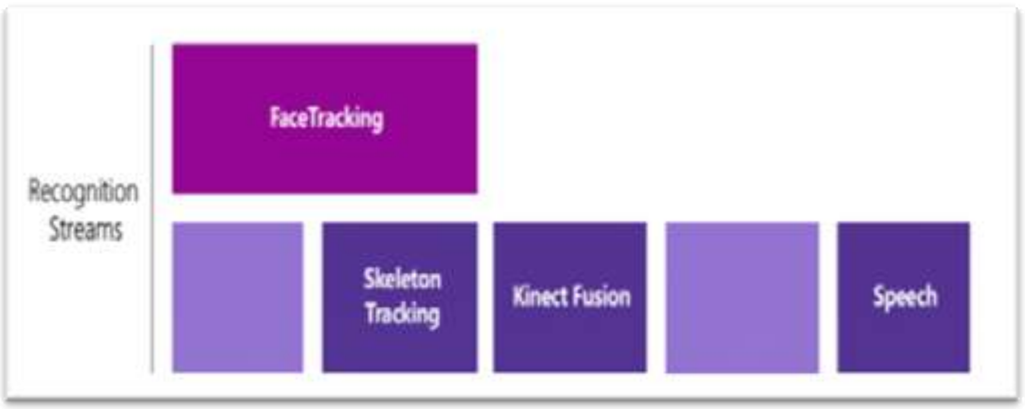


Figure 3-4: Recognition streams

They are those who rely on the ones below them and process the enable Color, Depth and Skeleton Tracking in order to enable Face Tracking.

3.4 Complete design

The complete design of our application is given in Fig.3-4

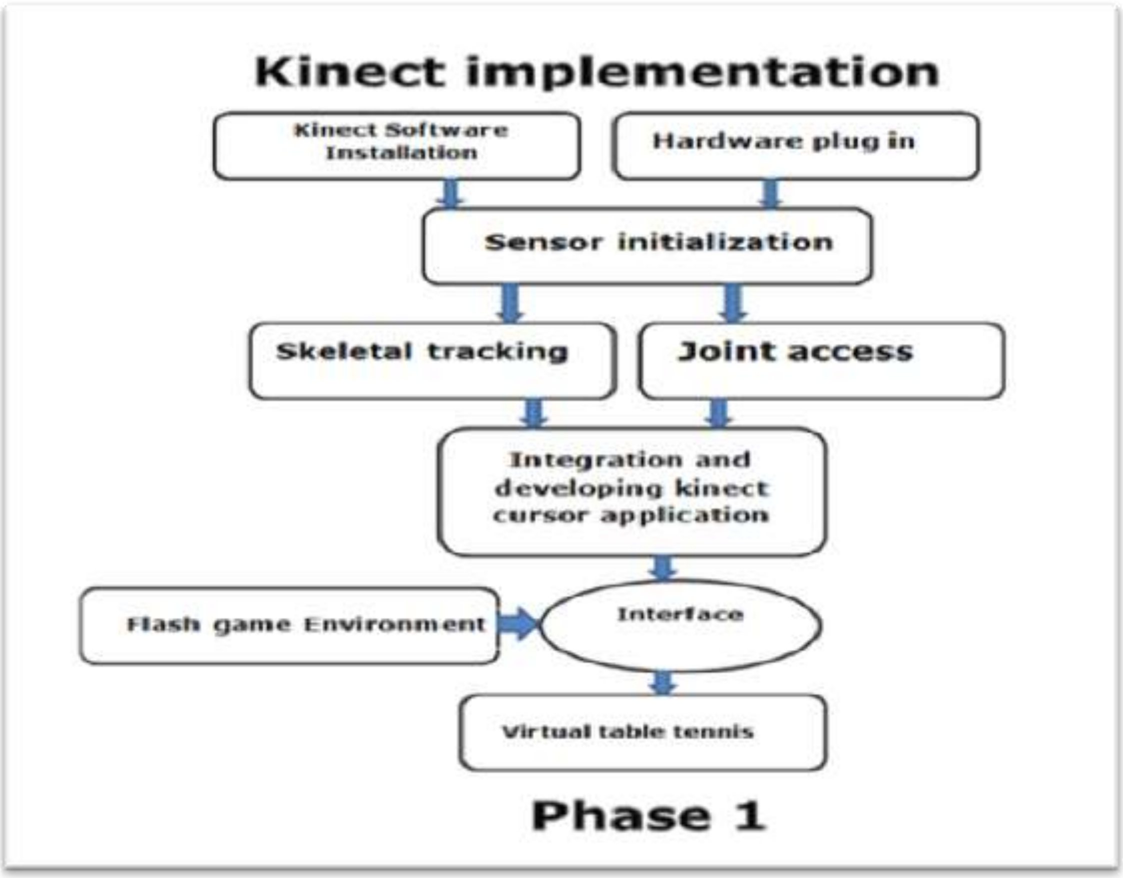


Figure 3-4: Complete Design

3.5 Kinect Initialization

The Microsoft official SDK for Kinect is easy to use. Basically you download it, plug the Kinect into a free USB socket and start programming.

3.5.1 Software Installation

It requires at least 190 MB of available storage space, the Kinect system software allows users to operate the Xbox 360 Dashboard console user interface through voice commands and hand gestures. Techniques such as voice recognition and facial recognition are employed to automatically identify users. Games that require Kinect have a purple sticker on them with a white silhouette of the Kinect sensor and "Requires Kinect Sensor" underneath in white text, and also come in purple packaging. Games that have optional Kinect support (meaning that Kinect is not necessary to play the game or that there are optional Kinect minigames included) feature a standard green Xbox 360 case with a purple bar underneath the header, a silhouette of the Kinect sensor and "Better with Kinect Sensor" next to it in white text.

3.5.2 Hardware Connection

Looking at the plug on the end of the cable that comes out of the Kinect one realize that what it looks like a USB connector but not quite. The connector used is a proprietary connector designed to allow the Kinect to be plugged into the new black XBOX 360 without the use of an external power supply. Older Xboxes don't have the special connector and they come with an external power supply which has an adaptor to convert the Kinect plug into a standard USB connector and power supply connector.

3.6 Kinect Sensor Organization

Kinect Sensor is an object that represents a Kinect sensor as shown in Fig.3-5 We can enable colour/depth/skeletal tracking for a sensor, change the angle with Elevation Angle, check what its status is and if it is running, etc. From Fig.3-5 one can see that each sensor has three streams, Color / Depth / Skeleton, each represented by an object Color Image Stream / Depth Image Stream / Skeleton Stream that gives Frames with the data in it when we call the Open Next Frame-method. Color and depth are quite the same since a frame has a Width, Height, Bytes per Pixel & Pixel Data Length while a Skeleton Frame has a Tracking Mode and Skeleton Array Length.

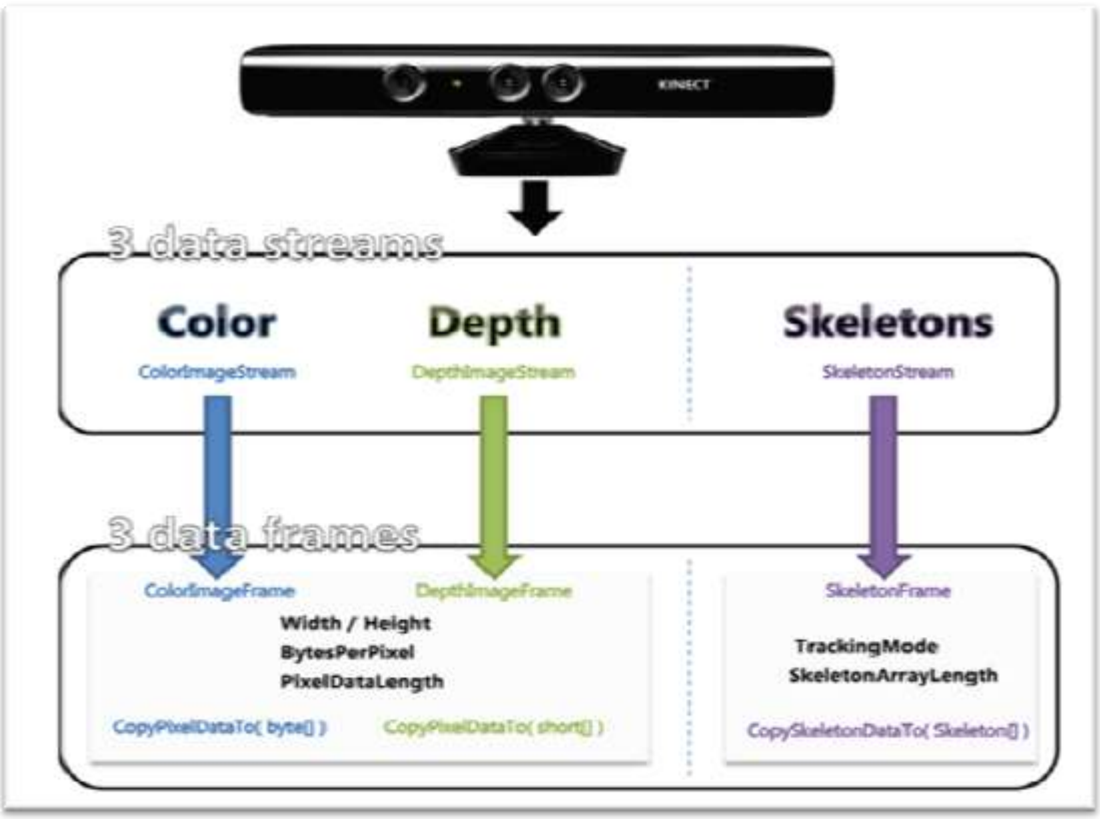
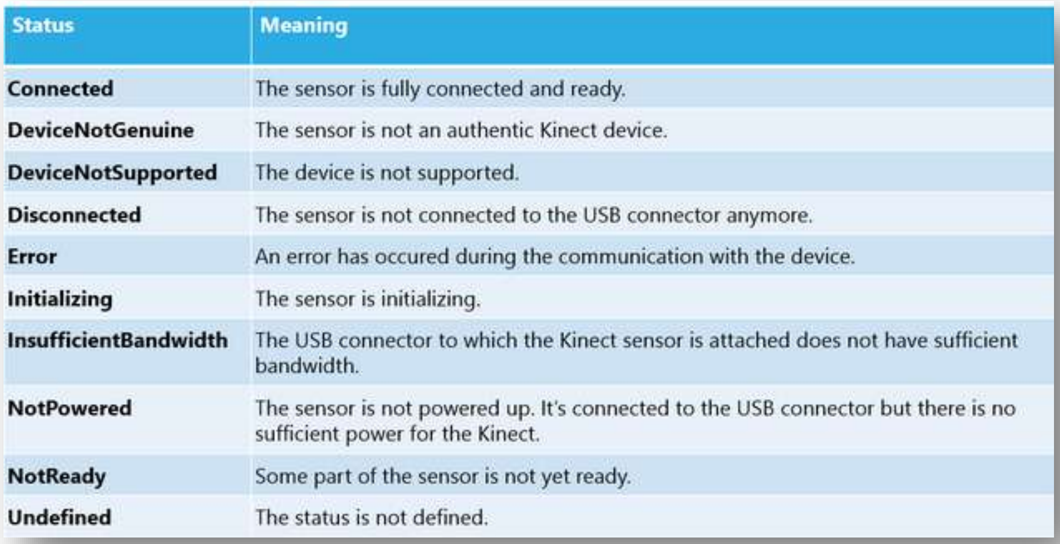


Figure 3-5: Recognition Stream

3.6.1 Status of a sensor



Status	Meaning
Connected	The sensor is fully connected and ready.
DeviceNotGenuine	The sensor is not an authentic Kinect device.
DeviceNotSupported	The device is not supported.
Disconnected	The sensor is not connected to the USB connector anymore.
Error	An error has occured during the communication with the device.
Initializing	The sensor is initializing.
InsufficientBandwidth	The USB connector to which the Kinect sensor is attached does not have sufficient bandwidth.
NotPowered	The sensor is not powered up. It's connected to the USB connector but there is no sufficient power for the Kinect.
NotReady	Some part of the sensor is not yet ready.
Undefined	The status is not defined.

Figure 3-6: Different statuses of a sensor

Every Kinect Sensor-object has a status, called Kinect Status, indicating in what state your sensor is. Some of the examples are shown in Fig.3-6

3.7 Skeletal Tracking

Our application involves the skeletal tracking and then selecting the objects of our use.

3.7.1 Overview

Skeletal Tracking allows Kinect to recognize people and follow their actions. Using the infrared (IR) camera, Kinect can recognize up to six users in the field of view of the sensor. Of these, up to two users can be tracked in detail. An application can locate the joints of the tracked users in space and track their movements over time as shown in the Fig.3-7

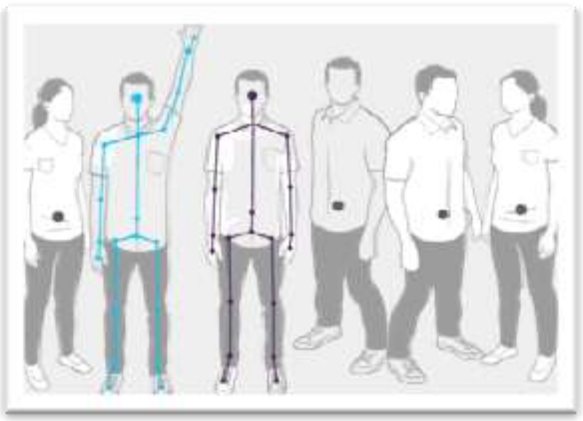


Figure 3-7: Kinect can recognize six people

Skeletal Tracking is optimized to recognize users standing or sitting, and facing the Kinect; sideways poses provide some challenges regarding the part of the user that is not visible to the sensor.

To be recognized, users simply need to be in front of the sensor, making sure the sensor can see their head and upper body, no specific pose or calibration action needs to be taken for a user to be tracked.

3.7.2 Skeleton Data

We have only used the skeleton class in our application. It contains skeleton tracking data, which is joint data that is streamed For example if you want to interact with it and your device is still initializing will result in errors and failure, we don't want that. As well as UX where we can let the user know whether the sensor is connected and ready or not out of a Kinect in the skeleton stream. The skeleton data which a kinect can cover is shown in the Fig.3-8

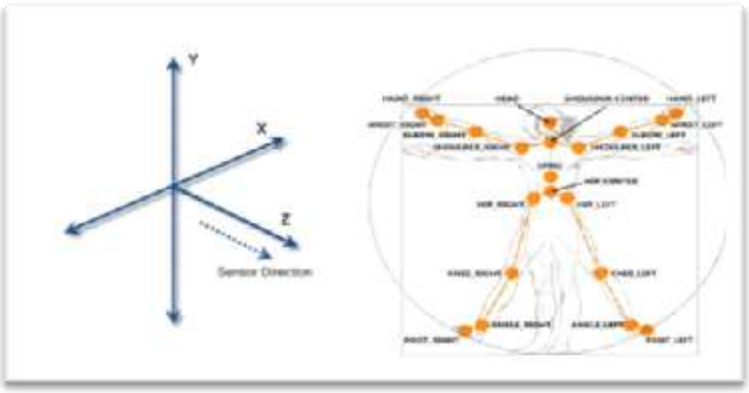


Figure 3-8: Skeleton data Field of View

Kinect field of view of the users is determined by the settings of the IR camera, which are set with the Depth Range Enumeration. In default range mode, Kinect can see people standing between 0.8 meters (2.6 feet) and 4.0 meters (13.1 feet) away; users will have to be able to use their arms at that distance, suggesting a practical range of 1.2 to 3.5 meters. For more details, see the k4w_hig_main.

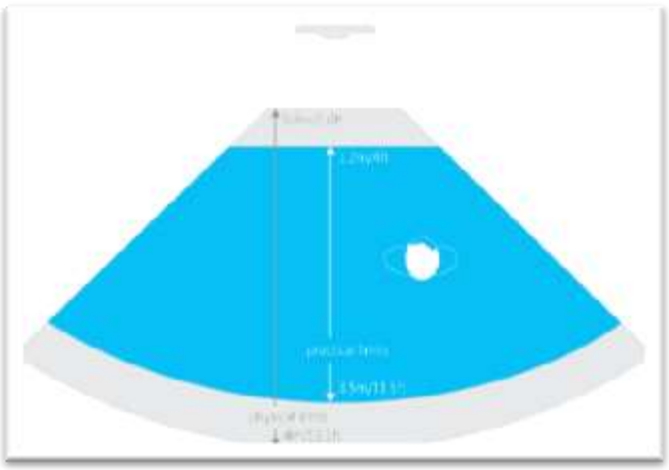


Figure 3-9: Skeleton Data

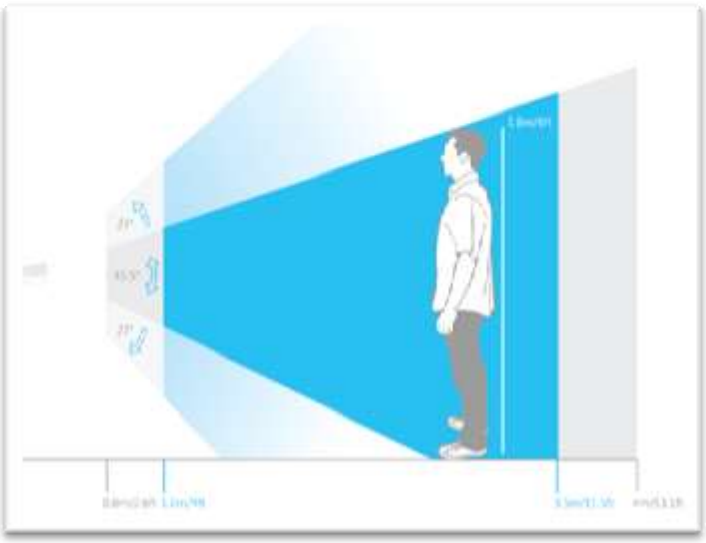


Figure 3-10: Kinect vertical Field of View in default Range

In near range mode, Kinect can see people standing between 0.4 meters (1.3 feet) and 3.0 meters (9.8 feet); it has a practical range of 0.8 to 2.5 meters. For more details, see Tracking Skeletons in Near Depth Range.

3.7.3 Set up skeletal tracking

Setting up skeletal tracking involves Go to the properties window (F4), select the Main Window, select the Events tab, and double click on the Loaded event to create the Window Loaded event as shown in the Fig. 3-11

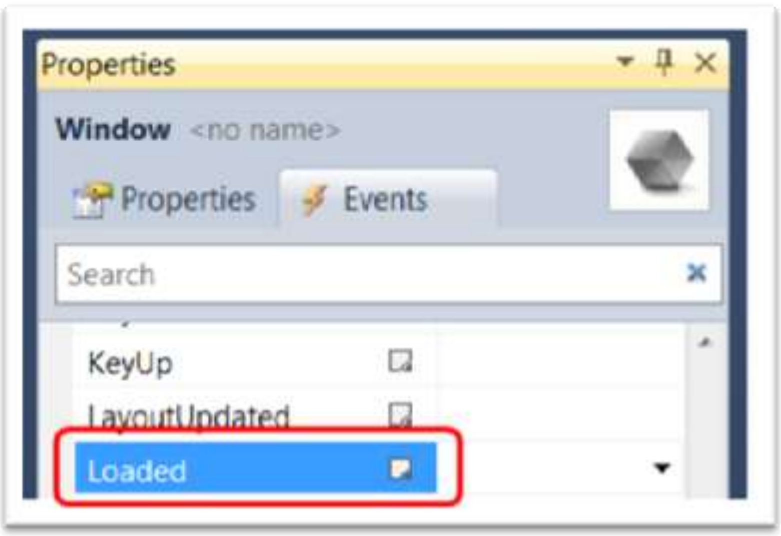


Figure 3-11: Window loading

3.7.4 Initializing the runtime

We created a new variable outside of the Window_Loaded event to reference the Kinect runtime and run the application with the function calling. For more Details see [Annex A]. When the breakpoint fires, one can inspect the Skeleton Frame Ready Event Args to see that the Skeleton Frame returns a collection of six skeletons.

3.7.5 Skeletal tracking

To enable skeletal tracking, we called the Skeleton Stream. Enable method. We access the Kinect Sensor and Skeleton Stream property from the Kinect Sensor class. To receive information about the recognized users, we subscribed to the KinectSensor. SkeletonFrameReady event are managed by the KinectSensor class. We write the C++ code for it. [Annex C]

3.8 Getting a Joint position

A Joint position returns X, Y, Z values as explained below

- X = Horizontal position measured as the distance, in meters from the Kinect along the X Axis
- Y = Vertical position measured as the distance, in meters from the Kinect along the Y Axis
- Z = Distance from Kinect measured in meters

3.9 Joint Positions Access and Drawing a skeleton

For tracked skeletons, an array of joints provides the position of the twenty recognized human joints in space over time. For example, applications can use the hand joint to guide a cursor on the screen or simply draw the position of user's body on the screen. The Joint class has a Tracking State that can access for each joint in the skeleton and used to verify the value of the Joint Tracking State . [Annex D]

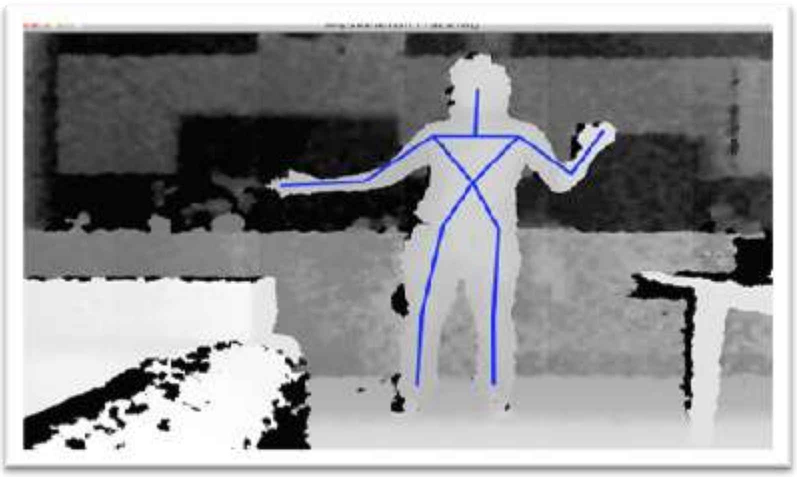


Figure 3-12: Joint Access

3.10 Integration

Finally, we put all the objects in the nui_Skeleton FrameReady event, call the SetEllipsePosition methods for the three joints. [Annex B] The results are shown in the Fig.3-3

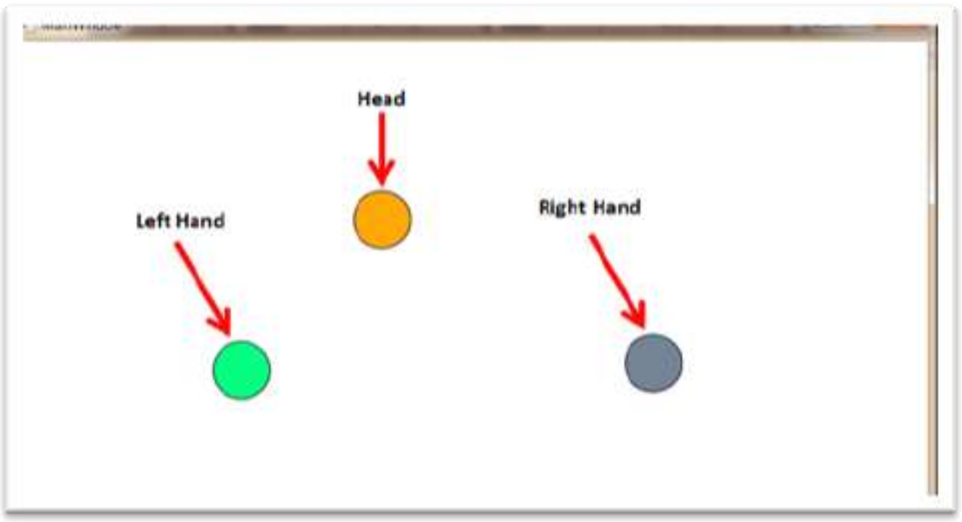


Figure 3-3: Integration of objects

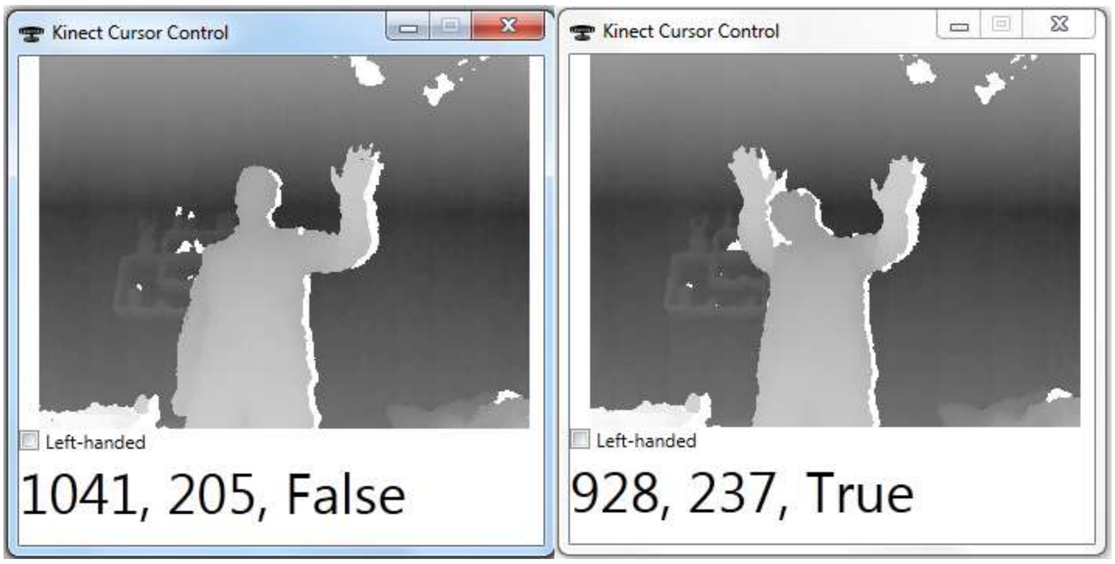
3.11 Cursor Control using kinect

Kinect Mouse Cursor is a demo application that uses the Kinect for Windows SDK and its skeletal tracking features to allow a user to use their hands to control the Windows mouse cursor.

This sample demonstrates how to use the Kinect skeletal tracking feature to move the mouse cursor with a user's hands.

- By default, the right hand moves the cursor and the left hand controls the left mouse button.
- Then the left hand is raised, the mouse button is held down. When the left hand is lowered, the mouse button is released.

- Use the checkbox on the main UI to switch to "left-handed" mode where the hand functions are reversed.



1041, 205, False

928, 237, True

Figure 3-14: Cursor Application showing true and false results

3.12 Interfacing with the graphics

After the application development of cursor control through kinect, our next task was to interface it with the game environment. First we tested it by assigning our cursor control to mouse in our laptop. After that we develop the game adobe flash and assign our cursor control to it. We successfully interface it with our game.

CHAPTER 4

Implementation Using Digital Camera

4.1 Overview of Chapter

The Chapter covers the second and third phase of the project. It explains the complete design that was followed to make a working model of project.

4.2 Approach for cost Effective solution

The camera based game was completed in two parts as shown in the Fig.4-1

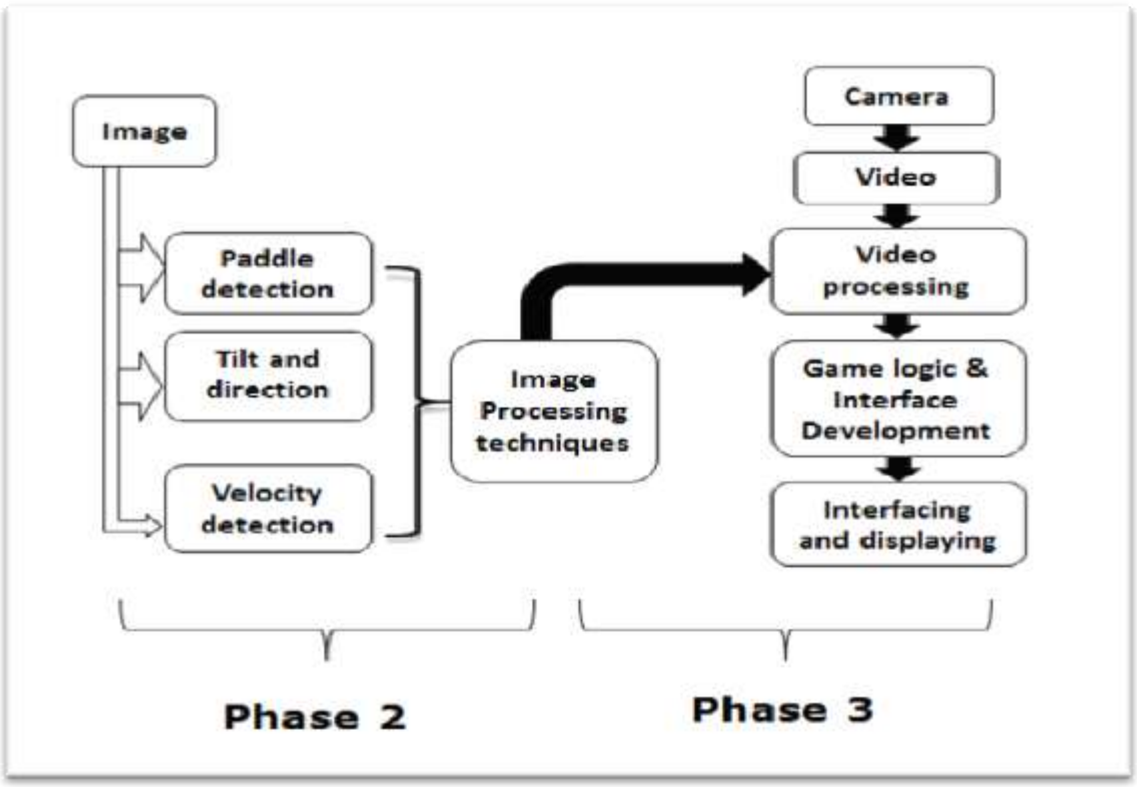


Figure 4-1: Phase 2 and 3 Design Overview

1. Algorithms development for parameter calculation on images(Phase two)
2. Implementing these algorithms for video(phase three)

4.3 Phase 2

The complete flow chart for the images is shown in the Fig. 4-2

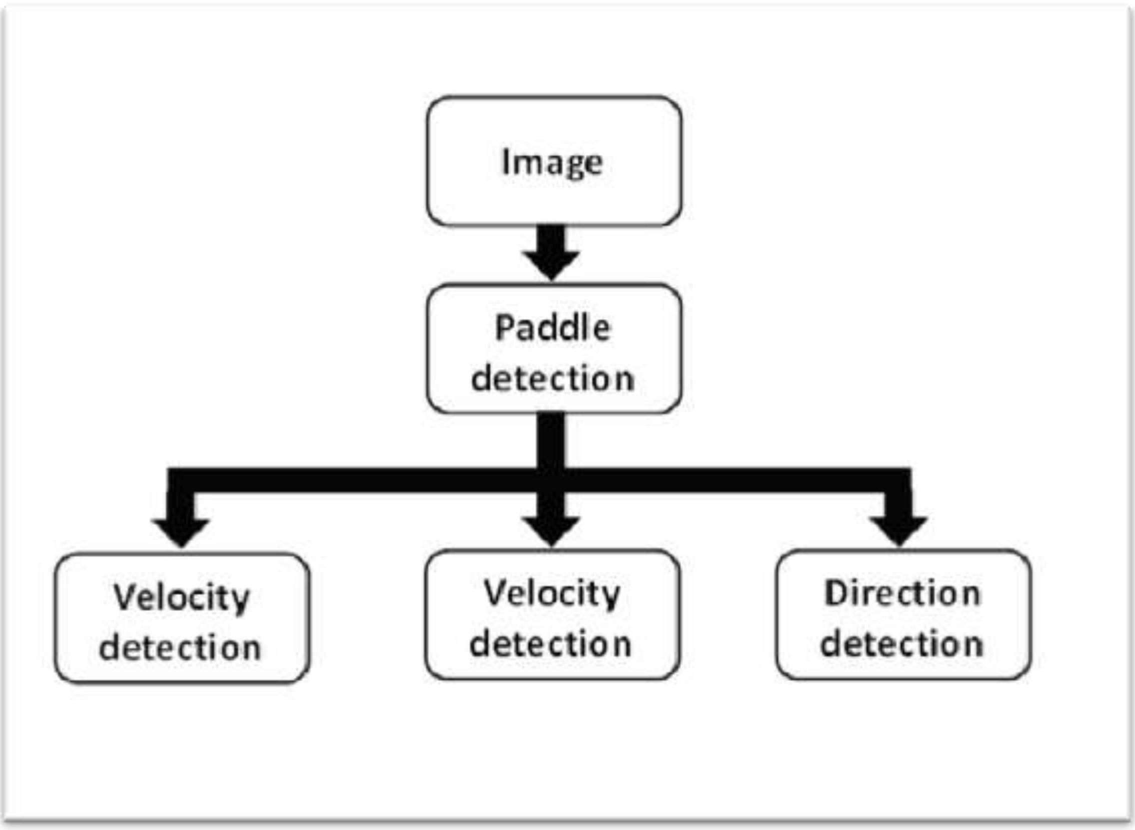


Figure 4-1: phase 1 complete design

The second phase was to develop algorithms to calculate these parameters for the images as the algorithm used for images can easily be implemented on video. The images were captured with the same video camera that was to be used for video acquisition for the game. Paddle detection and position calculation require single image where as two images were needed to calculate the velocity and position of the paddle

4.3.1 Paddle detection

Paddle was detected the basis of color detection. A red colored paddle is used to play the game. It was required to subtract the red component from the grayscale image to extract the red components in the image. We used a median filter to filter out noise. Median Filter is a type of filter for smoothening and reducing noise from an image while preserving strong edges. It is a simple filter that uses the median to obtain its result. Filter replaces the value of a pixel by the median of the values of surrounding pixels up to a defined range. The Results are shown in the Fig. 4-2

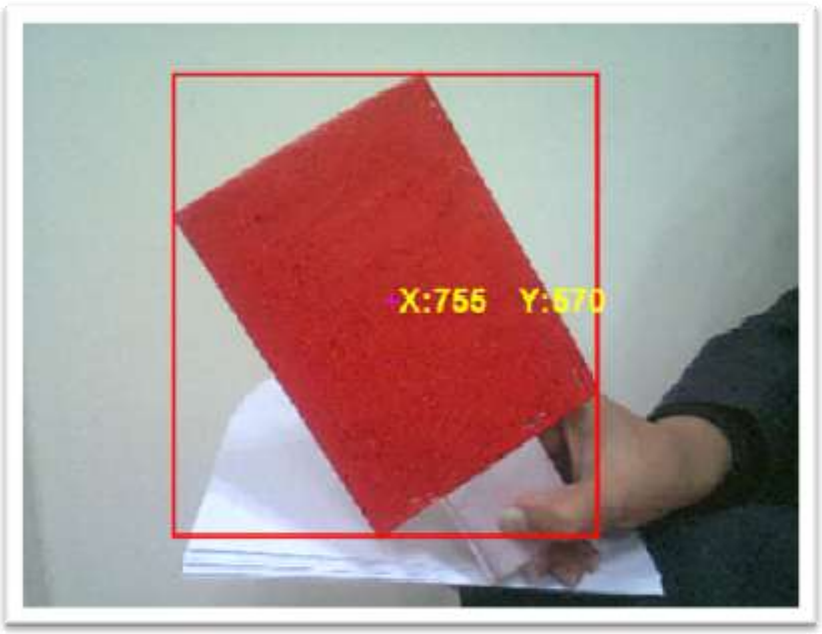


Figure 4-2: Red Paddle Detection

Algorithm of color detection involves conversion of image into gray scale, after those pixels less than 300px are removed. For more details see [Annex E]

4.3.2 Position detection

After the detection of paddle, position of paddle is calculated in an image of 320x240 pixels. The position of paddle is required to be found for the calculation of direction as the position can simply be used to tell whether the paddle has moved right or left.

Position of paddle includes centroid. The centroid is simply found by using inbuilt Matlab function. The results are shown in Fig. 4-3

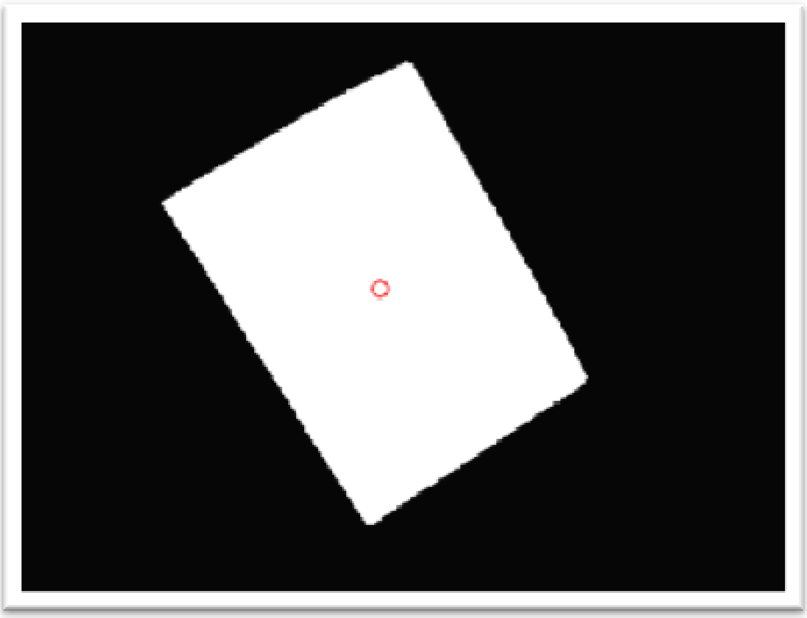


Figure 4-3: centroid of paddle

4.3.3 Velocity calculation

The paddle has been detected and central position of the paddle has been found. The velocity can be calculated with the help of the fact that paddle zooms in and out when it is moved closer to or away from the camera. So the difference in area of the paddle with respect to time can be used to calculate the velocity.

Two images with a paddle are captured. The distance between the paddles in two images is already known and the time for the paddle to cover that already known distance is calculated as shown in Fig.5-4. Using the difference between the areas of two paddles, the velocity can be calculated. But before calculation of velocity, the area of paddle is needed to be known.

The simplest method to calculate the area is to use the lengths of all the four sides of paddle. This can be done by detection of four corners of paddle. Inbuilt command for corner detection was used first. But the result included not only four required corners but also all the edge points of the paddle along with the corners in the image background. So to get four extreme corners of the paddle, the first step that was taken was to remove all the pixels of the image except paddle pixels.

Background algorithm can be applied to remove the static background, but keeping in mind the delay sensitive nature of the game conventional background removal algorithms were not used. Instead a simple approach was used. Red green and blue components of the image were separated. The red part of the image was retained as it is, the blue and green parts of the image were multiplied with zero and the image was recreated. This solved the problem of detection of corners in the background although using this method has a flaw. The background is not actually removed. If there is any read color in the background other than that of paddle that will be retained in the image and corners will be calculated. This method was used to reduce delay.

With the removal of background one problem was still there; it was that all the edges were detected as corners. To filters the required corners out of all the random edge points of the paddle, first step that was taken was to store all the corner values that include x coordinate and y coordinate of the corner point. The second step was to sort these corners. The corner with a minimum value of x was stored, same was done with the corners having maximum value of x, minimum value of y and maximum value of y. So the four pairs of corners valued were retained that were maximum of x and the value of y corresponding to maximum of x, maximum of y and the value of x corresponding to maximum of y, minimum of x and the value of y corresponding to minimum of x, minimum of y and the value of x corresponding to minimum of y. rest of the corners were discarded. The third step was to use the retained corner values to find the required corners of the paddle. Paddle orientation was used for that purpose. The paddle can have three basic orientations.

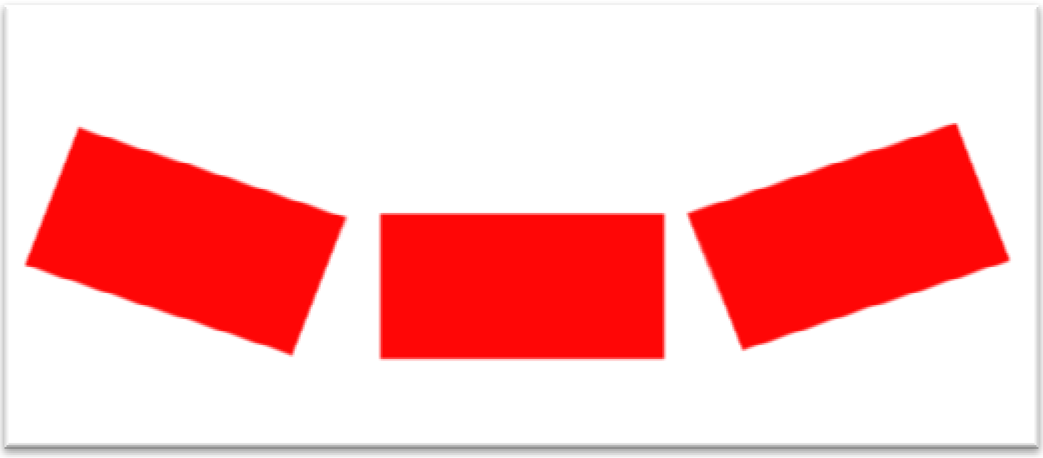


Figure 5-4: Calculating Velocity

Different algorithms were selected to find the corners of the three basic orientations of paddle.

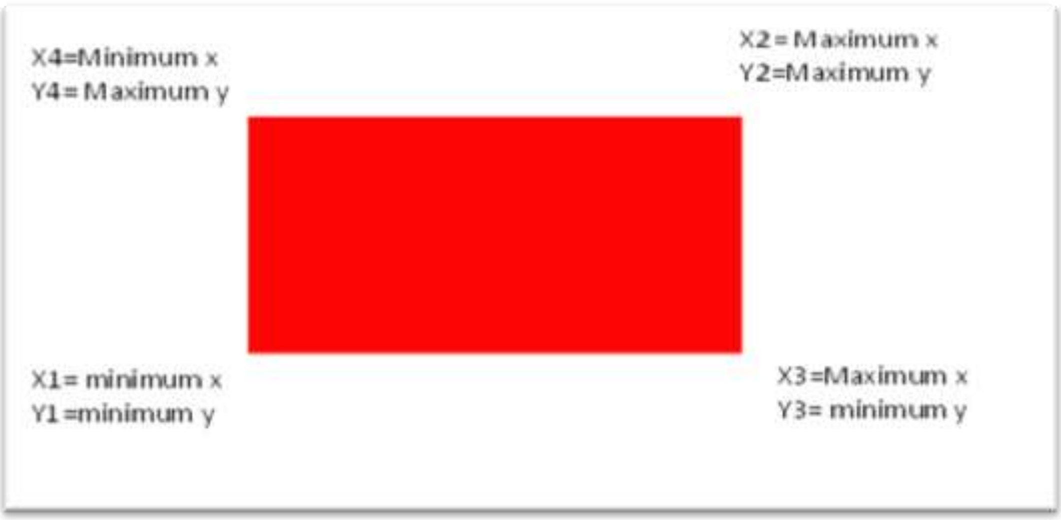


Figure 5-5: Corner 1

Let the lower left corner has coordinates (x1, y1), upper left corner has coordinates (x4, y4), lower right corner has co-ordinates (x3, y3), and upper right corner has coordinates (x2, y2). The corners for figure a are shown in Fig. 5-5

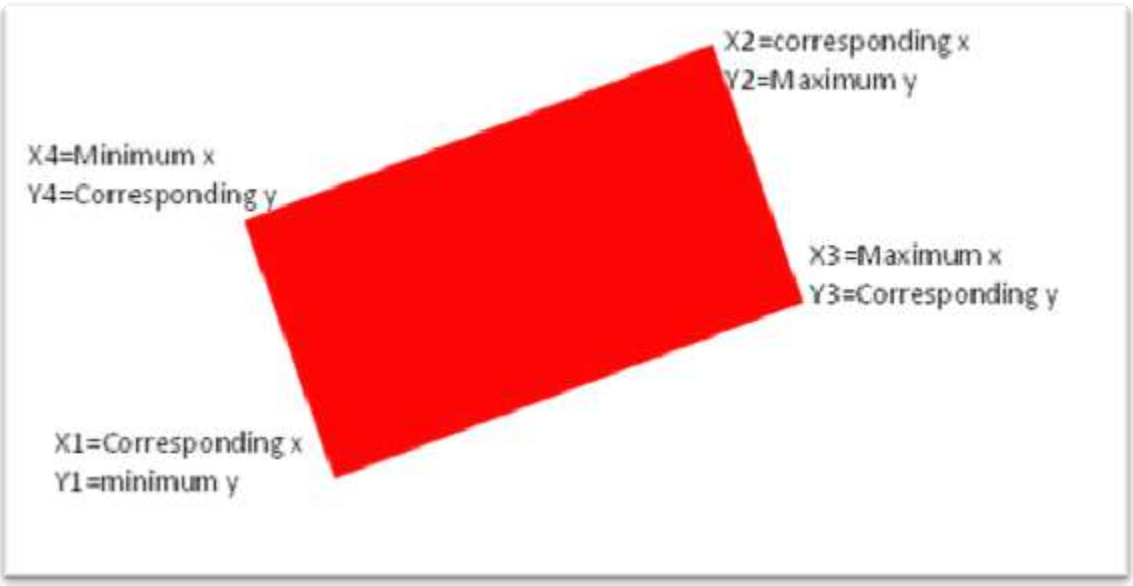


Figure 5-6: Corner 2

Now corners are shown in Fig. 5-7. The Fig. 5-6 the problem was that maximum of x and y and minimum of x and y include almost all the corners so to find the required corners for a completely horizontal paddle, the values of maximum of x,y and minimum of x,y were found. These values were used to find the corners as

X1, y1=minimum of x, y;

X2, y2=maximum of x, y

X3, y3=maximum of x, minimum of y;

X4, y4=minimum of x, maximum of y

The four required corners are found. The next step is to find the velocity. For that we need to have the length or distance moved by the paddle in the direction of camera and time. The distance can be calculated by the difference in the area. The area is calculated by using the corners of the paddle. For calculating the distance, two images having paddle at a known distance from each other were used. From that a factor was calculated that relates the difference in the area with the distance moved by the paddle with the help of an equation formulated after many experiments. The equation is

Length= (current_area/previous_area-1)/0.21

Whereas current area is the area of the image for which the velocity is calculated, the previous area corresponds to the area of paddle against which the change in area is measured.

Now the time can easily be found with the help of an inbuilt command tic toc, to calculate the execution time. As velocity is length/ time so the velocity of the paddle was calculated with the help of this algorithm.

4.3.4 Direction detection

The direction of the paddle is found using the position of the paddle. The direction includes up, down, left and right movement of paddle. This is simply calculated using two images. When the paddle moves in the vertical direction the y coordinate of the centroid is compared with the previous value of y co-ordinate of the centroid. If the present value is more the paddle has moved up and if the value is less the paddle has moved downwards. Similarly when the paddle moves in the horizontal direction the x coordinate of the centroid is compared with the previous value of x co-ordinate of the centroid. If the present value is more the paddle has moved in the right direction and if the value is less the paddle has moved in the left direction.

The extent to which the paddle has moved up, down, right or left can be found with the help of ratios of the present values with that of the previous values.

To further refine the calculation of direction of the paddle, one condition was added to all the up, down right and left movement of paddle. The condition was added after observation of human reflexes when he plays tennis. For the upward movement the paddled along with upward movement will be tilted upwards. This tilt can be found out with the help of corners calculated in above section. The upward tilt is confirmed with the help of the fact that the lower side of the paddle is greater in length as compared to the upper side. Similarly the tilt is downward if the lower side of the paddle is smaller in length as compared to the upper side. Similarly the tilt towards right is confirmed as the left side of the paddle is greater in length than the right side and the tilt to the left is confirmed as the right side of the paddle is greater in length than the left side.

The tilt along with the direction of motion of the paddle specifies the direction in which the ball is to be thrown once it hits the paddle.

4.5 Phase 3

The design that was followed to complete the final phase of the game is shown in the Fig. 4-7 The tasks of finding position, velocity and direction/tilt on images is completed now moving onto acquiring a video from the camera to be used in applying the algorithms developed above in real time. First step was to interface the camera with the Matlab. There is an inbuilt function for that which detects every camera attached with the computer and the required camera that is to be used for the game can be selected. The camera is then used to acquire video which is processed further to detect paddle, remove background, find the position of paddle, use the formula acquired above to find the velocity of the paddle, and finally find the direction/tilt. With implementation of the modules mentioned, video processing part is finished.

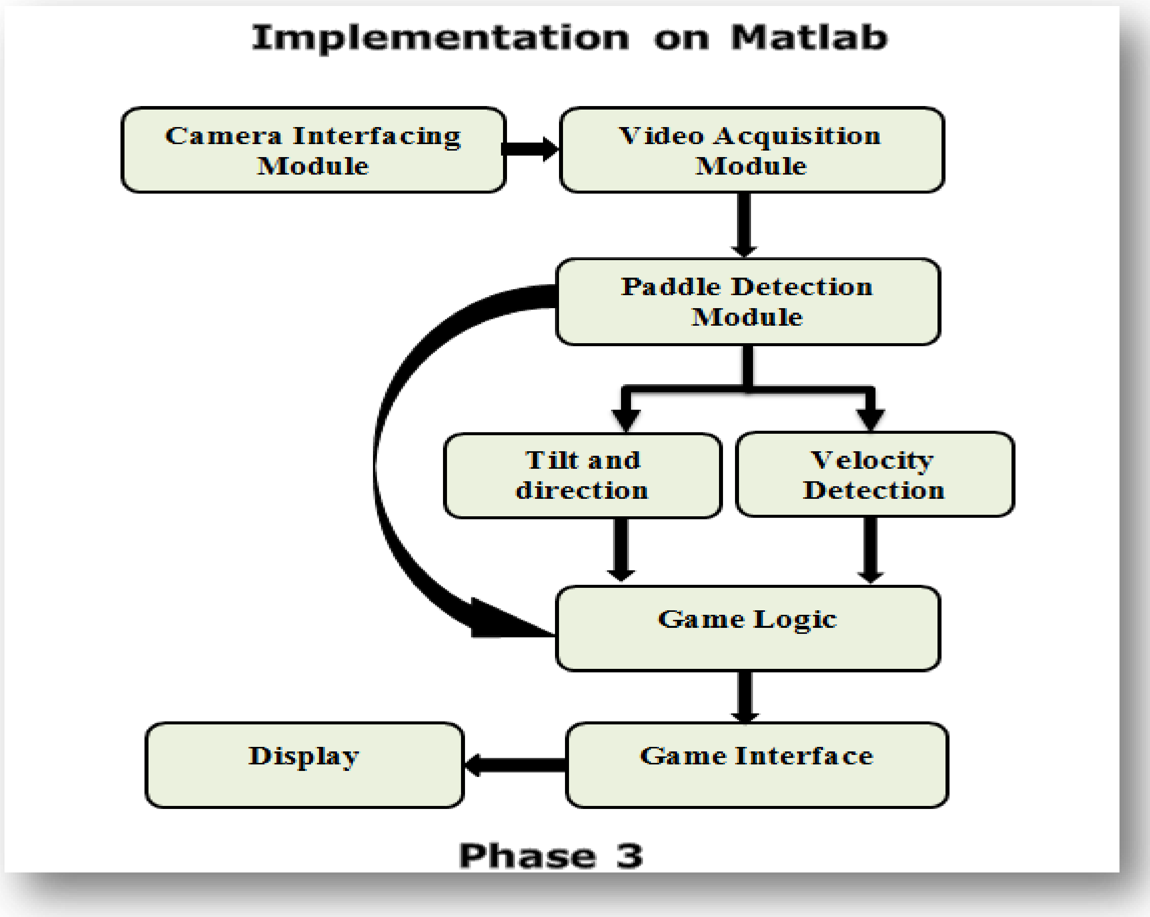


Figure 4-7: Complete Design phase 3

The camera acquires the video with resolution of 320*240 and frame rate of 30 frames that was set for the incoming video stream. To implement image processing algorithms, the images are required. For that the snapshot form the video was acquired. The snapshot or frame was first used to detect a red colored paddle using color detection algorithm explained in the phase two.

The second step was to remove the background so that the corners can be detected. Background removal algorithms were not used. Instead the Red green and blue components of the image were separated. The red part of the image was retained as it is, the blue and green parts of the image were multiplied with zero and the image was recreated. Only the red colored part remained in the video, other portion of the video showed a black color. This method reduced delay but the problem with this method is that any red color in the background will not be removed.

The third step was to find the corners of the paddle from the video input. Inbuilt command detects all the corners along the edges. To filter out the required corners, the orientation of paddle was made use of. Three basic orientations were used and the paddle corners were detected using different logic as explained in phase 2. The four corners are required both for velocity and direction/tilt calculation.

The next step after the corner detection was to calculate the velocity and direction/tilt of the paddle. The velocity was calculated as mentioned in phase two with the help of basic equation of physics that is the ratio of distance to time. The distance moved forward by the paddle was found with the help of difference in the area of paddle. With the help of equation derived in phase two, the difference in area was related with length. The length along with the execution time was used to calculate the velocity. The direction/tilt was calculated with the combination of two algorithms. The right left and up down movement was found by comparing the central position of the paddle or centroid with the position of the paddle for previous frame. The x co-ordinates of paddle were compared for right and left movement and y co-ordinates were compared for up and down movement of paddle. The direction was further confirmed with the help of one more factor called as tilt. The paddle will be tilted up if it is moving upward which is confirmed with the help of the fact that the lower side of the paddle is greater in length as compared to the upper side. Similar is that for right, when the paddle is moving right it is tilted towards right which is confirmed as the left side of the paddle is greater in length than the right side, when the paddle is moving downwards it is tilted down which is confirmed as the lower side of the paddle is smaller in length as compared to the upper side. When the paddle is moving left it is tilted towards left which is confirmed as the right side of the paddle is greater in length than the left side. The tilt can easily be handled with the help of ratio of sides of paddle. The tilt along with the change in the center position of paddle gives the direction in which the paddle is to be moved.

4.5.1 Game logic

The game logic takes input of position of paddle, velocity and direction of paddle and calculates the (x, y, and z) co-ordinates of the ball after catering for the direction, tilt velocity and paddle position and updates this co-ordinate continuously as the ball moves forward and backward. Computation of the co-ordinates for the ball also includes the gravity effect and friction (bounce) after hitting the horizontal surface of the play field/table. Game logic set the conditions for as to when the ball is hit or missed.

To calculate the ball coordinate a simple approach was used that was the movement of tale tennis ball can be modeled with the help of a sine or cosecant with an absolute value as the equation of motion were difficult to implement. The two factors that were to be added were gravity effect bouncing back of ball after hitting the table. The cosecant wave was used to initially to calculate the coordinates of the ball incorporated the bouncing. As the ball was not model on motion and kinetics laws, an alternative approach was used to incorporate gravity. A negative exponential was multiplied with the cosecant wave gave the same effect as that of gravity.

The effect of velocity was added (velocity cannot be added directly as laws of motion and kinetics are not used for modeling) such that the ball moves fast and slow with respect to velocity. It was implemented by using the approach that if the velocity is less the coordinates were calculated with less spacing between two consecutive coordinate points. More time is required by the ball to cover more points to reach to the other end of the table. If the velocity is more the spacing between the consecutive points is more. Thus the ball has to cover fewer points for reaching the other end of the table. Thus taking less time and eventually giving the effect that ball is moving with more speed.

Now adding the up, down, right and left direction was to be done. The right and left movement and the extent to which the paddle has moved right or left were used. Along with the decaying and bouncing of ball in forward direction, an additional factor was added according to the ratio of right and left movement of paddle such that the ball moves right or left as well. The up and down movement of paddle was incorporated by increasing and decreasing the height of the bounce with respect to the ratio of up or down movement respectively.

4.5.2 Game interface

The game interface includes an interface with a table and ball moving on the table with the desired direction and velocity. The interface was designed on Matlab. A static table was made on a three dimensional plot in Matlab. The code of the figure was generated and the figure was set as a background. Now to show a ball moving in the three dimensions a marker was used. The position of marker was updated as per the coordinate calculated in the game logic. The ball or marker takes the coordinates from the game logic and was shown moving on a table in the desired direction with the desired tilt.

CHAPTER 5

Analysis and Evaluation

5.1 Overview of the Chapter

The chapter includes the results to validate all the work done to complete the development phases. Kinect results include the skeletal identification, joint access and integration of the two to develop the cursor application. The video processing results include paddle detection using red color detection, the background removal results, after that centroid and corner detection, then velocity calculation results after that direction detection results. This chapter also includes the logic validation of game logic developed.

5.2 Results of Kinect Implementation

5.2.1 Getting a Joint position

We at first got the joint positions after installing and programming our kinect [Annex A]. Our Results are shown in the Fig. 5-1

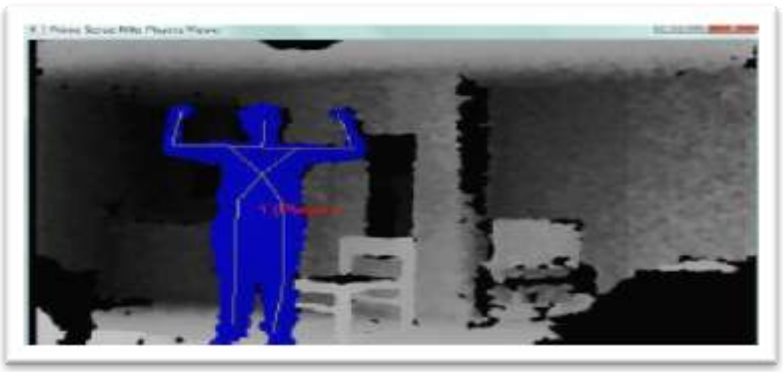


Figure 5-1: Results of Joint position

5.2.2 Skeletal tracking

We set our field view by adjusting the angles from the software, after that we were able to set the vicinity of the correct tracking of whole skeleton. Our results are shown in the Fig. 5-2

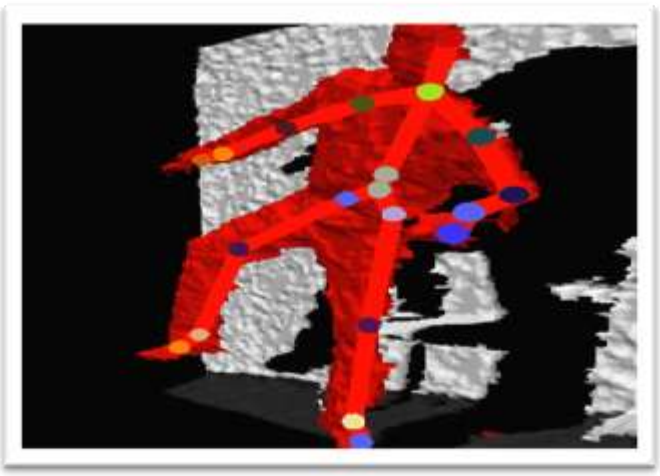


Figure 5-2: Result of Skeletal tracking

5.2.3 Integration

To see our objects we integrate the above two results. The integration is shown in the Fig. 5-3

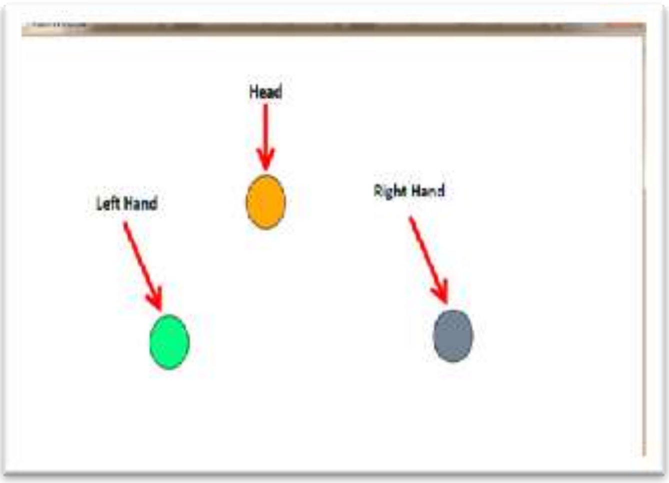
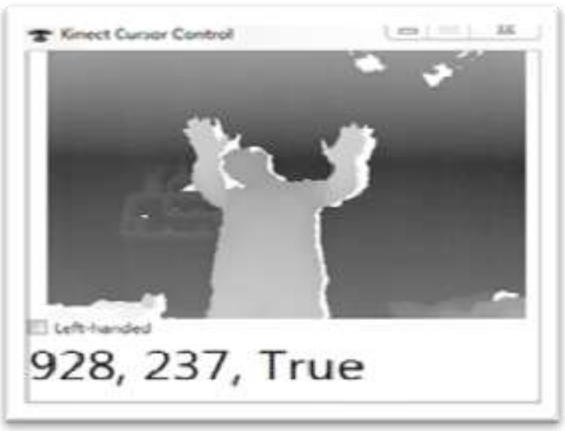


Figure 5-3: Integration Results

5.2.4 Cursor Control Application Results

We programmed the kinect to use only one object i.e. hand only in the application and results are shown in the Fig. 5-4



928, 237, True

Figure 5-4: Cursor Application result

5.2.5 Graphic Interface

Our game interface after developing it in adobe flash player is shown in the

Figs.5-5, 5-6

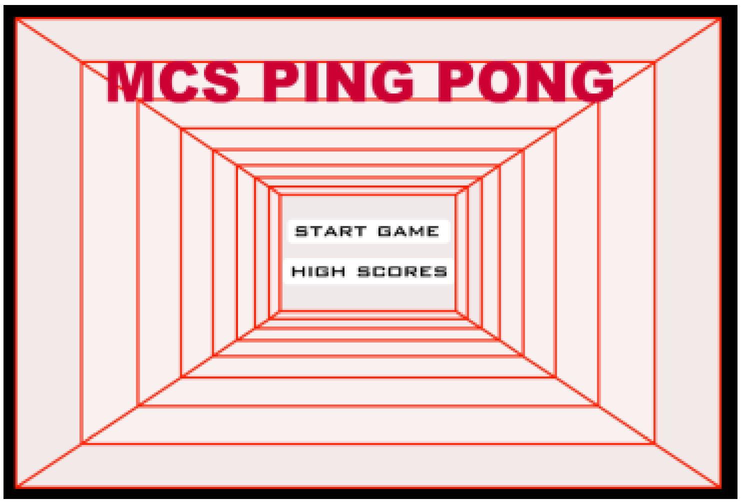


Figure 5-5: Game startup Layout

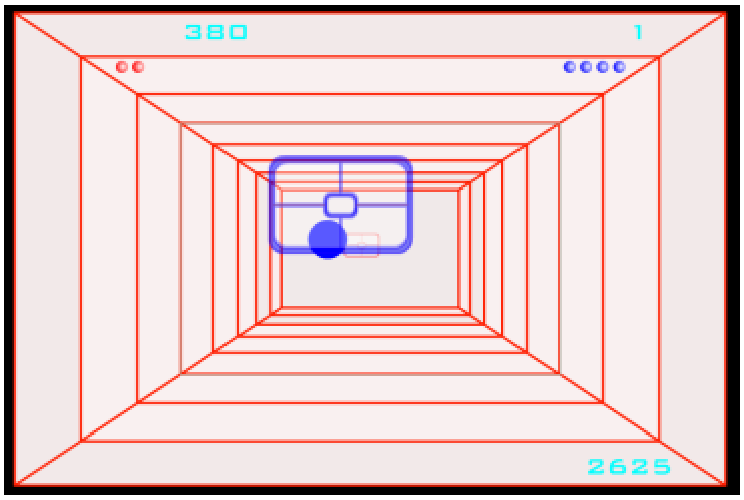


Figure 5-6: Running Game pic

5.2.5 Application results

After the interfacing of our kinect cursor control with the flash game we got our required virtual control of our game as shown in the Fig.

5-7

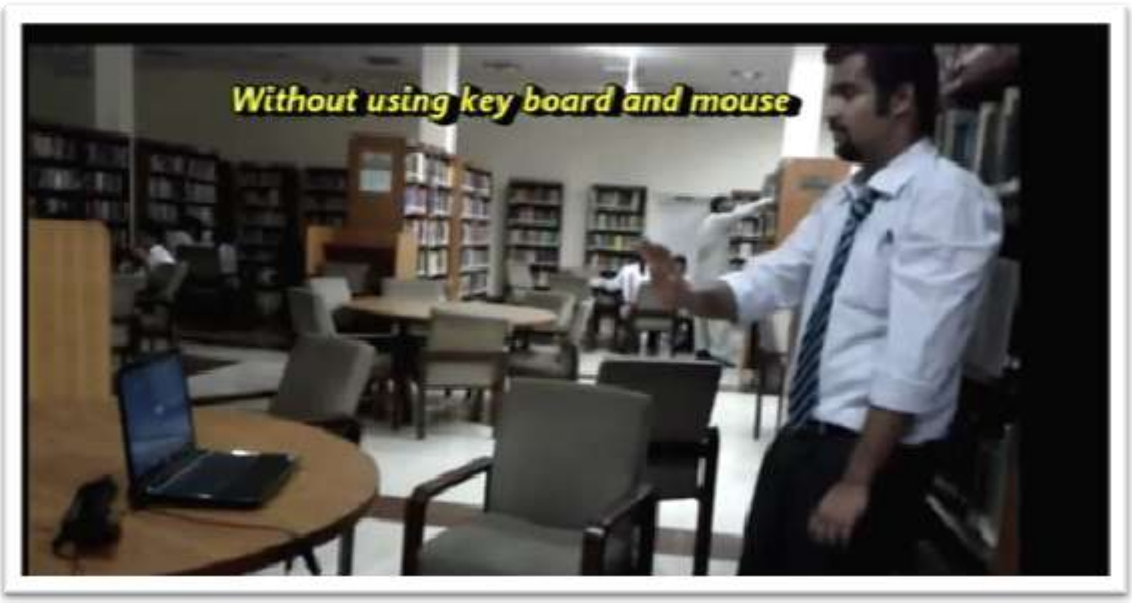


Figure 5-7: Complete application results

5.3 Results using integrated digital camera

Kinect device was replaced with the digital camera. From the digital camera the video was acquired. After acquiring the video snapshots of video were taken. The video processing algorithms were implemented on the frames/snapshots. The first step was to detect the red colored paddle from the video. Then the centroid of this red colored square paddle was found. After that the pixels other than red colored pixels were multiplied with zero giving result in red and black color image. After the removal of extra pixels the corners of paddle were detected and four corners out of all corners were filtered out. The velocity was calculated and finally the tilt was calculated.

5.3.1 Results of paddle detection

Paddle detection module detects the paddle from the real time video. We make use of the color detection algorithm to detect the red color of paddle.. The results are shown in Fig. 5-8

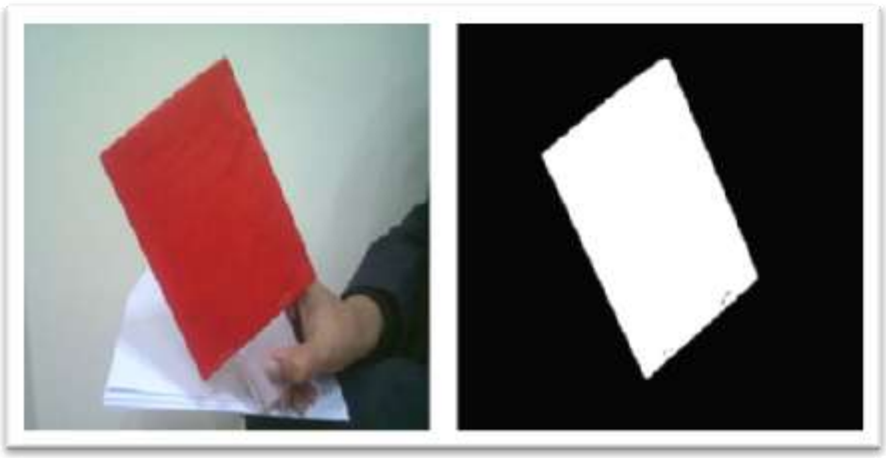


Figure 5-8: Paddle detection

Algorithm of color detection involves conversion of image into gray scale, and then the gray scale image is subtracted from the red component in the image. Then the image is converted in binary and after that areas having less than 300px are removed all connected components are labeled and then the image is ready for further processing

5.3.2 Position detection results

After the detection of paddle, position of paddle is calculated in the frame. The position of paddle is required to be found for the calculation of direction. The results are shown in Fig. 5-9

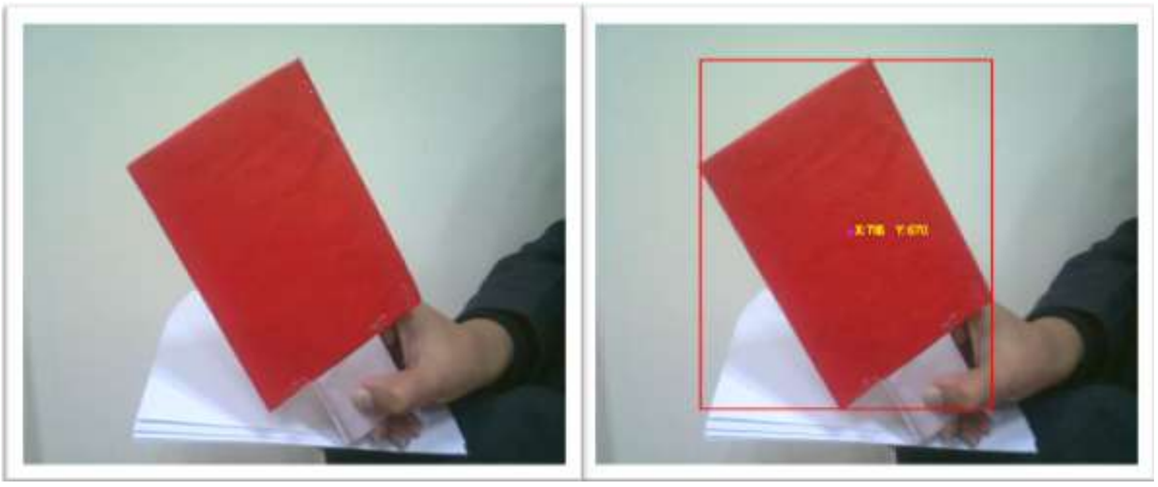


Figure 5-9: Paddle Position

Position of paddle can simply be found by finding the centroid. The centroid is found by using inbuilt Matlab function.

5.3.3 Background removal

A simple approach was used to remove. Red, green and blue components of the image were separated and only the red part of the image was retained as it is, the blue and green parts of the image were multiplied with zero and the image was recreated resulting in a black background and red paddle. The results are shown in Fig. 5-10

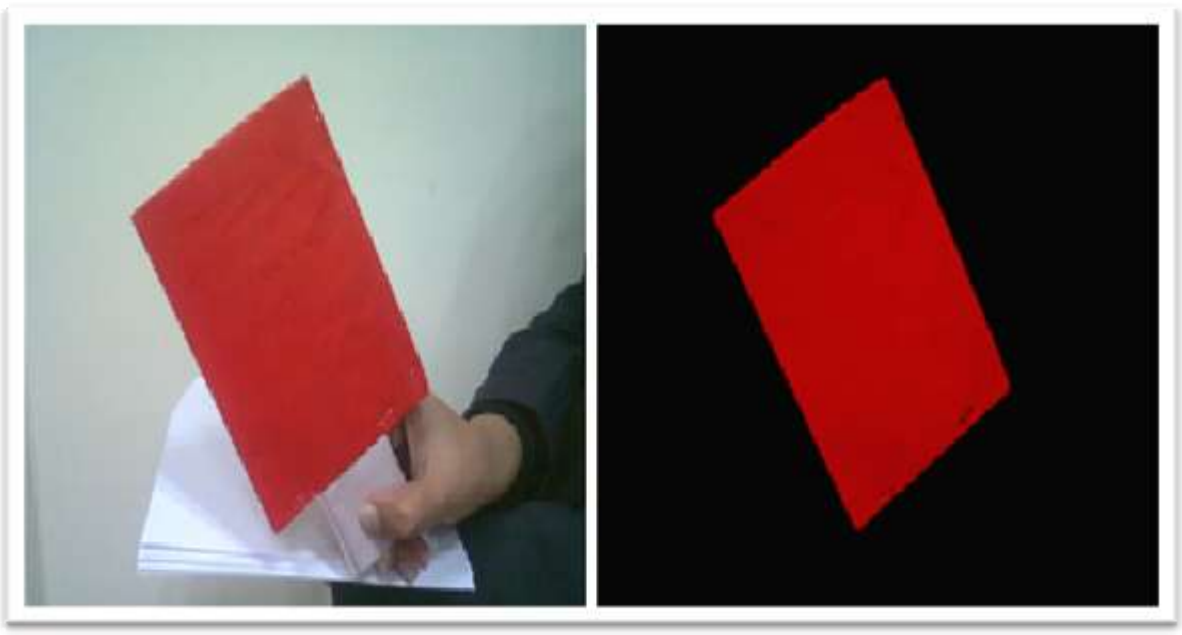


Figure 5-10: Back ground Removal

5.3.4 Corner detection

The corners of the paddle that were detected by the inbuilt command but all the boundary pixels of the paddle were detected as corners. The results of three basic orientations are shown in the Fig. 5-11. To filters the required corners out of all the boundary corners, first step was to store all the corner values that include x coordinate and y coordinate of the corner point. The second step was to filter the required corners. The corners having minimum value of x, maximum value of x, minimum value of y and maximum value of y were filtered out. So the four pairs of corners valued were maximum of x and the value of y corresponding to maximum of x, maximum of y and the value of x corresponding to maximum of y, minimum of x and the value of y corresponding to minimum of x, minimum of y and the value of x corresponding to minimum of y.

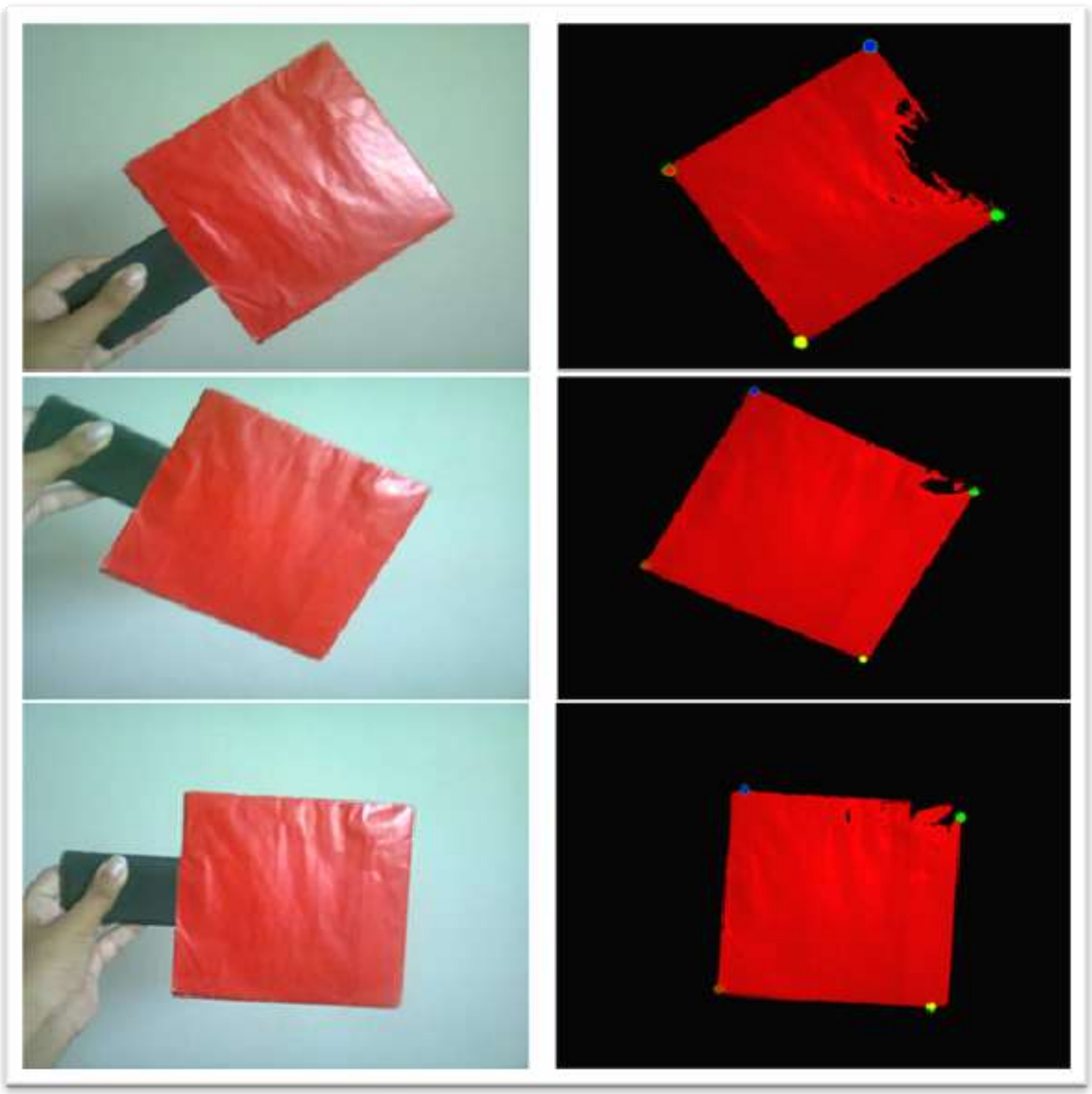


Figure 5-11: Corner Detection with three basic orientations

5.3.4 Velocity detection results:

The length or distance moved by the paddle in the direction of camera is required to be found. The distance is calculated by the difference in the area. The area is calculated by using the corners of the paddle. For calculating the distance, two images having paddle at a known distance from each other were used. The distance was 11.69 inches or 26.69 cm. as the area for the paddle in both the frames is known. The area of frame 1 was found to be 98815 and that of frame 2 was 404417. As the area is increasing from frame 1 to frame 2 so that means the paddle is moving toward the camera. First the ratio of area was calculated, that was 4.0927.

Keeping in mind all the factors mentioned above the equation can be written as

Length= (area of frame 2/area of frame 1 - 1)*x

Whereas x is a factor with which the length and area is related. Note that area and length are not linearly related but we can take the relationship to be linear for small length in which the game is to be played.

Length=.2696 m

Area of frame 2/area of frame 1=4.0927

x=length/ (area of frame 2/area of frame 1 - 1)

x=0.096001

Now if the paddle does not move, the ratio of areas will be 1 and the length calculated is zero. This validates that the equation can be used to find length.

The same procedure was adopted for two different frames at different lengths to validate the value of x. the value of x ranges from 0.08-0.16 owing to the non-linearity in the relation of area and length. The value was used that was 0.13.

The further validation was done with the help of two frames with the distance between paddles equal to 0.21m. The ratio of the area of paddle was 2.4565. When the ratio of area was input to the equation the length was found to be 0.1893m. This length is close to the original value.

Now the time can easily be found with the help of an inbuilt command tic toc, to calculate the execution time of one loop. The execution time was found to be 0.15 seconds. As velocity is length/ time so the velocity of the paddle was calculated with the help of this algorithm.

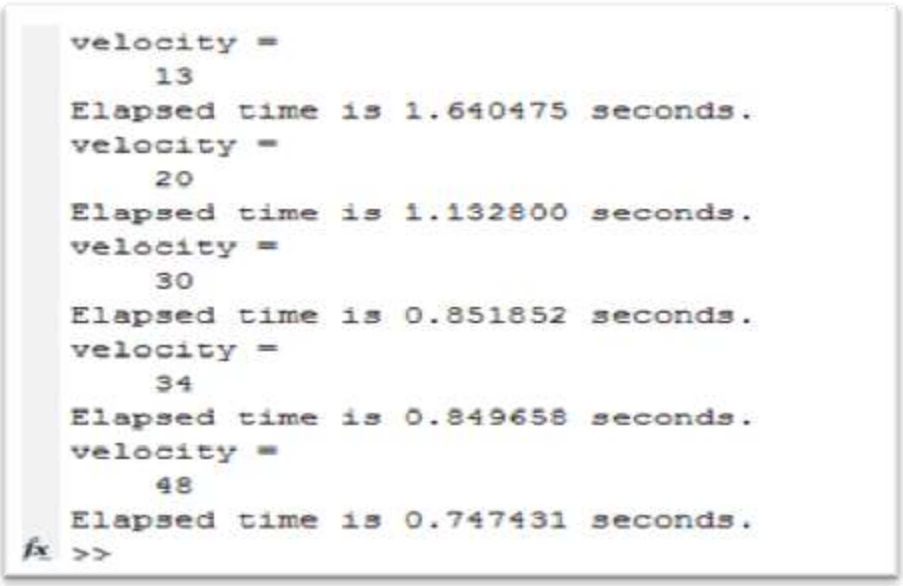
5.3.5 Direction detection

The direction detection includes the up, down, right and left direction. The direction of the paddle is found using the position of the paddle. This is simply calculated by the comparison of central points of two frames. When the paddle moves in the vertical direction the y coordinate of the centroid is compared with the previous value of y co-ordinate of the centroid. If the present value is more the paddle has moved up and if the value is less the paddle has moved downwards. When the present value is less, then the ratio will become present y co-ordinate/previous y coordinate. Similarly when the present value is greater then the ratio will become previous y co-ordinate/ present y coordinate. This always generates positive values up to 1. The closer the value is to zero the more the paddle has moved in up or down direction. This up and down movement was validated or confirmed with the upward tilt in case of upward movement and down tilt in case of downward movement. The up and down tilt is found by the ratio of upper and lower side of the paddle. When the ratio is greater than 1 the paddle id tilted down and when the tilt is less than 1 the paddle is tilted up.

Similarly when the paddle moves in the horizontal direction the x coordinate of the centroid is compared with the previous value of x co-ordinate of the centroid. If the present value is more the paddle has moved in the right direction and if the value is less the paddle has moved in the left direction. When the present value is less, then the ratio will become present x co-ordinate/previous x coordinate. Similarly when the present value is greater than the ratio will become previous x co-ordinate/ present x coordinate. This always generates positive values up to 1. The closer the value is to zero the more the paddle has moved in right or left direction. This right and left movement was validated or confirmed with the right tilt in case of movement in right hand direction and left tilt in case of leftward movement. The right and left tilt is found by the ratio of right to left side of the paddle. When the ratio is greater than 1 the paddle is tilted leftwards and when the tilt is less than 1 the paddle is tilted rightwards.

5.3.6 Game logic and interface

The game logic includes the calculation of co-ordinates of the ball while incorporation the effects of gravity, bouncing, velocity and direction of the paddle. The effect of gravity and bouncing were added. This is validated from the fact that the values of coordinates decay and bounce was ensured as the ball movement was initially modeled with cosecant waveform. The results are shown in Fig. 5-12



```
velocity =
    13
Elapsed time is 1.640475 seconds.
velocity =
    20
Elapsed time is 1.132800 seconds.
velocity =
    30
Elapsed time is 0.851852 seconds.
velocity =
    34
Elapsed time is 0.849658 seconds.
velocity =
    48
Elapsed time is 0.747431 seconds.
fx >>
```

Figure 5-12: Validation Results

The velocity effect was incorporated that can be validated from the value of velocity and time for the ball to complete on complete cycle. For high velocity the time taken by the ball is less than that for the low velocity.

Our game graphics are shown in the Fig. 5-13

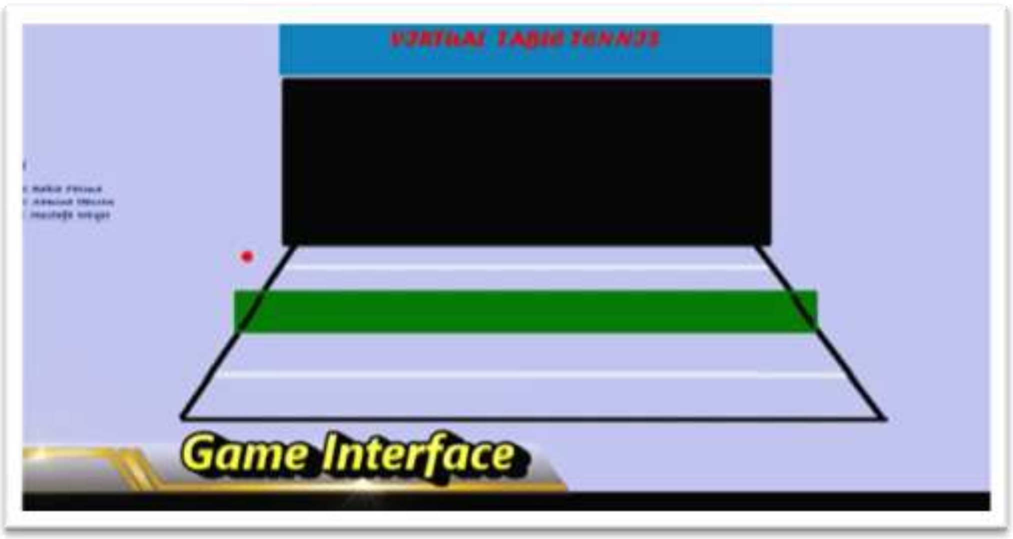


Figure 5-13: Game Interface

Similarly the direction was incorporated as the paddle moves right or left the value according to the ratio was added to or subtracted from the straight movement of the ball such that it moves right or left. The up and down movement was validated from the height of the bounce of the ball. This was validated from the coordinates of the ball found when the ball paddle was moved up or down and right or left.

5.3.7 Conclusion

All the results of paddle detection, position calculation, background removal, corner detection, velocity calculation and direction detection has been shown in this chapter. The accuracy of algorithms used has been validated with the help of results .

CHAPTER 6

Future Work

6.1 Overview of the Chapter

The chapter aims to pin point the future work that can be done on the project. This chapter includes two sections. One section includes the improvements that can be done in the existing algorithms and techniques used in the project so as to develop the project in a better form. The second section includes the future work. This section includes the additional work that can be done on the project to make the project feasible on commercial level.

6.2 Future Developments

The project aims to replace the conventional controllers that are keyboard and mouse used to play table tennis with a real paddle or racket.

6.2.1 Improvements (Section 1)

We can enhance our kinect implementation results further. By using two kinect devices one can allow multiple users to play simultaneously. By changing the game graphics and software part and nature of the game one can use the same application for multiple diversified games such as combating, car race, virtual tracking etc.

6.2.2 Improvements (Section 2)

The second approach was to use a digital camera to play the table tennis game. The project can be carried out further in much way more over the techniques used can be improved to make the game better and fit for playing.

The improvements that can be done in future are described below:

There is a limitation in the technique used to remove the background of the video, which is that if red color appears in the background, it will not be discarded even if it does not belong to the paddle. This presents a serious problem as the ball can falsely be hit by the red colored object in the background. This problem can be solved by implementing any conventional background removal algorithms thus giving more precision to the game.

Velocity calculation algorithm has a limitation that the calculation of the length with the help of the derived equation does not cater for the nonlinear relation between the area of the paddle and distance covered in the horizontal direction. Further experimentation can be done to derive an empirical formula that includes the nonlinear relationship of the area with that of distance so that the velocity can be calculated more accurately with the help of difference in the area of the paddle.

6.3 Future work

Only one digital camera is used in the project which limits the precision of velocity and tilt and direction detection algorithms. Two or three cameras can be used. By using three cameras the length, horizontal angle and vertical angle can be found directly with using minimum experiment based algorithms. Thus the use of three cameras lead to precision yet the input from three different cameras is difficult to handle.

The tilt detection can be developed further to find the actual horizontal (azimuth) and vertical (elevation) angles of the paddle if single camera approach is carried forward. The calculation of actual horizontal and vertical angle will result the ball to move in the exact direction that was intended by the user or played.

The future work can be carried out by using parallel processing instead of serial. In the current game the parameter calculation and the game itself runs in series. First the parameters i.e. position, corner, velocity and direction/tilt are calculated then passed on to the game logic for finding the coordinates of the ball and then the ball is displayed moving. The execution in series tends to add a delay between two consecutive hits by the paddle. This delay can be reduced to a certain limit but the delay of execution of algorithms along with passing the values to the game logic will always be there. If the parallel execution of algorithms, ball coordinate calculation and display is done the only delay will be to pass the values to the game logic and display. Thus the delay involved will be minimal as compared to that of the serial execution.

Another work that can be done is to use the laws of motion and kinetics to model the movement of the ball instead of modeling the ball movement with alternative approach. This would result in the exact co-ordinate calculation for the ball incorporating gravity, the third law of motion and energy conservation laws.

The interface is developed on Matlab, which can be replaced with the interface on software like visual c. this would make the project interface more attractive.

Once fully developed, the project can be used as a virtual trainer for the beginners of the game. As the compulsion of having tabled and other playing accessories is no more there, anyone can simply play the game at home. Moreover the game can also be used by physically disabled people.

6.4 Conclusion

The game can be improved and developed further to make it look like an actual table tennis game. As the Matlab game only requires a normal digital camera, it presents a very cost effective solution to its substitutes i.e. Kinect or x-box. Catering for the limitations and adding advance algorithms would result in an optimized game.

The fully developed game can be used on commercial level as a cost effective replacement of other substitutes. Also can be used as virtual trainers for starters and disabled people.

BIBLIOGRAPHY

[1]Image Processing Techniques for Video Content Extraction

Inês Oliveira, Nuno Correia, Nuno Guimarães INESC/IST, R. Alves Redol, 9, 6o, 1000 Lisboa

[2]EURA SIP Journal on Image and Video Processing 2009, **2009**: 859371 doi:10.1155/2009/859371

http://jivp.eurasipjournals.com/content/2009/1/859371

[3]Stephan Rusdorf, Guido Brunnett, Mario Lorenz, and Tobias Winkler, “**Real-TimeInteraction with a Humanoid Avatar in an Immersive Table Tennis Simulation,”** IEEE transactions on visualization and computer graphics, vol. 13, no. 1, january/february 2007

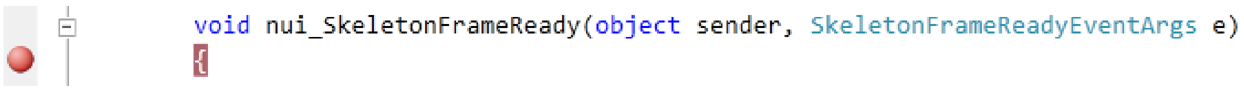
[4]Alp V. Asutay, Arun P. Indugula,Christoph W. Borst, “Virtual Tennis: A Hybrid Distributed Virtual Reality Environment with Fishtank vs. HMD,” IEEE Computer Society Washington, DC, USA ©2005

Appendix A

Initializing the Run time

```
        Runtime nui = new Runtime();

    nui.Initialize(RuntimeOptions.UseSkeletalTracking);
    nui.SkeletonFrameReady += new
    EventHandler<SkeletonFrameReadyEventArgs>(nui_SkeletonFrameReady);
    void nui_SkeletonFrameReady(object sender,
     SkeletonFrameReadyEventArgs
     e)
    {
    }
```



```
void nui_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
{
```

Setting Objects

```
SetEllipsePosition(headEllipse, skeleton.Joints[JointID.Head]);

SetEllipsePosition(leftEllipse, skeleton.Joints[JointID.HandLeft]);

SetEllipsePosition(rightEllipse, skeleton.Joints[JointID.HandRight])
```

Appendix B

Skeletal tracking

```
KinectSensor kinect = null;

  void StartKinectST()
  {
    kinect = KinectSensor.KinectSensors.FirstOrDefault(s => s.Status ==
KinectStatus.Connected); // Get first Kinect Sensor
    kinect.SkeletonStream.Enable(); // Enable skeletal tracking

    skeletonData = new Skeleton[kinect.SkeletonStream.FrameSkeletonArrayLength];
// Allocate ST data

    kinect.SkeletonFrameReady += new
EventHandler<SkeletonFrameReadyEventArgs>(kinect_SkeletonFrameReady); // Get
Ready for Skeleton Ready Events

    kinect.Start(); // Start Kinect sensor
  }
```

To enable skeletal tracking, call the INuiSensor::NuiSkeletonTrackingEnable method. To receive information about the recognized users, call theINuiSensor::NuiSkeletonGetNextFrame method.

```
    // Call StartKinectST once at application start.
    HRESULT MyApplication::StartKinectST()
    {
      m_hNextSkeletonEvent = NULL;

      // Initialize m_pNuiSensor
      HRESULT hr = FindKinectSensor();
      if (SUCCEEDED(hr))
      {
        // Initialize the Kinect and specify that we'll be using skeleton
        hr = m_pNuiSensor-
>NuiInitialize(NUI_INITIALIZE_FLAG_USES_SKELETON);
        if (SUCCEEDED(hr))
        {
          // Create an event that will be signaled when skeleton data is available
          m_hNextSkeletonEvent = CreateEventW(NULL, TRUE, FALSE, NULL);

          // Open a skeleton stream to receive skeleton data
          hr = m_pNuiSensor->NuiSkeletonTrackingEnable(m_hNextSkeletonEvent, 0);
        }
      }
      return hr;
    }

    // Call UpdateKinectST on each iteration of the application's update loop.
    void MyApplication::UpdateKinectST()
    {
      // Wait for 0ms, just quickly test if it is time to process a skeleton
      if ( WAIT_OBJECT_0 == WaitForSingleObject(m_hNextSkeletonEvent, 0) )
      {
        NUI_SKELETON_FRAME skeletonFrame = {0};

        // Get the skeleton frame that is ready
        if (SUCCEEDED(m_pNuiSensor->NuiSkeletonGetNextFrame(0,
&skeletonFrame)))
        {
          // Process the skeleton frame
          SkeletonFrameReady(&skeletonFrame);
        }
      }
    }
```

Appendix C

Integration

```
private void DrawTrackedSkeletonJoints(JointCollection jointCollection)
    {
      // Render Head and Shoulders
      DrawBone(jointCollection[JointType.Head],
jointCollection[JointType.ShoulderCenter]);
      DrawBone(jointCollection[JointType.ShoulderCenter],
jointCollection[JointType.ShoulderLeft]);
      DrawBone(jointCollection[JointType.ShoulderCenter],
jointCollection[JointType.ShoulderRight]);
      // Render Left Arm
      DrawBone(jointCollection[JointType.ShoulderLeft],
jointCollection[JointType.ElbowLeft]);
      DrawBone(jointCollection[JointType.ElbowLeft],
jointCollection[JointType.WristLeft]);
      DrawBone(jointCollection[JointType.WristLeft],
jointCollection[JointType.HandLeft]);
      // Render Right Arm
      DrawBone(jointCollection[JointType.ShoulderRight],
jointCollection[JointType.ElbowRight]);
      DrawBone(jointCollection[JointType.ElbowRight],
jointCollection[JointType.WristRight]);
    }
    private void DrawBone(Joint jointFrom, Joint jointTo)
    {
      if (jointFrom.TrackingState == JointTrackingState.NotTracked ||
      jointTo.TrackingState == JointTrackingState.NotTracked)
      {
        return; // nothing to draw, one of the joints is not tracked
      }
      if (jointFrom.TrackingState == JointTrackingState.Inferred ||
      jointTo.TrackingState == JointTrackingState.Inferred)
      {
        DrawNonTrackedBoneLine(jointFrom.Position, jointTo.Position);
      }

      if (jointFrom.TrackingState == JointTrackingState.Tracked &&
      jointTo.TrackingState == JointTrackingState.Tracked)
      {
        DrawTrackedBoneLine(jointFrom.Position, jointTo.Position);  // Draw bold lines
if the joints are both tracked
      }
    }
```

Appendix D

<u>Joint Access</u>

```
Private Sub Window_Loaded(ByVal sender As Object, ByVal e As
RoutedEventArgs)
   ' hook up our events for depth and sekeltal tracking
   AddHandler _runtime.DepthFrameReady, AddressOf _runtime_DepthFrameReady
   AddHandler _runtime.SkeletonFrameReady, AddressOf
_runtime_SkeletonFrameReady
   Try
     ' tell Kinect we need the depth buffer and skeletal tracking
      _runtime.Initialize(RuntimeOptions.UseDepth Or
RuntimeOptions.UseSkeletalTracking)
   Catch ex As Exception
      MessageBox.Show("Could not initialize Kinect device: " & ex.Message)
   End Try
   _runtime.SkeletonEngine.TransformSmooth = True
   Dim parameters As New TransformSmoothParameters()
   parameters.Smoothing = 0.7f
   parameters.Correction = 0.3f
   parameters.Prediction = 0.4f
   parameters.JitterRadius = 1.0f
   parameters.MaxDeviationRadius = 0.5f
   _runtime.SkeletonEngine.SmoothParameters = parameters
   Try
      ' open the depth stream at the proper resolution
      _runtime.DepthStream.Open(ImageStreamType.Depth, 2,
ImageResolution.Resolution320x240, ImageType.Depth)
   Catch ex As Exception
      MessageBox.Show("Could not open depth stream: " & ex.Message)
   End Try
End Sub
..

Private Sub _runtime_SkeletonFrameReady(ByVal sender As Object, ByVal e As
SkeletonFrameReadyEventArgs)
   For Each sd As SkeletonData In e.SkeletonFrame.Skeletons
      ' the first found/tracked skeleton moves the mouse cursor
      If sd.TrackingState = SkeletonTrackingState.Tracked Then
         ' make sure both hands are tracked
         If sd.Joints(JointID.HandLeft).TrackingState = JointTrackingState.Tracked
AndAlso sd.Joints(JointID.HandRight).TrackingState = JointTrackingState.Tracked
Then
            Dim cursorX, cursorY As Integer
            ' get the left and right hand Joints
            Dim jointRight As Joint = sd.Joints(JointID.HandRight)
            Dim jointLeft As Joint = sd.Joints(JointID.HandLeft)
            ' scale those Joints to the primary screen width and height
            Dim scaledRight As Joint =
jointRight.ScaleTo(CInt(Fix(SystemParameters.PrimaryScreenWidth)),
CInt(Fix(SystemParameters.PrimaryScreenHeight)), SkeletonMaxX, SkeletonMaxY)
            Dim scaledLeft As Joint =
jointLeft.ScaleTo(CInt(Fix(SystemParameters.PrimaryScreenWidth)),
CInt(Fix(SystemParameters.PrimaryScreenHeight)), SkeletonMaxX, SkeletonMaxY)
            ' figure out the cursor position based on left/right handedness
            If LeftHand.IsChecked.GetValueOrDefault() Then
               cursorX = CInt(Fix(scaledLeft.Position.X))
               cursorY = CInt(Fix(scaledLeft.Position.Y))
            Else
               cursorX = CInt(Fix(scaledRight.Position.X))
               cursorY = CInt(Fix(scaledRight.Position.Y))
            End If
            Dim leftClick As Boolean            ' figure out whether the mouse button is
down based on where the opposite hand is
            If (LeftHand.IsChecked.GetValueOrDefault() AndAlso
jointRight.Position.Y > ClickThreshold) OrElse ((Not
LeftHand.IsChecked.GetValueOrDefault()) AndAlso jointLeft.Position.Y >
ClickThreshold) Then
               leftClick = True
            Else
               leftClick = False
            End If
            Status.Text = cursorX & ", " & cursorY & ", " & leftClick
            NativeMethods.SendMouseInput(cursorX, cursorY,
CInt(Fix(SystemParameters.PrimaryScreenWidth)),
CInt(Fix(SystemParameters.PrimaryScreenHeight)), leftClick)            Return
         End If
      End If
   Next sd
End Sub
```

Appendix E

GUI

```
function varargout = development_gui(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @imageProc_OpeningFcn, ...
                   'gui_OutputFcn',  @imageProc_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before imageProc is made visible.
function imageProc_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for imageProc
handles.fileLoaded = 0;
handles.fileLoaded2 = 0;
set(handles.axes1,'Visible','off');
set(handles.axes2,'Visible','off');
set(handles.editPath, 'Visible', 'off');
set(handles.editSize, 'Visible', 'off');
set(handles.editComment, 'Visible', 'off');
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = imageProc_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;


if (handles.fileLoaded==1)
    handles.RGB=double(handles.RGB)/255;
   diff_im = imsubtract( handles.RGB(:,:,1), rgb2gray( handles.RGB));
   %Use a median filter to filter out noise
   diff_im = medfilt2(diff_im, [3 3]);
   % Convert the resulting grayscale image into a binary image.
   diff_im = im2bw(diff_im,0.18);
```

```
    % Remove all those pixels less than 300px
    diff_im = bwareaopen(diff_im,300);

    % Label all the connected components in the image.
    bw = bwlabel(diff_im, 8)
    % Here we do the image blob analysis.
    % We get a set of properties for each labeled region.
    stats = regionprops(bw, 'BoundingBox', 'Centroid');
    % Display the image
 axes(handles.axes2); imshow(handles.RGB);
    hold on
   %This is a loop to bound the red objects in a rectangular box.
    for object = 1:length(stats)
        bb = stats(object).BoundingBox;
        bc = stats(object).Centroid;
        rectangle('Position',bb,'EdgeColor','r','LineWidth',2)
        plot(bc(1),bc(2), '-m+')
        a=text(bc(1)+15,bc(2), strcat('X: ', num2str(round(bc(1))), '    Y: ',
num2str(round(bc(2)))));
        set(a, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color', 'yellow');
    end
    hold off
    handles.fileLoaded2 = 1;
    guidata(hObject, handles);
else
    h = msgbox('No primary file has been loaded!','Error','error');
end

% --- Executes on button press in centroid Button.
function SharpButton_Callback(hObject, eventdata, handles)
if (handles.fileLoaded==1)
r= handles.RGB(:,:,1); g= handles.RGB(:,:,2); b= handles.RGB(:,:,3);
diff=imsubtract(r,rgb2gray( handles.RGB));
bw=im2bw(diff,0.18);
area=bwareaopen(bw,300);
rm=immultiply(area,r);  gm=g.*0;  bm=b.*0;
i=cat(3,rm,gm,bm);
handles.RGB2 = im2bw(bw);
handles.RGB2 = imfill(handles.RGB2,'holes');
Ilabel = bwlabel(handles.RGB2);
stat = regionprops(Ilabel,'centroid');
axes(handles.axes2); imshow(handles.RGB2);hold on
for x = 1: numel(stat)
    plot(stat(x).Centroid(1),stat(x).Centroid(2),'ro');
end
    handles.fileLoaded2 = 1;
    guidata(hObject, handles);
else
    h = msgbox('No primary file has been loaded!','Error','error');
end
```

```
 function MotionButton_Callback(hObject, eventdata, handles)
if (handles.fileLoaded==1)
r=handles.RGB(:,:,1); g=handles.RGB(:,:,2); b=handles.RGB(:,:,3);
diff=imsubtract(r,rgb2gray(handles.RGB));
handles.RGB2=im2bw(diff,0.18);
area=bwareaopen(handles.RGB2,300);
rm=immultiply(area,r);  gm=g.*0;  bm=b.*0;
i=cat(3,rm,gm,bm);
threshold=128;
lbw=double(diff>threshold);
[Boundary] = bwboundaries(handles.RGB2,4); %Traces the boundary of each section
axes(handles.axes2); imshow(handles.RGB2); hold on;
colors=['b' 'g' 'r' 'c' 'm' 'y'];
for k=1:length(Boundary)
 perim = Boundary{k}; %Obtains perimeter coordinates (as a 2D matrix) from the cell
array
 cidx = mod(k,length(colors))+1;% Obtains colours for the plot
 plot(perim(:,2), perim(:,1),...
 colors(cidx),'LineWidth',2);
end
Coordmat = cell2mat(Boundary) %Converts the traced regions to a matrix
X = Coordmat(:,1)
Y = Coordmat(:,2) % This gives the edge coordinates in matrix form
cornerProbability = cornermetric(handles.RGB2);
cornerIdx = find(cornerProbability==max(cornerProbability(:)));
bw2 = imdilate(handles.RGB2,ones(3));
labeledImage = bwlabel(bw2);

cornerLabels = labeledImage(cornerIdx);
    handles.fileLoaded2 = 1;
    guidata(hObject, handles);
else
    h = msgbox('No primary file has been loaded!','Error','error');
end

if (handles.fileLoaded2==1)
    [file,path] = uiputfile('*.jpg','Save Secondary Image As');
    imwrite(handles.RGB2,[path file],'jpg');
else
    h = msgbox('No secondary file has been loaded!','Save Error','error');
end
function editSize_Callback(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editComment_Callback(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end
function editPath_Callback(hObject, eventdata, handles)
function editPath_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function sliderBright_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function sliderContrast_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
function editBright_CreateFcn(hObject, eventdata, handles)
function pushbutton22_Callback(hObject, eventdata, handles)
if (handles.fileLoaded==1)
r=handles.RGB(:,:,1); g=handles.RGB(:,:,2); b=handles.RGB(:,:,3);
diff=imsubtract(r,rgb2gray(handles.RGB));
handles.RGB2=im2bw(diff,0.18);
area=bwareaopen(handles.RGB2,300);
rm=immultiply(area,r);  gm=g.*0;  bm=b.*0;
i=cat(3,rm,gm,bm);
i=rgb2gray(i);
a=1;b=1;
bw = bwlabel(i, 8);
c = corner(handles.RGB2,300);
x1=0;y1=0;x2=0;y2=0;x3=0;y3=0;x4=0;y4=0;
[a,b]=find(c(:,1)<=min((c(:,1))),1,'first');
if(a>0)
x1=c(a,1);
y1=c(a,2);
end
end
[a,b]=find(c(:,2)<=min((c(:,2))),1,'first');
if(a>0)
x3=c(a,1);
y3=c(a,2);
end
end
axes(handles.axes2); imshow(handles.RGB2);
hold on
plot(x1,y1,'go','MarkerFaceColor','g','MarkerSize',10);
plot(x2,y2,'go','MarkerFaceColor','g','MarkerSize',10);
plot(x3,y3,'go','MarkerFaceColor','g','MarkerSize',10);
plot(x4,y4,'go','MarkerFaceColor','g','MarkerSize',10);
hyp=abs(((x1)/100)-((x2)/100))
base=abs(((x3)/100)-((x2)/100))
```

```
alt=abs(sqrt((hyp)^2-(base)^2))
  handles.fileLoaded2 = 1;
    guidata(hObject, handles);
else
    h = msgbox('No primary file has been loaded!','Error','error');
end
```

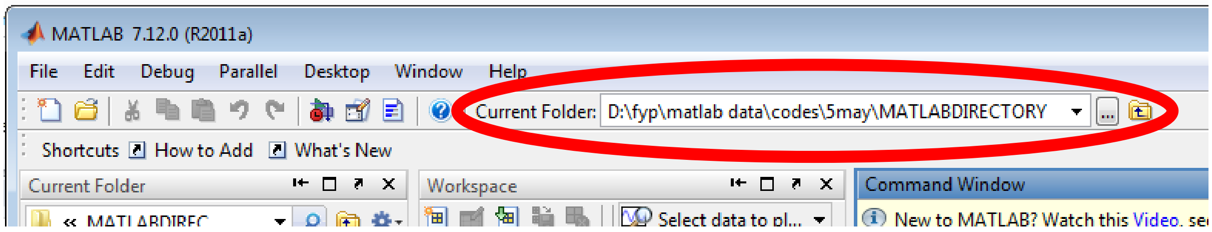
User Guide

For using the virtual table tennis game following software and hardware is required.

- Matlab 2011a or onwards version
- A digital camera having minimum one mega pixel resolution
- A square shaped red color paddle

Steps to follow

1) Install Matlab software. Any version after matlab 2011a can be used.
2) Open the Matlab software and set the directory by adjusting the current folder on matlab toolbar pane.


MATLAB 7.12.0 (R2011a)

Current Folder: D:\fyp\matlab data\codes\5may\MATLABDIRECTORY

Figure 14 - Matlab Directory

3) After setting the directory open the directory and copy the 5 given files in the directory i.e. game.m, coordinates.m, plot.m, paddledetect.m and interface.m.
4) Now configure your matlab. Go to parallel> Manage configuration.

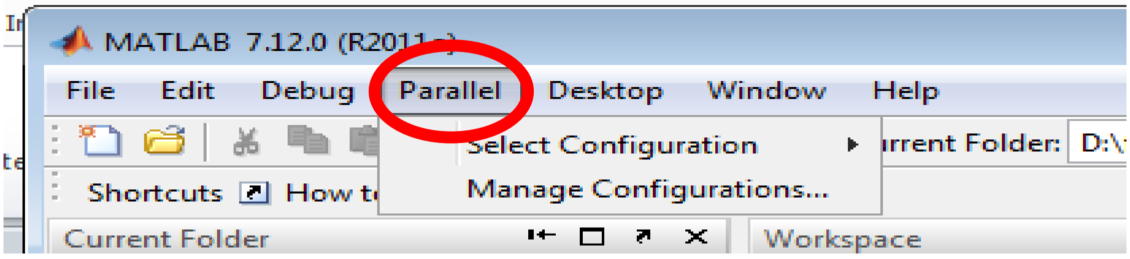
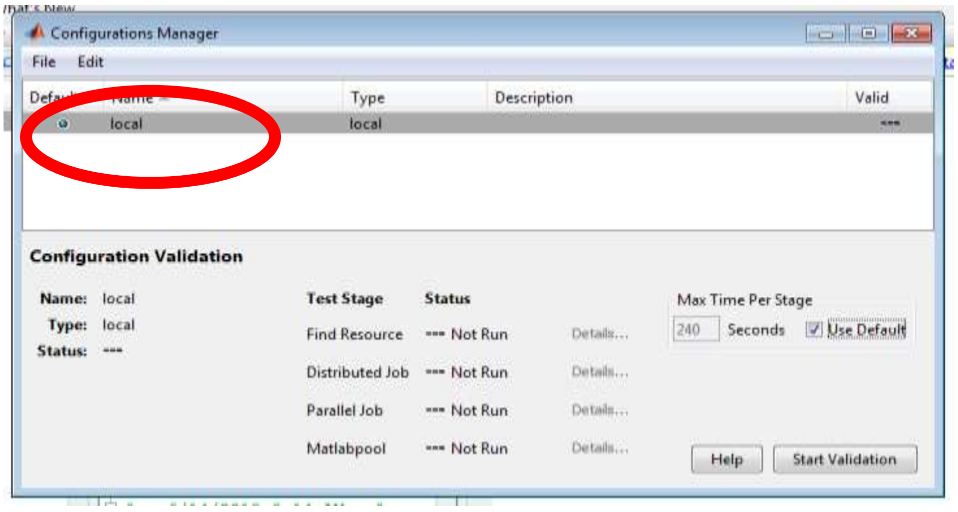


Figure 15- Configuration

5) The configuration manager window will open.


Configurations Manager

File Edit

Default	Name	Type	Description	Valid
●	local	local		---

Configuration Validation

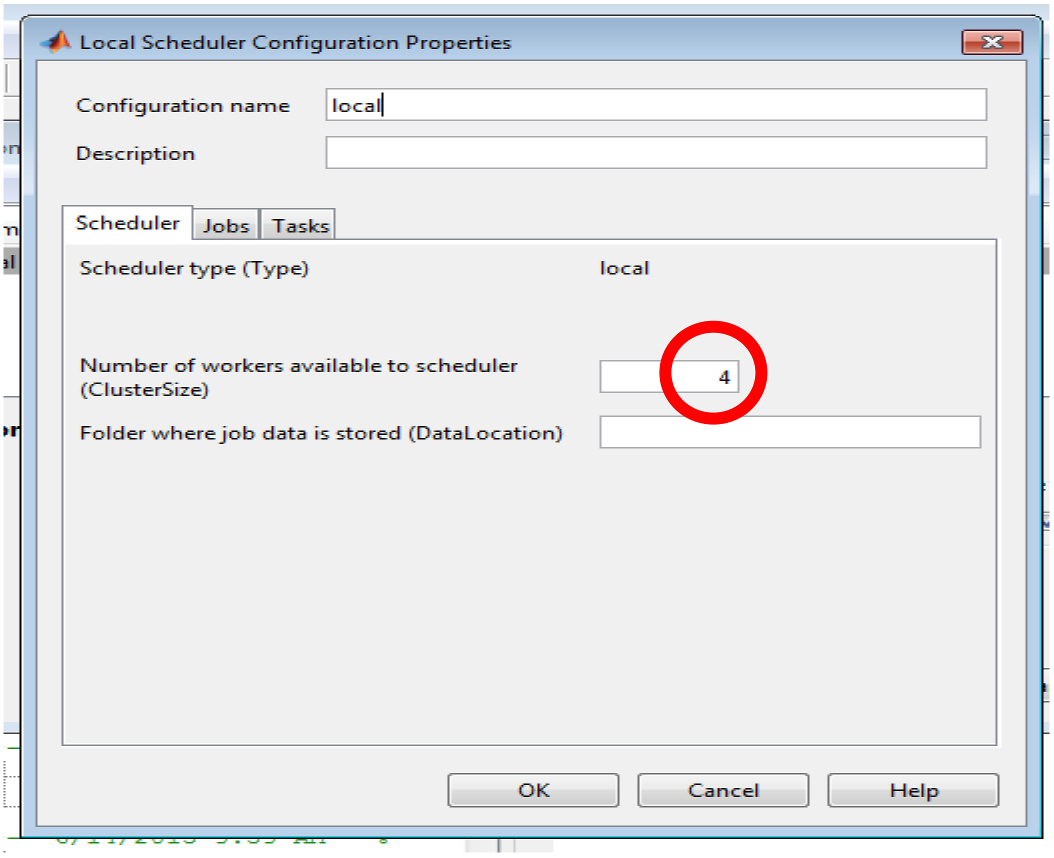
Name: local
Type: local
Status: ---

Test Stage	Status	
Find Resource	--- Not Run	Details...
Distributed Job	--- Not Run	Details...
Parallel Job	--- Not Run	Details...
Matlabpool	--- Not Run	Details...

Max Time Per Stage: 240 Seconds [✓ Use Default]

Figure 16-Configuration manager

6) Double click local. The scheduler tab will open


Local Scheduler Configuration Properties

Configuration name: local

Description:

Scheduler type (Type): local

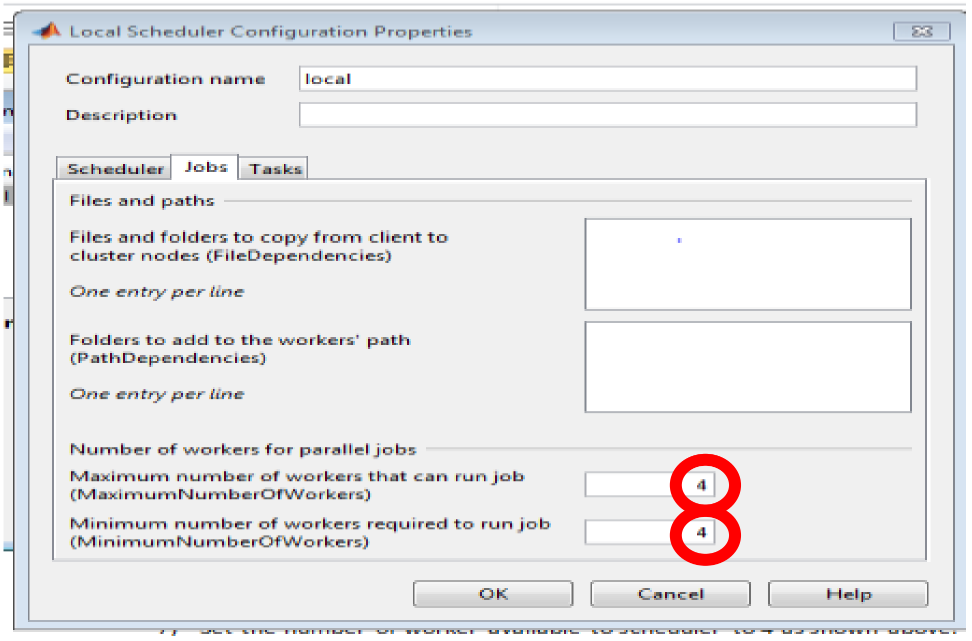
Number of workers available to scheduler (ClusterSize): 4

Folder where job data is stored (DataLocation):

Figure 17-Configuration > scheduler

7) Set the number of worker available to scheduler to 4 as shown above.

8) Go to the next tab that is Jobs. And set maximum number of worker that can run a job to 4 and maximum number of workers required to run a job to 4.


Local Scheduler Configuration Properties

Configuration name: local
Description:

Scheduler | Jobs | Tasks

Files and paths

Files and folders to copy from client to cluster nodes (FileDependencies)
One entry per line

Folders to add to the workers' path (PathDependencies)
One entry per line

Number of workers for parallel jobs

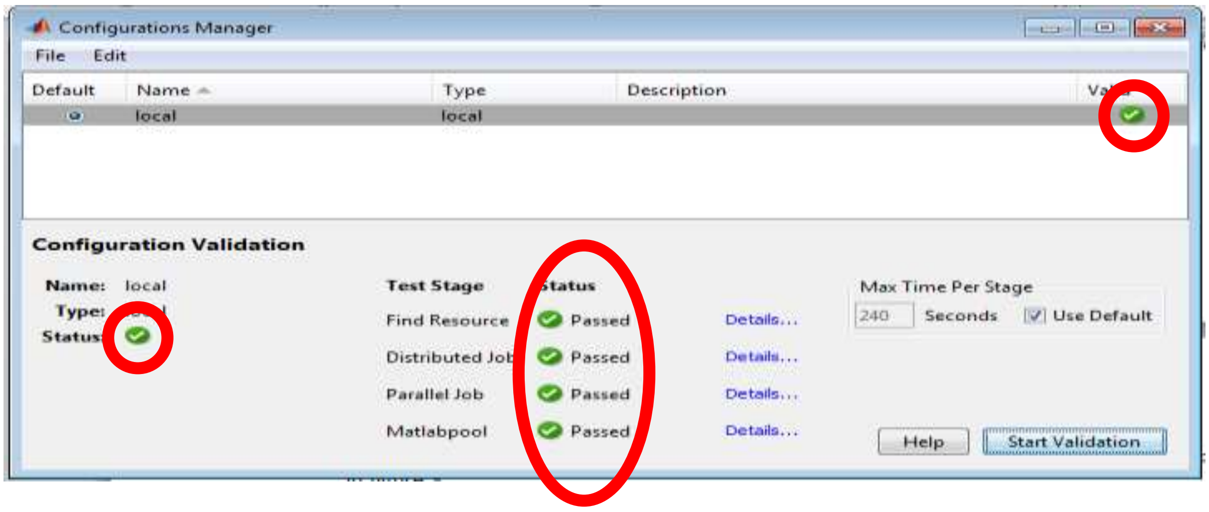
Maximum number of workers that can run job (MaximumNumberOfWorkers): 4

Minimum number of workers required to run job (MinimumNumberOfWorkers): 4

Figure 18-configuration> Jobs

9) Now close the window and click validate now on the configurations manager window shown in figure 3.

10) Wait till the configuration is validated. When the results are validated the configurations manager window will show a tick marked against the local configuration and the test stages status as shown below.


Configurations Manager

Default	Name	Type	Description	Va
●	local	local		✓

Configuration Validation

Name: local
Type: local
Status: ✓

Test Stage	Status	
Find Resource	✓ Passed	Details...
Distributed Job	✓ Passed	Details...
Parallel Job	✓ Passed	Details...
Matlabpool	✓ Passed	Details...

Max Time Per Stage: 240 Seconds [✓] Use Default

Figure 19-configuration validation

11) Close the window.

12) Now open the directory, open the files named as coordinates.m, interface.m and plot.m.
13) Save the files.
14) Now go to matlab directory pane and open paddledetect.m.
15) Editor window will open. Click the play button on the toolbar pane.

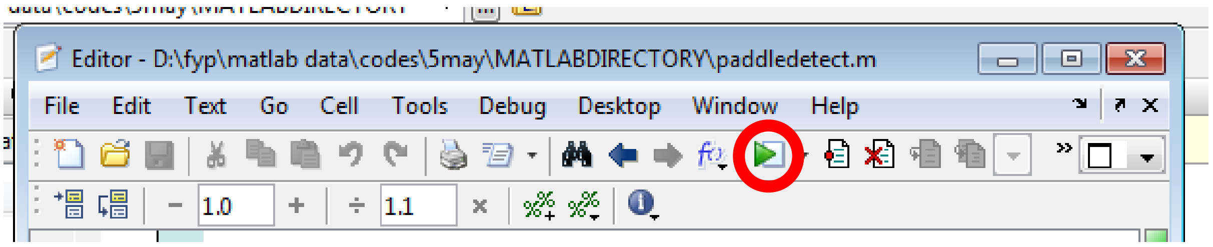

Editor - D:\fyp\matlab data\codes\5may\MATLABDIRECTORY\paddledetect.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1

Figure 20-Editor>Play

16) A window will open giving the real time video of paddle.
17) Adjust the camera position so that the paddle is visible in the video when it is moved up. Down right or left. Close the window.
18) The above 17 steps are required to be done once.
19) To play the game open Matlab directory.
20) Double click game. Matlab will open with an additional window naming editor.
21) Click the play button. The game interface will appear

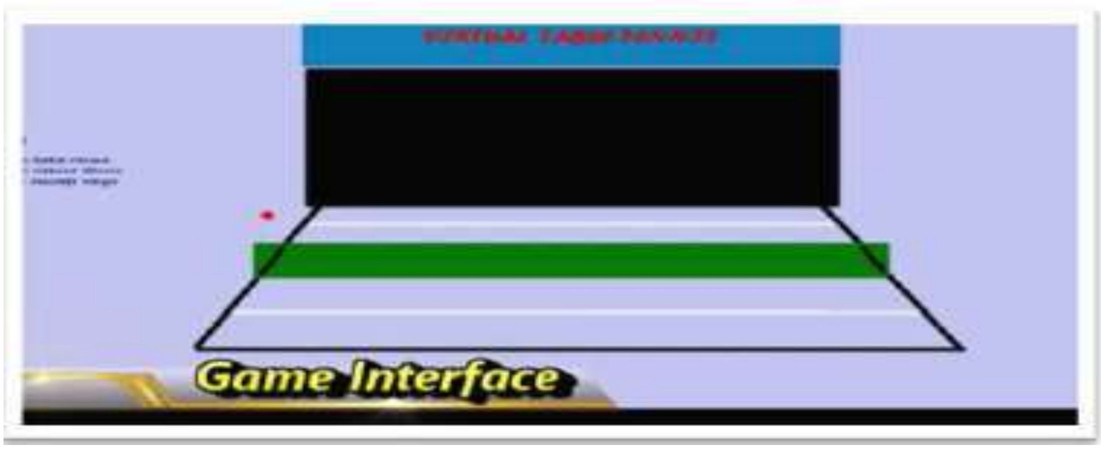


Figure 21-Game interface.

22) First move the paddle in the center of the camera. The game will start.