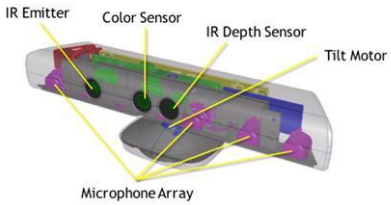
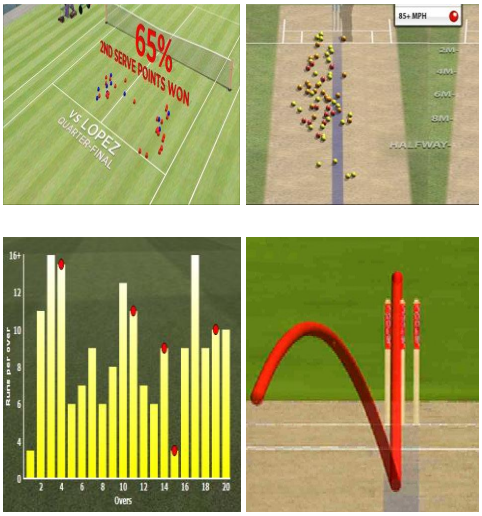
Project Title: Cost Effective Method for Sports Visualization and Statistics generation using Microsoft kinect

Author: Santosh Krishnan Sundararama Sastrigal
MSc Computer Animation and vfx, 2012 – 2013



Output from a Real World Application - Hawkeye



Output from this Project – Kinect Stats Generator

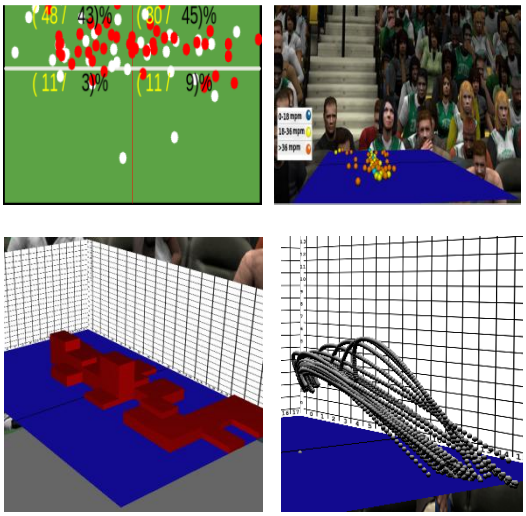


Figure 1: Image reference: cited within this document

Abstract

This report presents the implementation details of a cost effective method to generate statistics suitable for sports broadcasting and player strategy analysis similar to tools such as Hawkeye innovations used in Cricket and Tennis. The intended results of this project was to build a complete software tool using C++, openGL and openCV coupled with Microsoft Kinect to generate various player specific data. Table tennis was the chosen sport taken as reference to generate statistics for, as it is played indoors and confirms to the scope and hardware dependencies of the project.

Various statistics that can be generated out of this implemented tool are 2D Pitch locations of balls, 2D Percentage distribution of pitch locations representing coverage patterns of a player, 3D Height bar graph representing the density of ball pitch locations, 3D ball trajectory for all the rallies, 3D speed pitch map representing the instantaneous speed at the pitch locations and finally revolutions per minute imparted on the ball through the course of a rally. The tool possesses high portability and flexibility with the ability to save the generated data in XML format for later analysis at a different place or time without the need to connect to the kinect hardware. The tool also provides setup options to setup the tracking device providing ease of access to the user in utilizing this tool indoors within their home or coaching centres.

Table of Contents

1. Introduction

1.1 A Brief on Sports Broadcasting and Object Tracking

Broadcasting sporting events to the mass public has reached new heights and the use of technology and innovation to provide the viewers with as much information as possible in a bid to bring them closer to the sporting action has taken prime importance in sports broadcasting around the world. One such information is the statistics generated from a sporting piece of event taking place in a competitive environment.

According to [11], the statistics generated can be used in compelling ways to enhance the appreciation of the athleticism and strategy involved in the sport. This broad problem can be classified into the computer vision and image processing domain with object tracking being the central challenge.

1.2 Objective of this project

With the above briefed inspiration, this programming project was taken up to develop and complete a software tool using a low cost model typically involving low resolution camera setup to generate statistics offline to be rendered to any ordinary user, player or anyone interested in the sport. **The chosen algorithms needed to be flexible as well as extendible to any type of ball detection irrespective of its colour and hence a robust algorithm depending on the shape and size compactness had to be chosen as explained in** Section 4. The resulting statistics can then be used to provide statistical information for the players, the opponents, statisticians and commentators involved in the sport for further analysis of specific strengths, weaknesses and patterns. The end product is intended to be a software tool which would enable the user to setup the system in his/her own place, set various parameters governing the capturing device, process the statistics generated, analyze them instantly and also save them for later retrieval. The tool is also intended to be standalone in terms of dependencies on the capturing device. This means the tool could be used without the capturing device when the user needs to analyse a pre saved statistical data. With regards to rendering, this tool would be able to render the statistics fairly intuitively in a simplified and understandable manner to a general audience.

1. Overview of the report

2.1 Chapter 1 Ground Work

This describes the previous works in object tracking, ball tracking and sports broadcasting, software dependencies and hardware dependencies of the project, the decision on using Kinect as the capturing device, the merits and demerits of it and the decision on employing this tool for the sport of table Tennis.

2.2 Chapter 2 Technical Background/ Image Processing

This describes the technical background of the project which involves the description of various algorithms researched, complete description of the image processing algorithms utilised from opencv in this project and the technicalities of manipulating depth data from the kinect.

2.3 Chapter 3 Implementation/ Statistics Generation and Rendering

This describes the complete architecture of the application in terms of the algorithms developed within this project for statistics generation, the data flow architecture of the project and the pseudo code involved in generating each statistic and its relevance to the real world application. Once data processing is described, this would also describe about data storage in XML format for later retrieval and various rendering strategies employed to display the generated data intuitively.

2.4 Chapter 4 User Interface and Setup instructions

This describes the complete user interface of the system and the ease of use in setting up the system, generating data after a live session and analysing the existing data. Navigational details and other technical details involved in handling multiple windows are also described.

2.5 Chapter 5 Bugs and Efficiency

2.6 Chapter 6 Future works and Conclusions

2.7 Chapter 7 References

Chapter 1 Ground Work

1.1 Previous works in object tracking and sports broadcasting

Object tracking has been central to any image processing and computer vision systems and it can be the most complex of all the tasks in sports visualization due to the loss of information caused by the projection of 3D data on to 2D image, noise in images, object motion, object occlusions, scene illumination and processing requirements. [6]

On personal email contact with the Hawkeye innovations company based in UK, an important paper was obtained from the makers of the famous hawk eye system. According to this [10], the task of sports visualization was subdivided **into field of play extraction, camera calibration determination, ball tracking , impact point determination and further data processing** based on the impact points. Even though this paper was designed to cater to real time processing, the paper laid the foundation to understand the steps involved in pre processing, ball tracking and post processing.

Apart from Hawk eye, the broader problem of object tracking and specifically ball detection in sporting events have been researched by various papers such as [11] which develops a real time tracking system for Tennis broadcasts and [14] which explains about the steps involved in automatic annotation (highlights extraction) of a Tennis match from low resolution video. Also, in [10,11, 12], the emphasis has been to use multiple cameras to compensate for the loss of information from 3D to 2D projection and then reconstructing the data using homograph correspondence and triangulation with the stereo images. To keep the cost and complexity of the project to a minimum and without losing out on data, the algorithms had to be picked keeping in mind the system had to use a single capturing device.

Hence, these papers [4] [9] were analyzed in depth to identify methods to utilise single camera for detection.

3.2 Decision on using Kinect and taking Table Tennis as the reference sport

All the literature discussed above either uses multiple cameras to generate 3D data or the systems using a single camera are only capable of producing 2D data. To find a solution to this problem, there had to be a capturing device which gives the user the 3D information, but also works as a single capturing device. The solution was to use Microsoft Kinect. This ground breaking technology gives the user with all the 3D depth data needed, an RGB buffer and host of other data buffers to manipulate. This was a perfect solution to this project considering the cost effective requirements. The effectiveness of using Kinect as a measuring apparatus has been detailed in [1] which discusses about the accuracy of depth data from Kinect.

Figure 2 Illustrating the Xbox 360 kinect and its inside containing the SOC processor [geek.com, 2013]

Kinect's accuracy has been rated higher than that of comparatively similar web cams according to the same paper.

3.2.1 Merits of using Kinect

A monochrome portable CMOS sensor providing depth information having PS 1080 SOC processor as illustrated above in Figure2 running at a rate of 30 fps was seen as a fitting match to the requirements of the project (just under 150 GBP approximately).

3.2.2 Demerits of using Kinect

Since the Kinect sensor resolution is 640*480, the quality of data can greatly vary at various depth ranges and is quite erroneous at some instances. Also, the frame rate of 30 fps will not always be true due to programmatic constrains and hence can run at a lower rate resulting in loss of frames and eventually loss of ball candidate data. But, for the project scope, this data was enough to produce fairly accurate statistics.

3.2.3 Why Table Tennis?

Again, to remind, the type of this project places a restriction on using high resolution cameras. With kinect's 43° vertical by 57° horizontal field of view, it can only see a field of play which comes under its view frustrum. Also, the IR projector lens used in the Kinect means kinect cannot be used outdoors and under reflective conditions as detailed in [1].

Thus, a ball game which comes under these restrictions and also as much interesting and popular for the statisticians and players to be involved in, was Table Tennis which is ideally played indoors on a table of width which can be covered by kinect's view frustrum when setup top down as illustrated below in Figure 3.

3.3 Software Dependencies

IDE: This project uses the recommended Qt programming environment and the Qt widget based libraries.

Programming: C++ is the programming language used.

Image Processing: openCV is the image processing package used to process the buffer from kinect to generate meaningful data.

Drivers: Libusb and libfreenect are the open source drivers from openKinect community allowing us to connect the Microsoft product onto the linux systems. This also provides APIs to capture data from the kinect buffers using call backs and other utility functions such as

- *freenect_open_device,start_video and stop_video that opens and returns the kinect number, plays and stops the video respectively*
- *freenect_set_depth_callback/freenect_set_video_callback to call back the grabDepth and grabVideo modules of the program*
- *depthCallback(freenect_device *_dev, void *_depth,uint32_t _timestamp=0) and videoCallback modules contains the grabbed depth and video respectively*

Figure 3 illustrating kinect setup indoors

4 Chapter 2 Technical Background (Image processing Stage)

The project can be classified into three major steps and the following Figure 4 illustrates the chronological order of operations. This figure 4 will be used in this report at various places to indicate the current stage being explained. The first stage in the project is image processing to detect balls in motion.

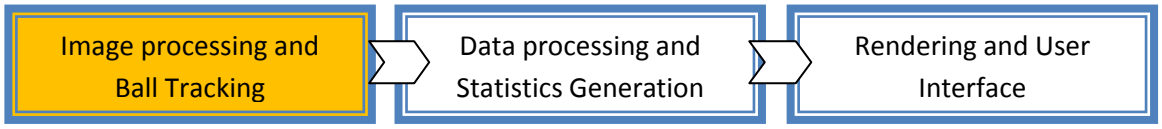


Figure 4 illustrating the current stage in the ball tracking process

4.1 Video and Depth grab from Kinect

Before going into the details of ball tracking, it is relevant to explain the method of retrieving the input video data on top of which the ball detection algorithms have to be applied.

Video Callback: According to the Qt and kinect architecture design explained here [http://jonmacey.blogspot.co.uk/2010/12/openkinect-and-qt-reference-design.html]; the video buffer is just copied onto the class member attribute once we lock the data with a Mutex. Mutex is utilized to be able to grab the data as soon as it is available without affecting the rate at which Libfreenect extracts the data from kinect.

Depth Callback: The depth call back like the video call back gives a void pointer to be casted to the original format of **unint16_t** which will contain 640 * 480 bytes of information to be manipulated into a meaningful data. For each value in the depth buffer of 640 * 480, a lookup table is referenced to convert the pixel value into the real world depth information. The look up table is constructed as explained in http://openkinect.org/wiki/Imaging_Information using the formula below:

$$depth = k3 * \tan\left(\frac{i}{k2} + k1\right) - offset \qquad -equation\ 1$$

Where i = current value in the 640*480 buffer

k1 = 1.1863, k2 = 2842.5, k3 = 0.1236, offset = 0.037

4.1.2 Depth Image Sliced

Once the depth data in Meters is looked up as explained above, the frame is sliced to extract the information within a window of lower threshold and higher threshold values as specified by the user with respect to the location of the kinect.

For example, for the top down setup as illustrated in Figure 3, the table is at a distance of approximately 0.75 meters measured from Kinect's location. This gives us a space to extract all the

information needed in the region between 0.02 and 1 meter taking into accounts the inaccuracies of kinect. This will result in a binary image as illustrated in the Figure 5 below.

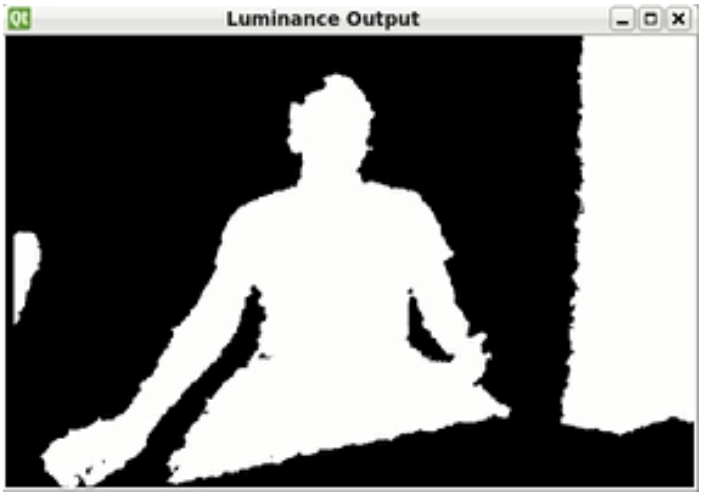


Figure 5 illustrates the depth image within the specified bounds

This greatly enhances the initial speed of the algorithm as the array of values from Kinect has been now converted into a binary representation where everything within a specified range is 255 and everything out of this volume of space is 0.

Once this meaningful frame is available, the real process of tracking the moving ball can be done on these frames continuously. The algorithms involved in the centrally important part of this project are explained in the following chapters.

High Level Data flow Diagram:

The data flow across various subsystems of the project can be represented as illustrated below in Figure 6.

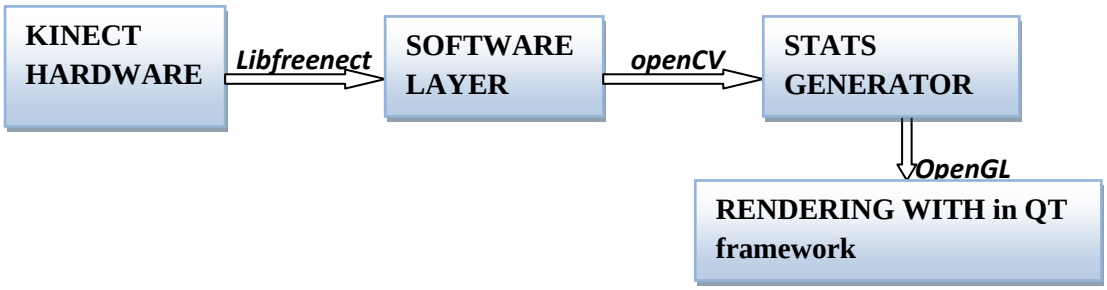


Figure 6 illustrates high level DFD of the project

4.2 Ball Detection Algorithm using openCV

The major work in choosing various algorithms in the process of ball detection is to determine what characteristics of the ball are going to be taken into account in the detection process. One of the major clues of ball detection, being colour, could have been easily used as a first stage filter in the algorithm, but it greatly reduces the flexibility and extendibility of the

algorithm. Hence, the solution had to be a **colour independent, robust algorithm depending on the size and shape** of the ball candidates.

According to [4,9], the basic steps in ball tracking using the video from a single camera can be summarised as

1. ***Identifying the motion regions in an image using optical flow or frame differencing***
2. ***Performing morphological operations on the binary image such as dilate and erode to remove noise***
3. ***Generating ball candidates using size and shape compactness (using convexity defects)***
4. ***Storing the Centre of ball candidates***
5. ***Linear regression for quadratic fitting and interpolation to find the missing points***

The implementation details and choice of algorithm chosen for each step has been detailed below. Various default values used for the modules in the actual project setup are given in the user manual separately.

The entire pseudo code of this algorithm can be visualized below in Figure 7.

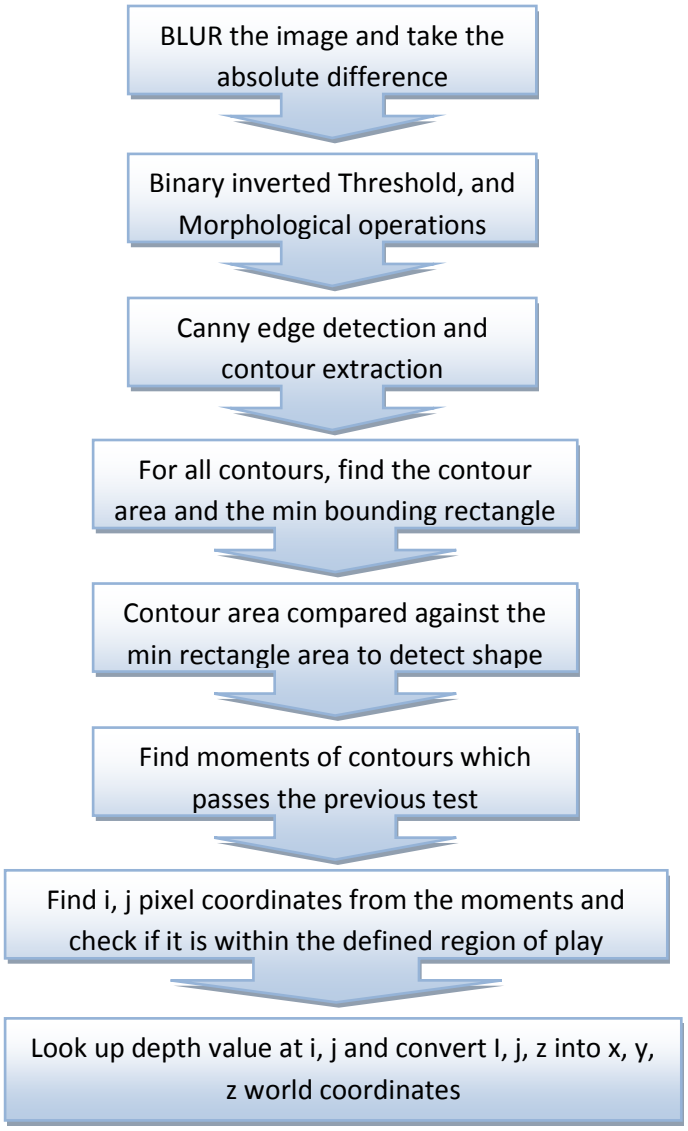


Figure 7 illustrates each step in the ball tracking process

4.3 The complete image processing sequence in detail:

Step 0 is the pre processing stage where the image is blurred using

blur (source frame, destination frame, blur size) module of openCV. This is achieved by openCV using the Kernel window which defines a coefficient matrix that slides across the entire image matrix given by the formula below

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

For the normalized block filter chosen for this project, the kernel K is defined as

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & \dots & 1 \\ . & . & . & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

This step removes noise by replacing a pixel value by the weighted sum of the input pixels and helps feed data cleaner into the steps further.

Comparisons to other available modules: Other noise removal modules such as Median filter, Guassian filter and bilateral filters are available in openCV, but the normal blur() of openCV was chosen after taking into consideration performance factors of these functions illustrated below in figure 8.

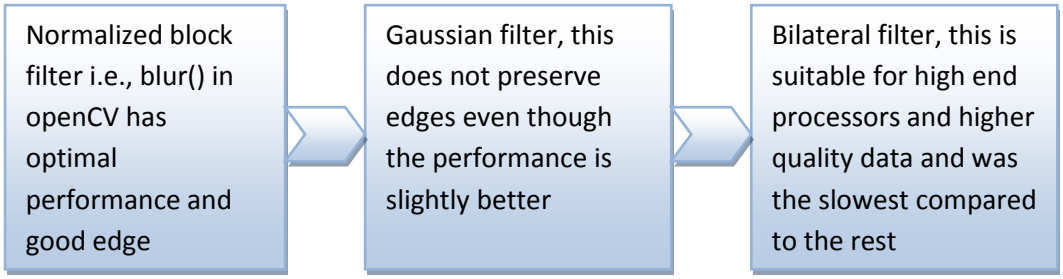


Figure 8 Filters in openCV for noise removal order in terms of optimal performance measures

***S*tep 1** of identifying the motion regions had two solutions, that is, optical flow algorithm and frame differencing. Frame differencing method is an efficient method in extracting motion regions in an image [6] when the background is static and it can be seen from the kinect setup illustrated in Figure 3, the background is going to be static with only moving objects being the ball and occasionally the player's hands when it comes into the field of view of kinect.

Hence, frame differencing is used as the first step in identifying motion regions which is implemented through openCV

absdiff(InputArray src1, InputArray src2, OutputArray dst) module meaning absolute difference between the two frames.

Step 2 has two successive operations namely Thresholding using

threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type) module which takes in a **type** information which is **CV_THRESH_BINARY_INV** which acts on the input buffer according to the formula given below.

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxVal} & \text{otherwise} \end{cases}$$

This means, each pixel value in the image is replaced with a value of 0 if it is greater than the threshold value; else it is replaced with the given MaxVal value. Step 1 of the process results in a difference frame where everything in motion on the screen is set to 255 and all the other pixels in the image are **zeroed** out by the absolute difference method.

This is followed by morphological operations of dilate and erode using **dilate ()** and **erode ()** that uses the given elliptical structuring element as input. The structuring element in this case is an ellipse of size provided by the user according to the size of the object to be detected. This structuring element acts as a sliding kernel to isolate individual elements and remove noise.

Step 3 in this process is to identify ball shapes. This step is the most important step and error at this stage can cause our entire system to process erroneous data. Canny edge detection algorithm called upon by

canny(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool **L2gradient**=false) gives us the detected edges of an object which feeds into

findContours(InputOutputArray **image**, OutputArrayOfArrays **contours**, int **mode**, int **method**, Point **offset**=Point()) to find the containing boundary of the object as illustrated below in figure 9 below.

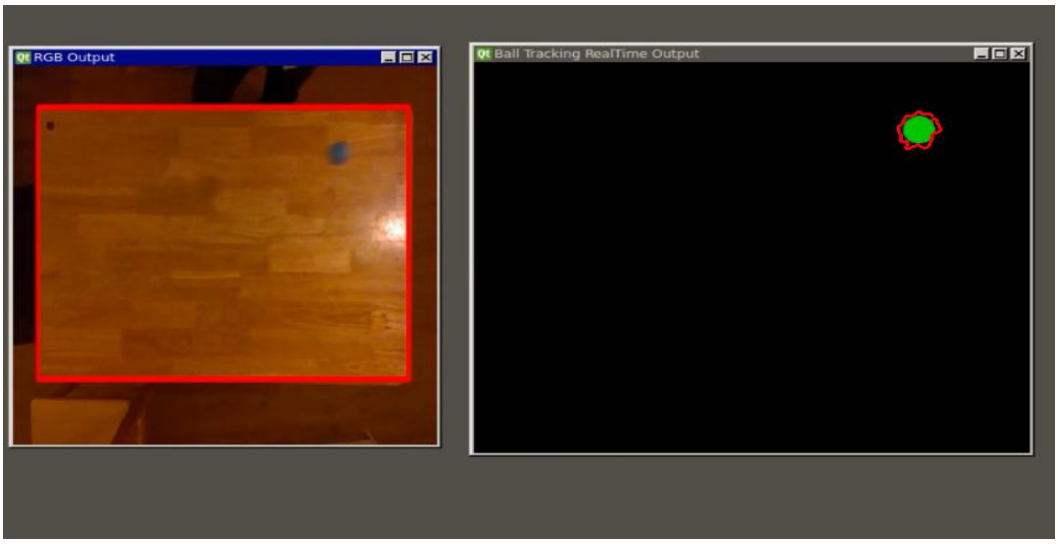


Figure 9 illustrating the contours drawn around a body in motion. The window in the left shows the live video of the ball in motion.

Step 4 is to find the centre of the ball candidate. To do this, first the minimum bounding rectangle of the object is to be found using

minAreaRect (InputArray points).

If the area of this minimum bounding rectangle is greater than a threshold value, the moments of that rectangle is found using

Moments (InputArray array) module.

Spatial moments are utilised in finding the area in pixel coordinates of the given raster image. This is achieved by openCV which implements the following formula.

$$m_{ji} = \sum_{x,y} \left(\text{array}(x,y) \cdot x^j \cdot y^i \right)$$

Where i, j represent a pixel

From the results of moments, the x, y centres of the ball is found using the equation below.

$$\bar{x} = \frac{m_{10}}{m_{00}},\ \bar{y} = \frac{m_{01}}{m_{00}}$$

4.3 2D Ball Coordinates into 3D world coordinates

Once the ball centres have been detected as explained in Section 4.2, the next step is to utilise the power of kinect. Ball candidates are now in 2D pixel coordinates as returned by the moments function in terms of (i, j). Each detected point also has an associated depth value obtained from the kinect's depth look up table which was constructed in the code as explained in equation 1 . **Error! Reference source not found.Error! Reference source not found.**

This can be visualized as illustrated in Figure 10 below.

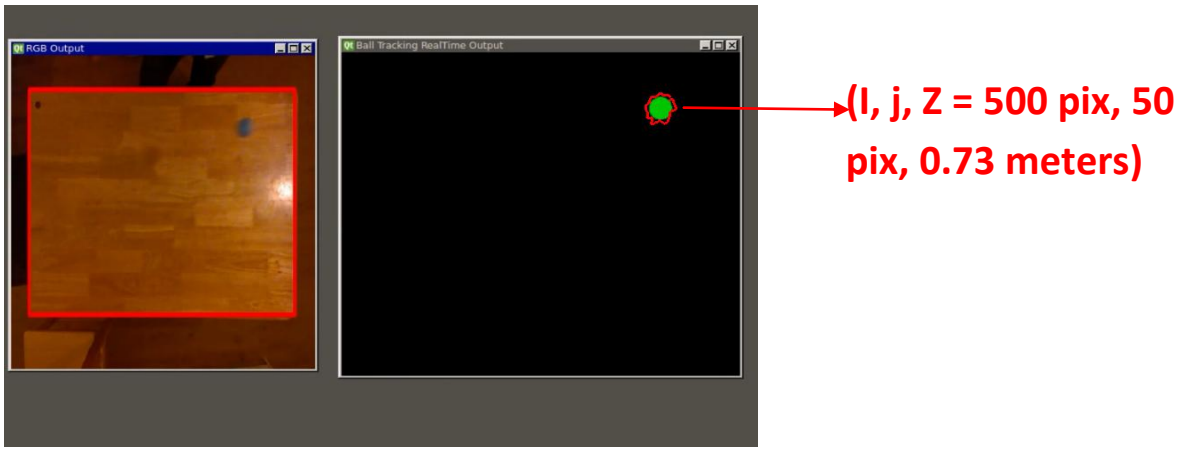


Figure 10 illustrating (I,j,z) representation of the pixel.

So, each detected point can now be represented in (i, j, z). This has to be converted into world coordinates before the data can be processed to result in ball locations defined as (x, y, and z).

In image processing, if one has to know the position of an object relative to the camera, then the camera calibration parameters are to be known. Camera calibration parameters represent the various intrinsic attributes such as the field of view and focal length whereas the extrinsic parameters are represented by the actual location of the camera in 3D space. Fortunately, since the kinect is a standard piece of hardware, various people have calculated the values accurately to map 2D pixel coordinates into 3D world coordinates. One of the accurate formulae found for this conversion is given below.

$$World\ X = \left(\frac{i}{640} - 0.5\right) * Z_{i,j} * 1.111467f$$

$$World\ Y = \left(0.5 - \frac{j}{480}\right) * Z_{i,j} * 0.833599f \qquad \text{- equation 2}$$

Where,

Z_{ij} represents the depth value at (i, j)

640, is the width of the image buffer and

480, the height

These constant values have been derived from http://cvcinema.blogs.upv.es/2011/05/09/the-detection-room/.

5. Chapter 3 Implementation Details (Statistics Generation stage)

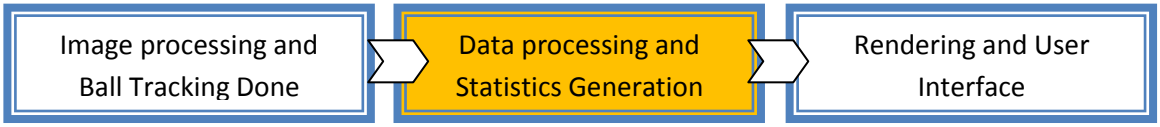


Figure 11 illustrating the current stage of the process

5.1 Player Specific Statistics generation

For the statistics to be relevant the resulting data sets have to be associated with the specific player concerned and this involved writing logic to store the tracked points generated out of the image processing step into player specific instances of the class.

The entire project is divided in terms of programming as follows. Only the major classes have been briefed here.

- ***BallTrackingUtility*** class to track the balls as explained in Section 4.
- ***BallPointsProcessing*** to process the data pushed in from BallTrackingUtility to classify tracked points into player specific instances
- ***TwoDStatsGeneration*** to generate all the 2D statistics of the project and similarly ***ThreeDStatsGeneration*** generates the 3D statistics of the project.
- ***QuadCurveFitUtility*** is a utility class for interpolation implementation through linear regression used by the ***ThreeDStatisticsGeneration*** class.
- ***PlayerData*** class is used to hold all the statistics relevant to the player, produced using the above classes.

The Player class illustrated as part of the class diagram given in the Appendix section contains data associated with a player and creating two instances of this class results in space to store our data sets independently into these two instances.

Statistics classification into player specific instances is subdivided into two parts as

- **RALLY Classification**
- **Impact Point index storage**

5.1.1 Rally Classification is a straight forward process achieved using direction change in X coordinates from the point of view of Kinect. So, if successive points differ in the direction in which they are travelling in terms of pixel coordinates, then it is classified into appropriate players. Figure 12 below illustrates the transformation from kinect's point of view to standardized coordinates used in this project.

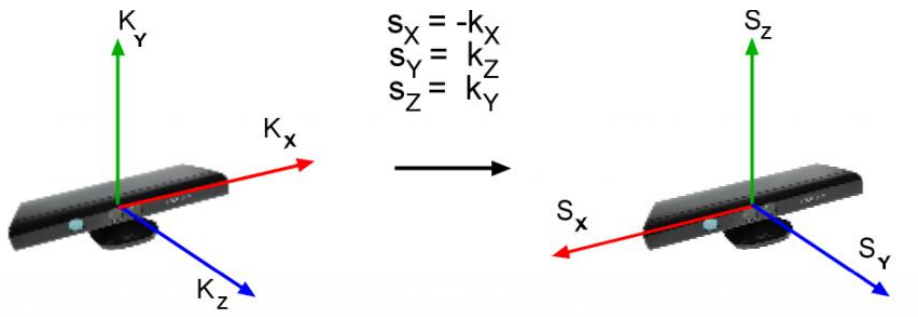


Figure 12 illustrates the coordinate transformations from POV of kinect to the standardized system [http://cvcinema.blogs.upv.es/2011/05/09/the-detection-room/, 2013]

For instance, frame 1 detects a ball at location X = 10 and frame 2 detects the ball at location X = 15, then the ball travels across the screen from left to right assuming the X axis goes from negative to positive axis from the left to right of the screen. This would be classified into a player A's rally bin.

Figure 13 illustrates Player A and Player B classification based on linear direction of travel in X coordinates

Similarly, when there is a change in the direction of ball movement, say from 15 to 10, then it is classified into another player. **When this change in direction is sensed, the rally counter is incremented for the appropriate player every time.** The result of this traversal will be a multidimensional data set representing the tracked points in rally specific rows.

This is a fairly straightforward process when the detection is continuous without erroneous data.

Problems faced with this model: The basic problem faced in this rally classification step was the duplication of points at successively the same spot but with change in decimal units when converted to world coordinates. This meant the ball was tracked at location X = 10.1 and instantly again at X = 9.9 when the actual direction of the ball was from 10 to 11. This caused the classification of rallies into different player bins when the track actually belonged to a single player. This is illustrated in the figure 14 below.

This was avoided by eliminating fairly closely detected points using a fuzzy value.

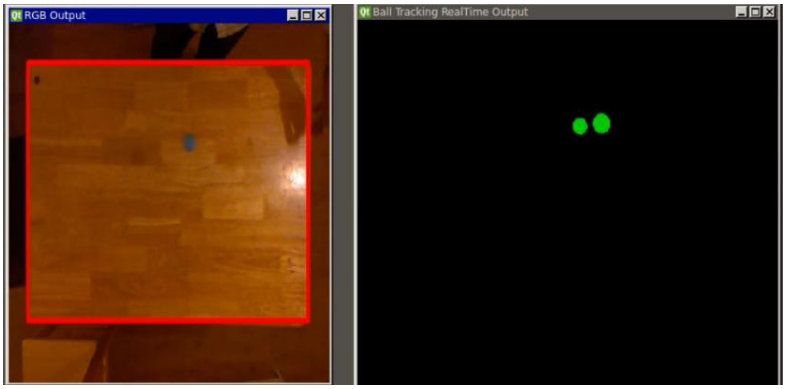


Figure 14 illustrating an instance where the ball candidates are fairly close to each other in pixel coordinates

5.1.2 Impact Point Index Storage

As seen above, in order to classify all the tracked points into player specific instances, all the data points are traversed. **So it would be only efficient if we can pick up more player specific data within this initial traversal loop apart from rally classification**. One of the basic statistics generated out of this tool is the ball pitch locations on the table Tennis table for every rally. So, there needs to be a way to maintain a data structure to denote the data points which represents **only the impact points** of a rally out of all the tracked points as illustrated in the figure 15 below.

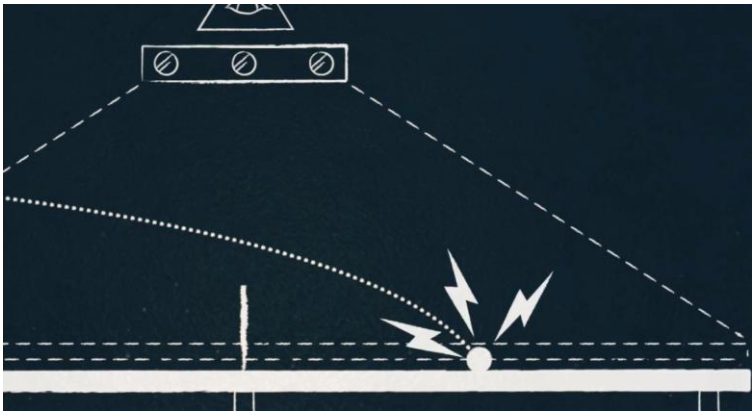
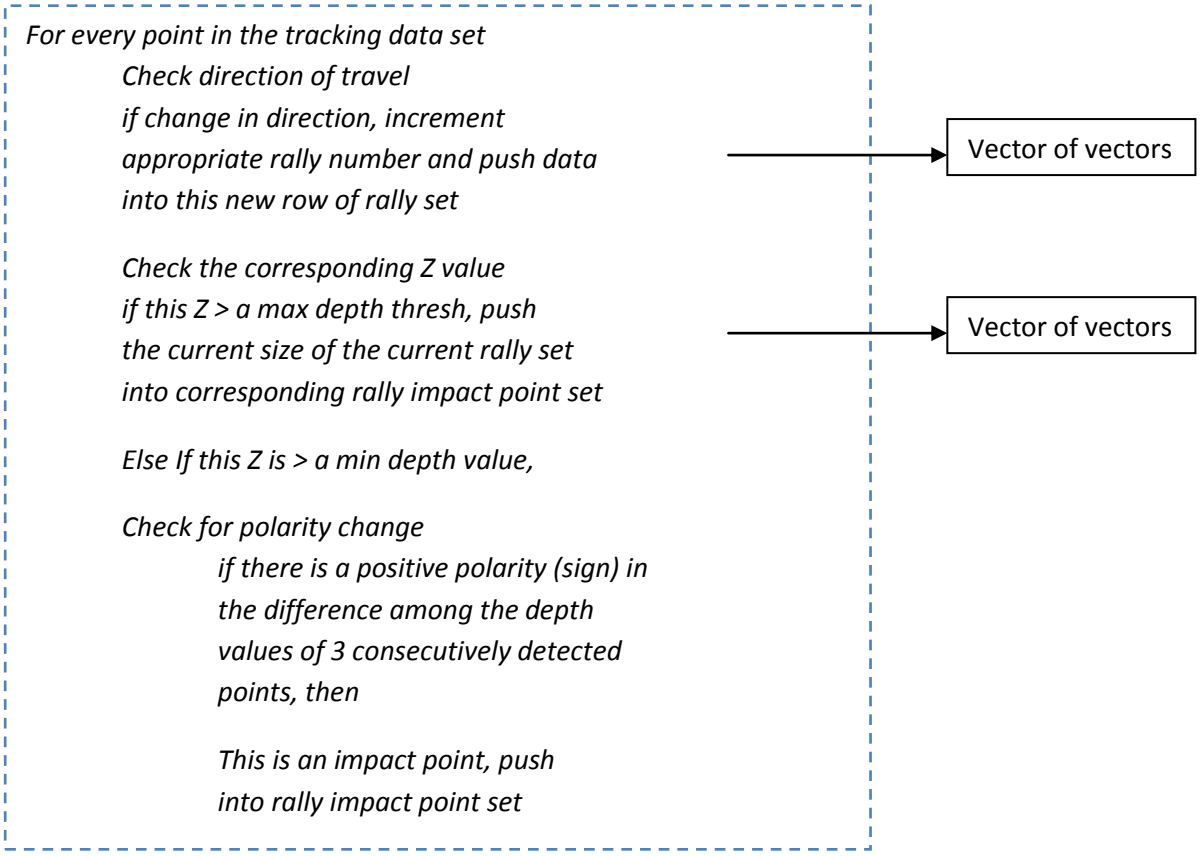


Figure 15 representing the impact point of a ball on the table Tennis board.
[http://blogs.claritycon.com/blog/2012/12/rally-keeper-kinect-based-ping-pong-visualizer/ , 2013]

To achieve this, a 2 dimensional data structure is maintained where each data value will correspond to the index into the master rally data structure where the impact point coordinates are located.

The pseudo code of both rally classification and impact point classification algorithm is given below:



Pseudo code 1 illustrating the construction of rally specific multi dimensional data structure

Problems faced with this model: As explained in the pseudo code above, depth detection is not straight forward. Since the project uses a lower resolution camera, there were additional overheads to make sure the depth points are identified at least approximately when the actual impact point frame has been missed by the kinect.

To avoid this, multiple level checks are carried out to determine an approximate depth point.

First, the Z value of the point is compared against a maximum depth threshold value given by the user. In this case, 0.74 meters as the table is at a distance of 0.75 from the kinect. Hence, Z values greater than this value can be straight away classified into impact points and their index can be stored in another vector of vectors. In case the depth value is less than the max thresh, then the algorithm makes sure to check if it is at least greater than a min thresh value to avoid taking data which are much lesser than 0.75, say 0.5 being taken into further calculations as this cannot represent depth. Hence, if the Z is at least greater than the min thresh value, then

Polarity of difference between the current Z value and the previous Z value is compared against the polarity of the difference between the current Z and the next Z value. If both are positive, then it can be decided that this mid-point must be an impact point. This is illustrated in the Figure 16 below. This is a part of the implementation carried out in the paper cited at [10] .

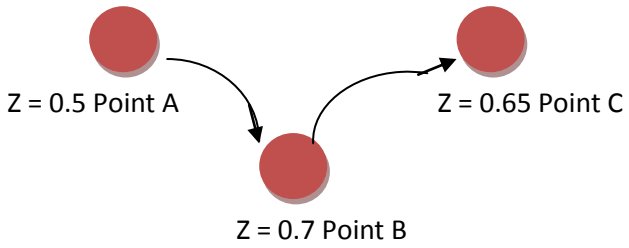


Figure 16 above and 17 below, illustrates the depth values of ball points from kinect's top down view and the polarity comparison used to define them as impact points

Point B - Point A = Positive value

Point B - Point C = Positive value

} Now, this Point B can be classified as an impact point

The programming snippet implemented in achieving this can be better visualized through the following figures.

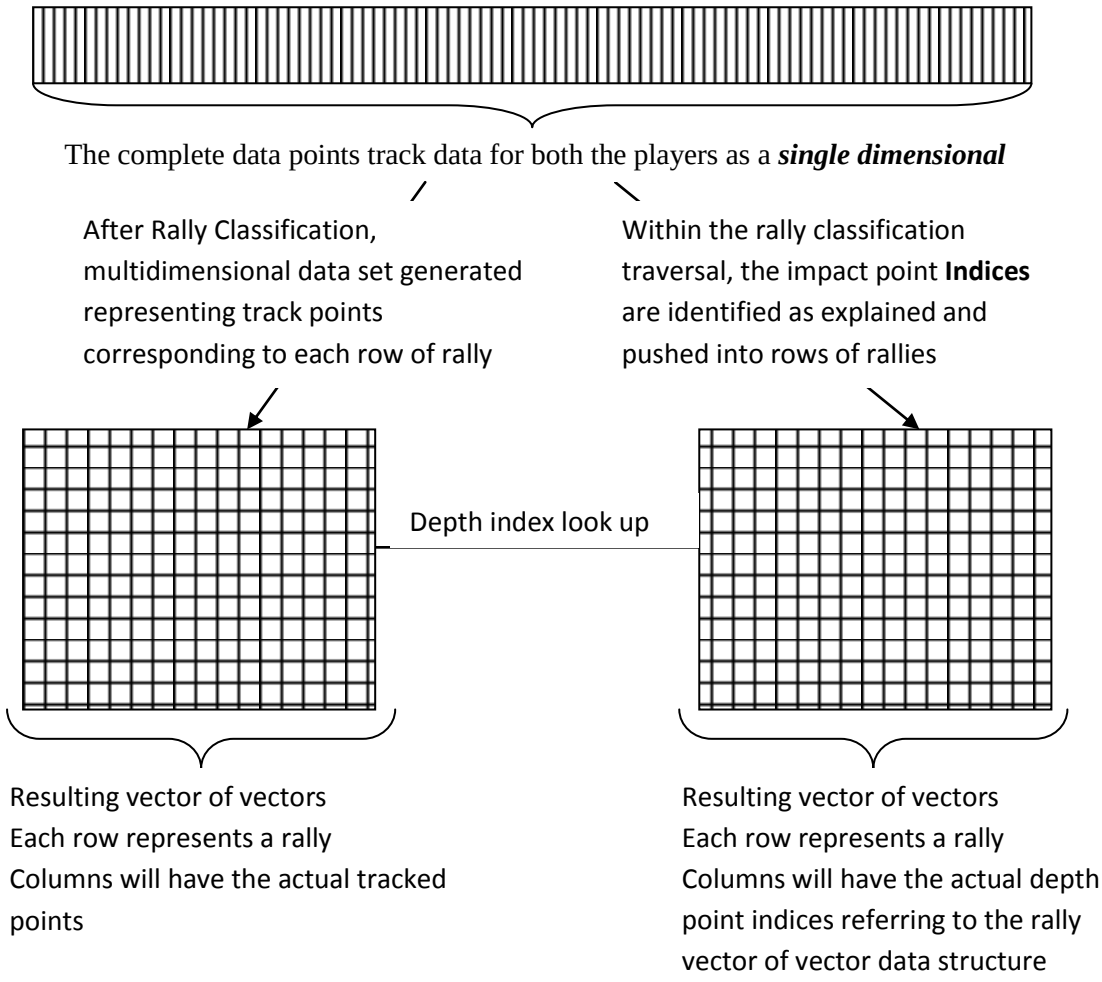


Figure 18 Player Classification Algorithm and associated Data Structure

These two multidimensional vector repositories are the basis of any other stats generated further and it is important to have this data structure in place for easy retrieval of track data at any point. **This algorithm was developed indigenously for this project.**

5.2 STATISTIC 1: PITCH MAP

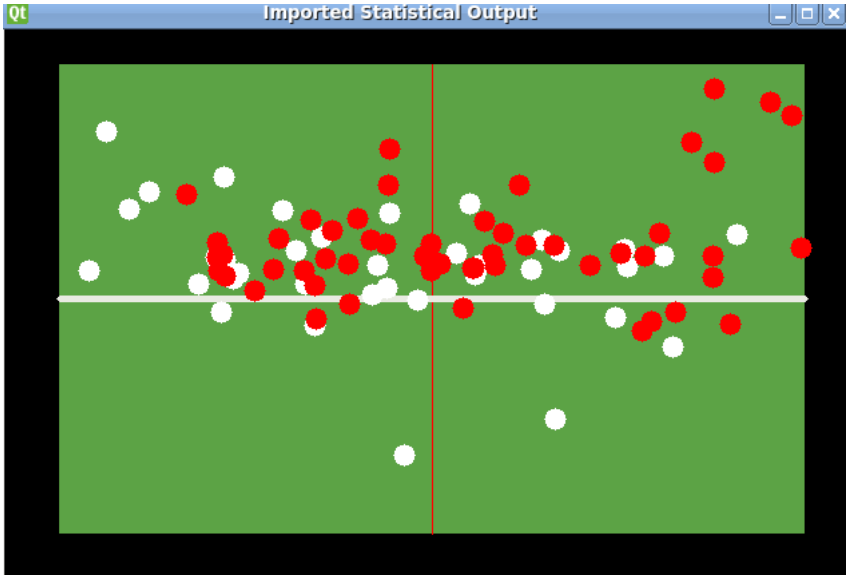


Figure 19 shows the output from the developed application for both players PLAYERA and PLAYERB

5.2.1 Pitch Map Generation

It can be seen from the above Figure 19 that the first level of statistics generated by the developed tool is a 2D pitch map. 2D pitch map represents all the impact points using player specific colours on the defined field of play (Table Tennis). As enough detail has been given in Section 5.1 about how the data is mapped onto specific players and how the depth is referenced using multi dimensional data structure, this specific statistic is a straightforward 2D render of all the depth points in pixel (i, j) coordinates. The player impact points are taken in pixel coordinates and filled in openCV Matrix buffers and rendered using texture pasting. This data is available for both the players together or for specific players as requested by the user.

5.2.2 Applications of pitch map

It is evident from figure 19 that even a basic stat like the pitch map of a player looks visually much more informative for the players and the viewers. Pitch maps are generally used in Tennis as illustrated in Figure 20 for helping referees judge line calls.

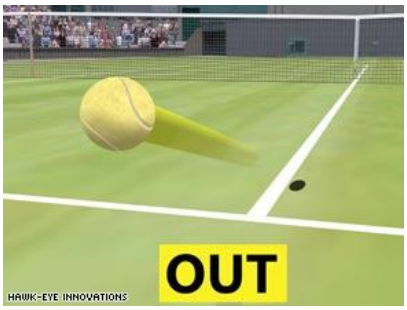


*Figure 20 depicts the use of pitch map in real world applications. [*http://beryl-pieces-tennisfaces.blogspot.co.uk/, 2013]

5.3 STATISTIC 2: PERCENTAGE MAP

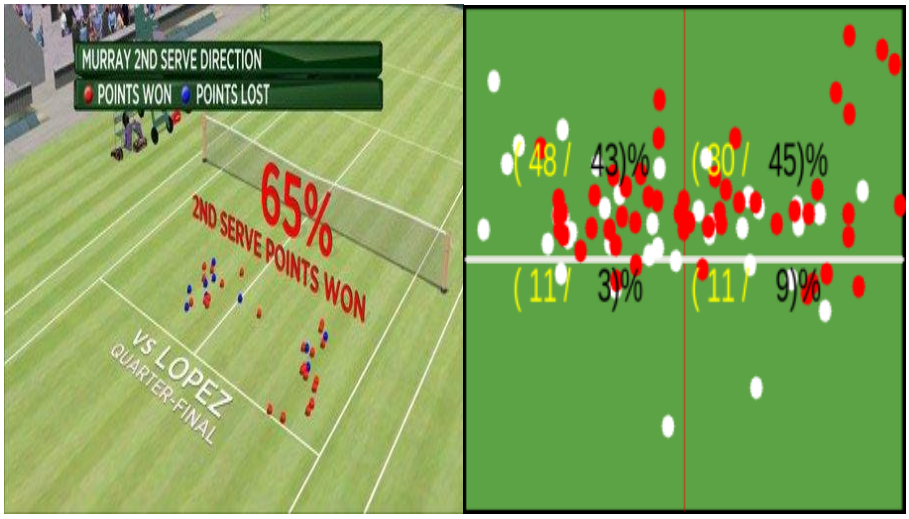


Figure 21 (left) shows the percentage map from a real tennis match [ATP website, 2013] compared against the output (right) from the developed tool representing percentage coverage of PLAYERA and PLAYERB

5.3.1 Percentage Map Generation

Percentage map is basically an extension of pitch map in terms of the raw data taken in pixel coordinates from the depth map multidimensional data structure.

It is visually much more informative in terms of the exact coverage patterns of the player in specific regions of the field of play (Table Tennis field). Since the field of play values are known to the application beforehand (taken as user input interactively), this can be used in sub dividing the impact points on the table into **Quadrant specific** points namely,

Top Left, Bottom Left, Top Right, Bottom Right

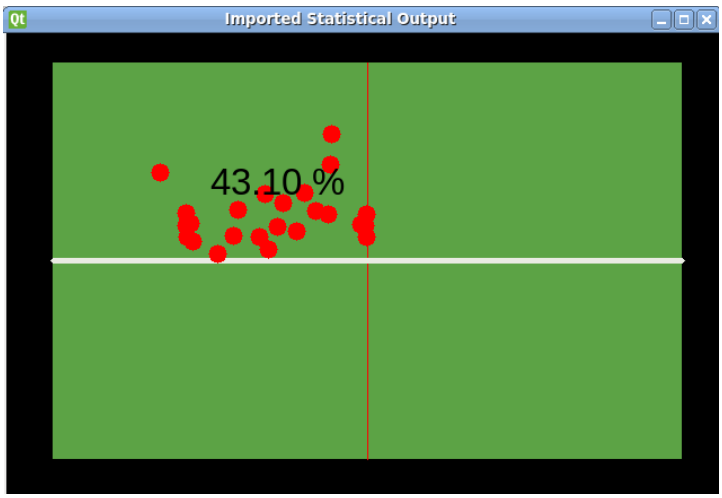


Figure 22 illustrates the Top Left court coverage pattern of PLAYERB

Percentage data is also available for both the players at a time or for specific players as requested by the user for specific quadrants of the field of play.

5.3.2 Applications of Percentage Map

As it can be seen, percentage data is a visually rich stat compared to ordinary pitch map. Percentage data is primarily useful in analysing the strengths and weaknesses, through analysis of how much of the field of play a player is able to cover through his shorts and the variety in playing styles of different players. This will eventually help in understanding their own game and to device specific strategies against specific players. In Cricket, percentage distributions are used in various instances as illustrated in <figures> and in Tennis it is primarily used for detecting court coverage patterns which is similar to the statistics generated as part of this project as shown in figure 21.

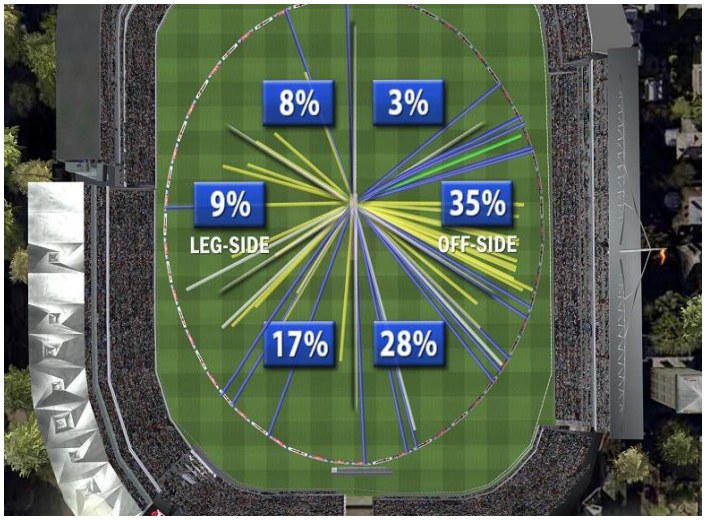


Figure 23 shows the use of percentage distributions in cricket in a live match scenario [thetimes.co.uk, 2013]

5.4 STATISTIC 3: PITCH HEIGHT GRAPH

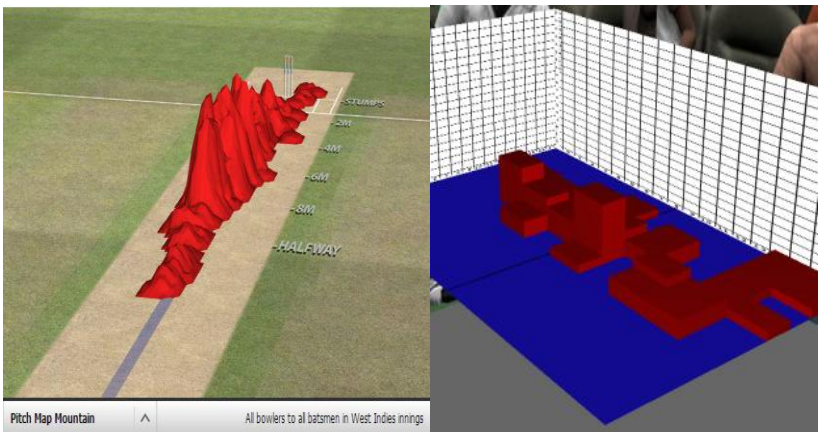


Figure 24 (left) shows the use of height graph of Hawkeye in Cricket (espncricinfo.com, 2013) compared against the output of the height map of PLAYERB (right) generated from the developed tool

5.4.1 Pitch height graph generation

Pitch height graph is one of the advanced 3 dimensional statistics generated out of this tool. Until now, the data was in 2 dimensions and all that the players could analyse was their court distribution patterns of the shots that they have played. In 2D, all the impact points are projected on the same spot if the i,j values are same and there **is no information on the count** of such spots on the table.

Hence, adding the third dimension of height map was the solution arrived to solve this challenge similar to the statistic displayed in Cricket to represent pitch density as illustrated in figure24 (above left).

5.4.1.2 Hash Map technique for height graph

Consider the field of play of a table Tennis court as illustrated in Figure 25 below.

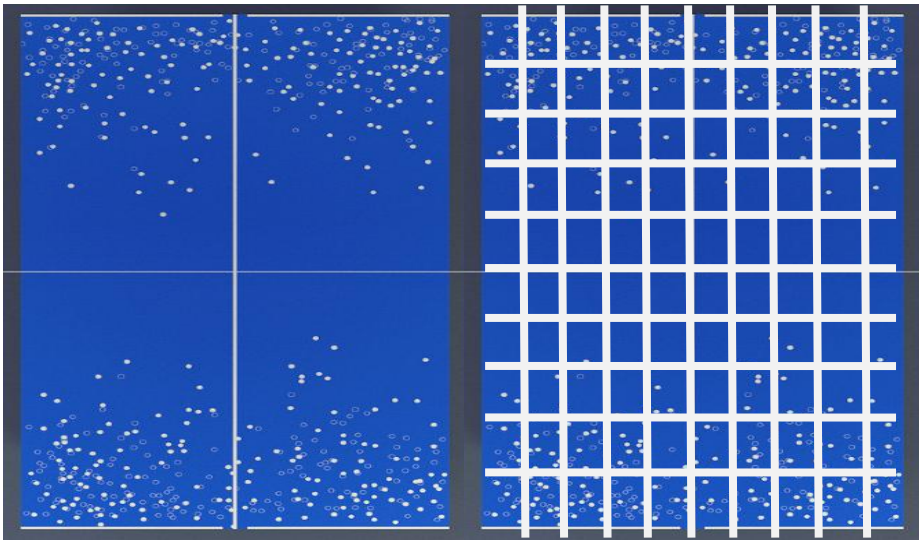


Figure 25 (left) shows a typical table tennis field, which is subdivided for hash mapping (right). [http://designchapel.com/blog/2012/02/09/waldner/, 2013]

The objective of the height map is to represent the number of ball points that have pitched in specific regions on the table Tennis court. **Hash map** was chosen as it is an effective method in classifying data into bins based on regions and can **handle large number of data** together with **O(n) to construct the hash map and O(1) to retrieve** the specific value stored in the bin. Since height map just needs the count of the data in a specific cell, it was a straight forward hashing implementation, the pseudo code of which is explained below.

The table Tennis field is divided into 10 x 10 grids as shown in Figure 25 (right) and then the following hash mapping algorithm is applied.

Note on scale: Since the pitch map is in pixel coordinates, each impact point is scaled down by 10 before applying the hash mapping. These scale units have been intuitively displayed to the end user as part of the statistics rendering window.

Hash Map Construction

```
For every point in the impact point array of a player
        hashX = point.x/gridcellWidth
        hashY = point.y/gridcellHeight
        hashValue = (hashX + (hashY * gridResolution))

        hashMap[hashValue].push_back(point)
```

Pseudo code 2

Hash Map Retrieval

```
hashIncrementIndex = 0

For each row in grid
    For each column in the grid
        number of pitches =hasMap[hashIncrementIndex].size() – 1
        hashIncrementIndex++
```

Pseudo code 3

Rendering through primitive cube: Once the number of pitches is retrieved as illustrated in the above pseudo code 2 and pseudo code 3, this value is fed into the Y component of a primitive cube which is then rendered with its X and Z positions being the gridCell X and Z position which results in player specific height maps, where each cube represents the exact number of ball impacts in the specific grid cell. This is mapped on to the field of play and the scale to read the graph is also rendered alongside.

5.4.2 Applications of Pitch map Height graph

As quoted earlier, the 2D data cannot extend beyond its limitations and the real visualization is appreciable only when it is in 3 dimensions. Through this 3D height graph the players and statisticians would be able to analyse the exact number of ball pitch points on the field of play and compare against each other. Height maps in terms of sports visualisation are generally useful when the field of action is comparably smaller as in this project where the field of play is a table Tennis board. In real world application, currently, height maps are used in Cricket as shown in figure 24(left) and are not to be seen in any other sport. Implementing the concept through hash mapping for a relatively smaller game such as table Tennis is a novel idea that came about through this project and visualising the data on the actual table Tennis board connects to the players and the audience much more than an ordinary mathematical graph.

5.5 STATISTIC 4: TRAJECTORY VIEW

As the project moves forward to more 3D data visualization, an important and much researched statistic is the trajectory of a moving ball in 3 dimensions. This is by far the toughest to generate accurately with the limitations of the project in terms of cost and processing speeds. Below are the details on the problems faced, solutions attempted and the final implemented solution.

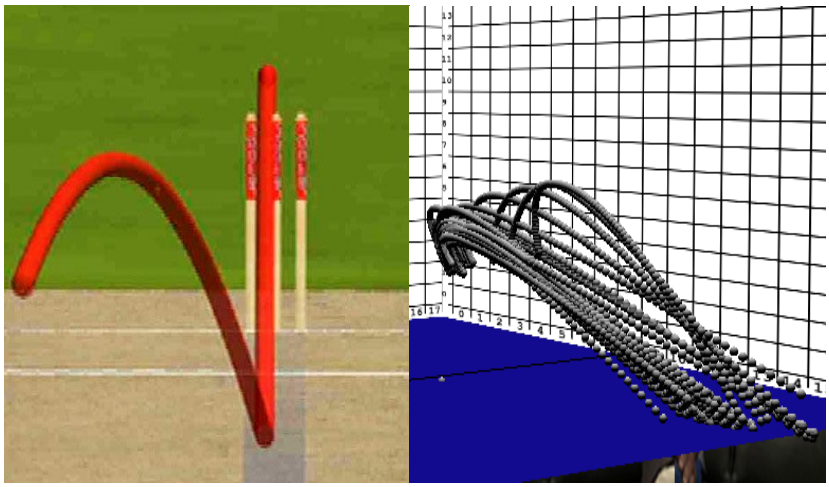


Figure 26 (left) shows Hawkeye trajectory [espncricinfo.com, 2013] compared against PLAYERA trajectory output generated from the developed tool

5.5.1 Trajectory graph generation

Trajectory generation has been researched in detail in papers [10, 11, and 13]. All these have one common thing in generating trajectory in 3D, which is the concept of 3D reconstruction through stereo images from multiple cameras placed around the field of play as illustrated in the Figure 27. Utilizing kinect for depth data was a good idea in terms of the cost. In terms of accuracy, kinect lacks a fair amount when it comes to attempting to create trajectory data through interpolation due to floating point errors propagating throughout the system.

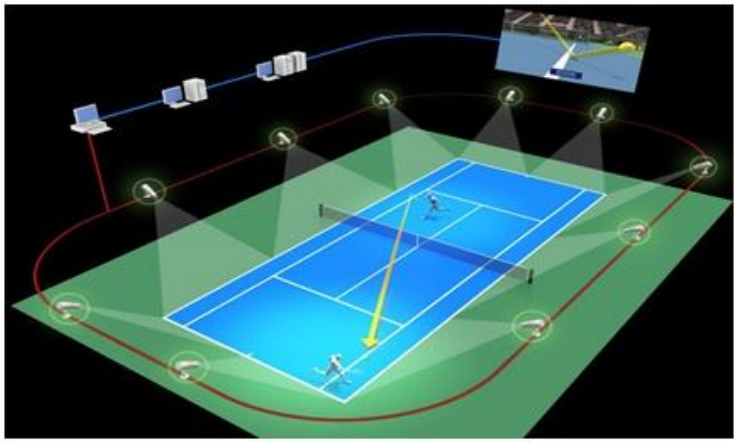


Figure 27 depicts a typical setup of Hawkeye cameras around the field of play [ATP website, 2013]

Creating trajectories through interpolation places **a mandatory condition on the resolution of the cameras used to capture data**. Currently, 350 fps camera is used in cricket and t**here are various instances such as [3] of Hawk-eye being questioned about its trajectory implementation and the company has been trying to update its solution by using more hardware improvements rather than software improvements as discussed** in the video available here (from 7:41 to 7:51), http://ibnlive.in.com/videos/400955/india-favourites-but-need-to-be-wary-of-england-pacers.html.

This is because the software solutions can only act on the data provided by the hardware and if the hardware is powerful enough to generate good amount of data points, then the errors reduce proportionally.

With that said about the existing problems in real world solutions, problems in the trajectory interpolation in the developed application are discussed further.

Initially, trajectory interpolation was carried out in individual coordinate axis with polynomial representations where Y is represented in terms of X and curve fitting is done to obtain the path of a moving object using parabolic equations. The following equation represents a parabola where the coefficients a, b and c are to be found out by minimizing the error function S.

$$S = \sum \left[y_i - ax^2 + bx + c \right]$$

This sum is generally squared and can be represented as follows

$$F(A,B,C) = \sum_{i=1}^{N} (A*X[i]^2+B*X[i]+C-Y[i])^2$$

Minimizing this error involved taking partial derivates of F with respect to the three coefficients A, B and C and the equations can be rearranged resulting in three equations consisting of 3 unknowns.

```
dF/dA = SUM 2*(A*X[i]^2+B*X[i]+C-Y[i])*X[i]^2 = 0,
dF/dB = SUM 2*(A*X[i]^2+B*X[i]+C-Y[i])*X[i] = 0,
dF/dC = SUM 2*(A*X[i]^2+B*X[i]+C-Y[i]) = 0
```

These 3 simultaneous linear equations are solved using Cramer's rule as described within the code.

But, this method is not extendible to 3 dimensions and to have greater control over the generated curves, parametric representation has been used as suggested by Dr. Hammadi.

Parametric representations have the flexibility of representing each coordinate axis in terms of an independent variable't' **which can be applied to all the 3 dimensions (X,Y,Z) independently.**

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d$$

The equation above represents the parametric representation of the Y values in terms of t. This can be extended to all coordinates independently.

Various approaches to parametric curves were attempted and Hermite spline curves were chosen as one such attempt which is a cuibic curve representation interpolating through all the given data points and maintaining C2 continuity at the points. But this attempt ended in an erroneous curve as seen in figure 28 below.

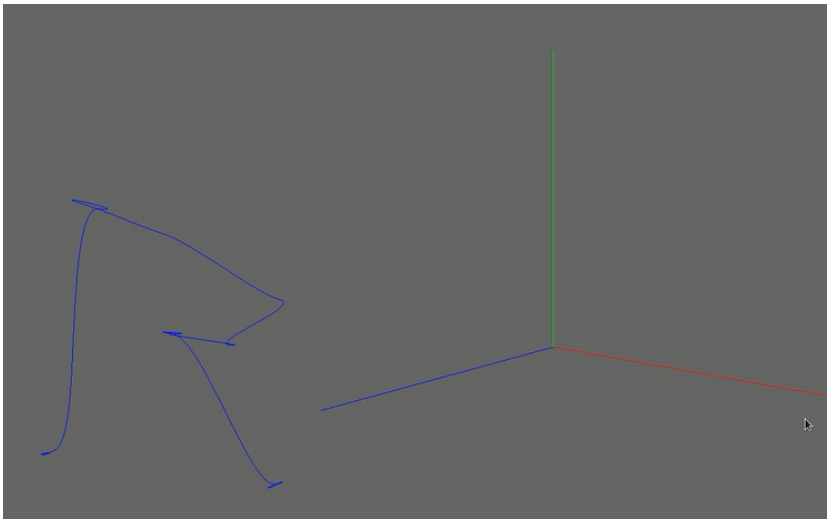


Figure 28, illustrating the hermite cubic splie curve generated as part of initial trials for trajectory generation

On further analysis it was found that the hermite curves were interpolating between the interpolating points independently.

After many days of analysis and trials, finally, a decision was arrived considering the time limits of the project.

Solution:

Quadratic linear regression in terms of parametric curves was implemented by passing in 't' from **0 to 1** for each coordinate axis for interpolating between the data points. This also stores the coefficients to be used in speed calculations later.

Disadvantage: Trajectories are not as accurate as a cubic curve, but was good enough for the scope of this project. If better hardware is used, the trajectories generated would be even more accurate.

This resulted in fairly continuous trajectories as illustrated in figure 26 (right).

Note on scale: Again, the scale to read the map is rendered along with the trajectories and with the current scale; each unit represents 0.02 meters in real world.

5.5.2 Applications of Trajectory View

Trajectory view is the most utilised statistic in real world in terms of Tennis and Cricket. A whole debate on LBW decisions in Cricket rallies around the trajectory calculations. This application has taken hardware limitations and accuracy restrictions into account to produce trajectory to help players visualize the various angles they create during their serves and shots and to compare themselves against the opponent's shot angles.

5.6 STATISTIC 5: SPEED PITCH MAP

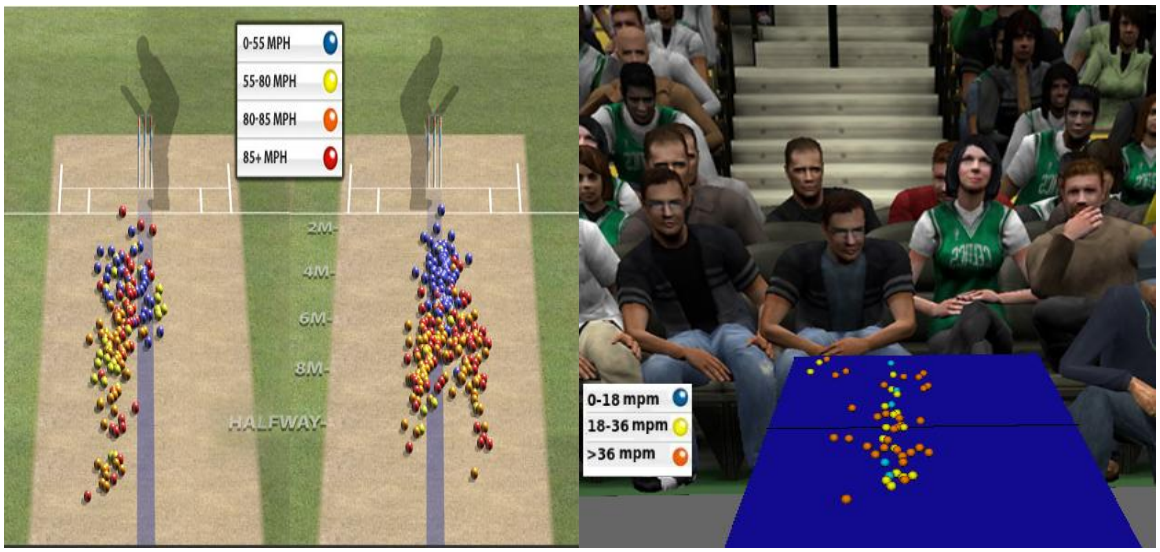


Figure 29, shows the speed pitch map generated by Hawkeye (left) compared against the speed pitch map generated from the tool in terms of meters per minute.

As it can be seen from the graphic displayed above in Figure 29, speed pitch map is famous in sports visualization as it expresses the brute power of the players involved in the sport and pushes them higher every time in order to break their own speed records, both in Cricket in terms of bowling speeds and in Tennis in terms of Serve speed. Speed is central to any ball game and the ability to detect them within this project solution has been a **direct extension of trajectory calculations done through parametric coordinates.**

5.6.1 Speed map generation

During interpolation for generating trajectories for all the rallies for both the players, the coefficients for each part of the trajectory is also stored as explained in the Section 5.5. Speed at a point on the curve is the differential at that point which represents the rate of change of position. **To calculate the differential at a point 't' in the parametric curve is same as finding the tangent at that point as illustrated by the below formulae.**

So when,

$$y = at^2 + bt + c$$

$$\frac{dy}{dt} = 2 * a * t + b$$

Similarly rate of change of x and rate of change of y with respect to t can be found out independently. Once the tangent vector is found in the form of (X,Y,Z), the magnitude of this tangent vector would represent the instantaneous velocity at the point 't' on the parametric curve.

The magnitude of a vector (X, Y, Z) is given by

$$\sqrt[2]{x^2 + y^2 + z^2}$$

By passing in the value of 't' at the depth points, the instantaneous velocity at the pitch location of the ball can found out.

The pseudo code of this process is given below.

Instantaneous velocity at pitch locations

For each interpolated point

Get the x, y and z differentials as explained in <formulae reference>
Calculate the magnitude of this vector as explained in <formulae ref>
Check if current index in the master for loop = depth index in the indices storage multi dimensional vector
If true, push the calculated magnitude into the speed vector repository

Pseudo code 4

5.6.2 Applications of Speed Map

As illustrated in the Figure 29(left), the speed map depicts the physical strength of the players and is one of the most interesting data in the point of view of players and statisticians. It is worth mentioning that such high level data are generated as part of this application using basic formulae and rendered usable for the application users. This application renders the velocities using the scale illustrated in the Figure 30. As usual, this is also rendered as part of the statistics display for the end user to read the map.

Figure 30 is the legend of the speed map rendered in a separate viewport

5.7 STATISTIC 6: RPM PITCH MAP

Revolutions per minute is the last statistic developed as part of this project and is a direct inspiration from the latest Ashes 2013 Cricket broadcast in Sky Sports illustrated below in the Figure 31.

Figure 31 RPM meter in Sky sports introduced in Ashes 2013 [topendsports.com]

This graphic represents the revolutions imparted on the Cricket ball by the spin bowlers. This is an interesting statistic in terms of the sport as until now fast bowlers dominated the statistics generation and RPM is the first of its kind for the spin bowlers.

The developed tool renders the RPM map visually similar to the speed map as the RPM is calculated at the impact points and is described in the following sections.

5.7.1 RPM pitch map generation

Even though the exact implementation details of the RPM meter from Sky sports is not known to the public yet, this project has used a very basic mathematical formula with all the available data to approximate the RPM calculation.

It is known from basic mathematics that,

Circumference of a circle is = $2\pi r$ = distance covered by the ball in one revolution

Since the diameter of the ball is known before hand from user input, the distance travelled by the ball during one full revolution can be calculated using the above equation.

With this information, RPM can be calculated as,

$$\text{RPM} = \frac{\text{velocity in meters per minute (obtained from statistic 5)}}{2\pi r \text{ (radius in meters)}}$$

This would result in revolutions per minute at the impact point location, which is then rendered as usual to the user with the scale represented given by Figure 32.

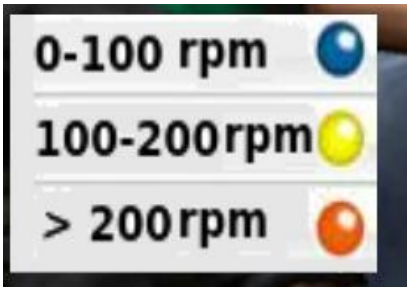


Figure 32 represents the RPM scale

5.7.2 Applications of RPM pitch map

RPM meter has been one of the very recent innovations in sports broadcasting and hence the effects of this statistic on the viewers and the players are not yet measured though there have been some negative reviews from some corners of the audience alleging the over use of technology to impart the user with unnecessary data as cited here http://www.telegraph.co.uk/sport/cricket/international/theashes/10176855/Ashes-2013-First-Test-shows-how-television-technology-can-now-control-the-action.html

With that said, the developed tool has successfully emulated the statistics generation for cost effective sports visualization starting from basic 2D pitch locations to the most recent RPM graphic.

5.8 XML File Parsing for later analysis

The generated statistics from a live video is helpful in readily analysing the game. But, in the case where the players and officials have to get back at **a later point to analyze the data from a game that took place earlier in time, there has to be a way to run the developed tool independently without connecting the capturing device.**

So the tool was equipped with the ability to run independently as a separate module to only analyze a previously archived data instead of live generation.

XML file parser of openCV was used to store the generated data individually for each of the players along with other common attributes such as the field of play.

The architecture of the XML data is illustrated below.

```
<opencv_storage>
        -<PLAYERA>
                <ImpactData> </ImpactData>
                <QuadSpecBL> </QuadSpecBL>
                <QuadSpecTL> </QuadSpecTL>
                <QuadSpecTR> </QuadSpecTR>
                <QuadSpecBR> </QuadSpecBR>
                 <PercentageBL> </PercentageBL>
                <PercentageTL> </PercentageTL>
                <PercentageTR> </PercentageTR>
               <PercentageBR> </PercentageBR>
                <InterpPtsMappedToRallies>  </InterpPtsMappedToRallies>
                <CubeDataVertices> </CubeDataVertices>
                <SpeedData> </SpeedData>
                <PlayerImpactPointsIn3D> </PlayerImpactPointsIn3D>
                <RPMData> </RPMData>
        </PLAYERA>
        -<PLAYERB>
                <Same architecture as PlayerA>
        </PLAYERB>

        <FieldOfPlay> </FieldOfPlay>
        <CourtWidth> </CourtWidth>
        <CourtDepth> </CourtDepth>
        <Width> </Width>
        <Height> </Height>
        <MidTopX> </MidTopX>
        <MidTopY> </MidTopY>
        <MidBottomX> </MidBottomX>
        <MidBottomY> </MidBottomY>
        <MidLeftX> </MidLeftX>
        <MidLeftY> </MidLeftY>
        <MidRightX> </MidRightX>
        <MidRightY> </MidRightY>
</opencv_storage>
```

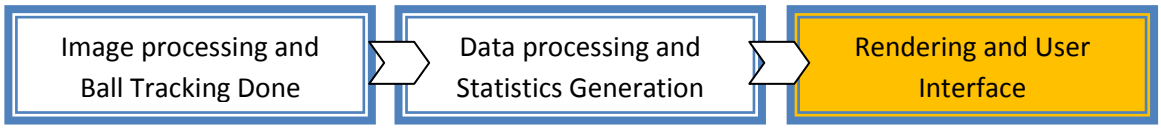
With this XML architecture, file reading and writing through openCV was accomplished using the FileStorage and FileNodeIterator modules to parse the data and write data into the file.

5.8.1 Applications of this XML file within this tool

- **Complete Portability:**
  This tool can now be used completely independently irrespective of the location in terms of analysing the data. For instance, an user who has captured the statistics in a live environment sitting in one part of the world can then Write the file, send it to his distant coach or statistician and the person at the receiving end can use this tool again to import the file received to analyse the data.

- **Dependency Avoidance:**
  XML file saving has given the developed application a new dimension in terms of the ease of use. Statisticians who are not aware of the setup of the kinect or any capturing device and who are not concerned about capturing data live need not depend on those hardware dependencies. All they would have to do is to run the tool, import the existing XML file which was saved before and start analysing the rendered statistics as they would normally do after a live session.

5.9 Rendering Stage



Finally, a brief about the different rendering techniques used in this project is given.

5.9.1 3D rendering

All the 3d statistics generated has been rendered using ngl vao primitives and texture shaders. Also, a legendShader() module takes care of displaying a **separate glViewPort()** for pasting the legends for enabling the users to read the graphs. This separate viewport uses **ngl ortho projection**.

5.9.2 2D rendering

The 2d statistics generated has been rendered through openCV matrix buffers texture mapped onto a Quad.

A small Logic on the GPU texture shader:

Since the project is already heavy in its use in various shader loads and switch calls, a single texture shader is used both for 2D texture pasting on quad and also while 3D statistics is rendered.

The **MVP matrix** is enabled on the GPU whenever a 3D statistic is chosen as illustrated below.

Logic to differentiate 2D tex pasting and 3D perspective projection

```
if(textureFlag == true)
{
        gl_Position = MVP*vec4(inVert, 1.0);
}
else
{
        gl_Position = vec4(inVert, 1.0);

}
```

6.0 Chapter 4 User Interface Design and How To

A complete description on the user interface is provided as a separate document in a user manual.

7.0 Chapter 5 Bugs and Efficiency of the project

Following describes an identified bug and the efficiency details in brief.

- Bug 1: The percentage distribution uses ngl::text for rendering text onto the openGL window on top of openCV texture shader. This works fine for a single session of live stats generation or a single session of Import statistics. But, if the user presses Live session or Import statistics again without restarting the application, the ngl::Text in the percentage distribution statistics does not render. After various sessions of debugging, it was found that the issue was because of multiple shader switches within the tool.

- Start up Issue: The users are instructed to restart the application if the RGB video does not appear on the very first run of the application once kinect is connected to the system. This is system dependent and sometimes kinect starts up with the correct data buffers on the first run itself, but in other cases, an application restart would work fine.

This is a minor bug and does not affect any other statistic.

- Efficiency: The tool is completely robust in terms of boundary conditions and memory leaks. It is completely **non – crash prone and error free** even after implementing the UI design to allow the user to switch between **LIVE, IMPORT and SETUP at anytime within a session without having to restart the application where objects are destroyed and re-created.** One deterrent in the efficiency of the application may be the use of multiple shader loads for each subwindow displayed in the project. Hence, the UI architecture creates only those windows that are necessary for the current session of either Live or Import or Setup.

- **Future Works:**
  One of the major future works in this project is to utilize the Microphone array of the Kinect. Kinect has intelligent pairs of eyes, the depth and the IR projector. But, its ears, being the microphone array can be opened up to create interesting add-ons to this project in terms of Voice recognition. The players could instruct the computer to display the required statistic instead of using their sweaty palms to analyse the data. Kinect can offer such interesting features and the use of this device on sports visualisation has been a complete learning experience in terms of programming, API handling and creating multimedia tools that relates to a wider audience.
  Also, use of multiple cameras for 3D reconstruction through stereo triangulation would enhance the accuracy of the 3D trajectory greatly and is the most worthy future works of all. For performance improvements, the application could be multithreaded with the use of multiple cameras with each camera handled in separate threads and the data received from the camera being processed together. This would involve deeper concepts of image processing and homograph correspondence between image frames and will surely be an interesting and challenging solution to implement.

- **Conclusion**
  To conclude, this project has been very satisfying in learning and implementing basic and advanced concepts behind a real world computer graphics application used in the field of sports visualization that relates to the general public easily and yet has the scope of a complete Masters project with its involved mathematics at each step of ball tracking and the image processing algorithms. **Using Kinect to produce such statistics has been a novel approach attempted within the allocated time for this Masters project**. **To realize the involvement of all the basic mathematical operations and high level C++ programming abilities in a consumer relevant field** such as sports broadcasting feels highly rewarding and worthy.

  To be able to compare the developed tool with **multimedia content developers and broadcasters such as Hawk-eye and Sky sports is a great motivation factor** for any technical fan of sports. Through these generated statistics, it is possible to appreciate the level of athleticism and spirit involved in playing a game at the highest possible competitive environment which also acts as a bridge to reach wider audience base and brings more people into the specific sport.

  Overall the project was challenging at the same time most enjoyable to work on as it is a unique project that is completely relevant to computer graphics, image processing and ball tracking algorithms as well as to the wider public by generating statistics that would be analysed in a real world scenario which could potentially change the result of a game, a country winning or losing, a player ending up as a loser or a champion.

[1] Andersen, M.R., Jensen, T., Lisouski, P., Mortensen, A.K., Hansen, M.K., Gregersen, T., Ahrendt, P., 2012. *Kinect Depth Sensor Evaluation for Computer Vision Applications*, In: Technical report ECE-TR-6, 37, Aarhus University, Denmark. Available from: http://eng.au.dk/fileadmin/DJF/ENG/PDF-filer/Tekniske_rapporter/Technical_Report_ECE-TR-6-samlet.pdf [Accessed 10 May 2013].

[2] Ballester, J., Pheatt, C., 2012. *Using the Xbox Kinect sensor for positional data acquisition*. In: American journal of Physics, 81(1), 71, Kansas: Available from: http://ajp.aapt.org/resource/1/ajpias/v81/i1/p71_s1?isAuthorized=no [Accessed 10 May 2013].

[3] Carter, S., 2011. *AJMAL TO TENDULKAR – LBW REVIEW*. London: Available from: http://www.sportskeeda.com/2011/04/09/hawk-eyes-explanation-on-tendulkars-controversial-lbw-in-india-vs-pakistan-semifinal/ [Accessed 10 May 2013].

[4] Chakraborty, B., Meher, S., 2011. *2D Trajectory-based Position Estimation and Tracking of the Ball in a Basketball Video*. In: Second International Conference on Trends in Optics and Photonics, 7 – 9 December 2011, Department of Applied Optics and Photonics, University of Calcutta, India. Available from: http://hdl.handle.net/2080/1569 [Accessed 10 May 2013].

[5] Crock, N., 2011. *Kinect depth vs Actual Distance.* Available from: http://mathnathan.com/2011/02/depthvsdistance/ [Accessed 10 May 2013].

[6] Gupta, K., Kulkarni, A.V., 2008. *Advances in Computer and Information Sciences and Engineering.* Netherlands, Springer, 245 – 250.

[7] Harris, C., Stennett, C., 1990. *RAPID - a video rate object tracker*. In: Andrew Sleigh, editors, *Proceedings of the British Machine Conference*. BMVA Press, September 1990. doi:10.5244/C.4.15. Available from: http://www.bmva.org/bmvc/1990/bmvc-90-015.html [Accessed 10 May 2013].

[8] H.-T. Chen et al., 2008. *Physics-based ball tracking and 3D trajectory reconstruction with applications to shooting location estimation in basketball video*, J. Vis. Commun. (2009), doi:10.1016/j.jvcir.2008.11.008. Available from: http://people.cs.nctu.edu.tw/~huatsung/papers/0901JVCI-Physics-based%20Ball%20Tracking%20and%203D%20Trajectory%20Reconstruction%20with%20Applications%20to%20Shooting%20Location%20Estimation%20in%20Basketball%20Video.pdf [Accessed 10 May 2013].

[9] H.-T. Chen et al., 2008. *A Trajectory-Based Ball Tracking Framework with Visual Enrichment for Broadcast Baseball Videos*. In: Journal of Information Science and Engineering 24, 143-157.

[10] Owens, N., Harris, C., Stennett, C., 2011. *Hawk-Eye Tennis System.* In: International Conference On Visual Information Engineering, December 2003. Available from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1341323&tag=1 [Accessed 10 May 2013].

[11] Pingali, G.S., Jean, Y., Carlbom, I., 1998. *Real time tracking for enhanced tennis broadcasts,* In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 23-25 Jun 1998, Santa Barbara, CA

[12] Pingali, G., Opalach, A., Jean, Y., 2000. *Ball Tracking and virtual replays for Innovative Tennis Broadcasts,* In: ICPR Proceedings of the International Conference on Pattern Recognition – 4, 41 – 52, 2000, USA

[13] Velammal, B.L., Kumar, P., 2010. *An Efficient Ball Detection Framework for Cricket. International Journal of Computer Science Issues, 7(3), No 2*. Available from: http://ijcsi.org/papers/7-3-2-30-35.pdf [Accessed 10 May 2013].

[14] Yan, F., Christmas, W., Kittler, J., *A Tennis Ball Tracking Algorithm for Automatic Annotation of Tennis Match*. In W. F. Clocksin, A. W. Fitzgibbon and P. H. S. Torr, editors, *Proceedings of the British Machine Conference,* pages 67.1-67.10. BMVA Press, September 2005. doi:10.5244/C.19.67. Available From: http://www.bmva.org/bmvc/2005/papers/paper-150.html [Accessed 10 May 2013].

Video References:

[15] AFP, 2012. *FIFA tests new Hawk Eye goal-line technology*. Video. Available from: https://www.youtube.com/watch?v=PosPemlf2LI [Accessed 10 May 2013].

[16] australianopentv, 2012. *Inside Hawkeye*. Video. Available from: https://www.youtube.com/watch?v=XhQyVnwBXBs [Accessed 10 May 2013].

[17] cricketyorkshire, 2011. Cricket *Yorkshire Hawk-Eye Session (Lord's). Video*. Available from: https://www.youtube.com/watch?v=aVVOH8aDG6Y [Accessed 10 May 2013].

[18] espnstar.com, 2012. *Cricket_ Hawk-Eye explained*. Video. Available from: https://www.youtube.com/watch?v=exEHTO-YnuER [Accessed 10 May 2013].

[19] FIFATV, 2012. *Goal-line technology_ Hawk-Eye explained*. Video. Available from: https://www.youtube.com/watch?v=exEHTO-YnuE [Accessed 10 May 2013].

[20] FIFATV, 2012. *Testing the goal-line technology systems*. Video. Available from: https://www.youtube.com/watch?v=u-bSsv3Z-3E [Accessed 10 May 2013].

[21] mysonyprofessional, 2012. *Sony Professional_ Hawk-Eye Goal Line Technology testing*. Video. Available from: https://www.youtube.com/watch?v=9Exsb0REOtI [Accessed 10 May 2013].

[22] SonyEricssonWTATour, 2010. *A Hawkeye view*. Video. Available from: https://www.youtube.com/watch?v=XPgsgwezO5M [Accessed 10 May 2013].

[23] SonyHowTo, 2013. *Hawkeye Technology from Sony*. Video. Available from: https://www.youtube.com/watch?v=KxgRGQlo85w[Accessed 10 May 2013].

[24] virtualeyepresenter, 2010. *Cricket 2010 - Ball Tracking System. Video*. Available from: https://www.youtube.com/watch?v=XAX2mIXxjiU [Accessed 10 May 2013].