VISION BASED POSITION TRACKING FOR VIRTUAL REALITY

COURSE PROJECT
AM5011: VIRTUAL REALITY ENGINEERING

GUIDE : Dr. M MANIVANNAN

SUBMITTED BY :

RIDHI PUPPALA - ME15B133
DASARI ANANYANANDA - ME15B163

A. INTRODUCTION

Virtual reality is an extremely fast growing field due to its wide and varied applications in many areas of science and technology. Most of the mobile Headsets for Virtual Reality devices like Google Cardboard, Samsung gear VR and several other small company products currently lack a good position tracking system while they accommodate a head orientation tracking mechanism. They can track the rotation of the head using IMU sensors in the phone, but determining head position is difficult. This affects the overall immersive experience of these VR devices.

State of art VR Head Mounted Displays like HTC VIVE and Oculus use a set of IR LEDs and multiple high resolution cameras for tracking the position of the user in a constrained physical domain. These products are very expensive and not affordable to everyone. With VR becoming more popular day by day, there is need to make it affordable while not compromising on the overall immersive experience.

B. PROBLEM DEFINITION

Create an efficient and cost-effective position tracking system for smartphone virtual reality, to enhance user experience such that it can work in combination with head orientation tracking and virtual environment generation software. Aim is to develop a compact and modular setup for positional tracking in constrained indoor locations.

C. Literature Survey for Possible Approaches

C.1. VSLAM

Approach:

Visual Simultaneous Localisation and Mapping is a technique that tracks a person's location within an unknown map, while continuously updating the map. Given a series of sensor observations over discrete time steps, SLAM estimates the agent's location and also generates a map of the environment. The SLAM problem uses other estimation and approximation algorithms, such as Particle Filter and Extended Kalman Filter. Modern SLAM systems are based on tracking a set of points through successive camera frames and using them to estimate their 3D position, while simultaneously using the estimated point locations to calculate the camera pose which could have been used. The camera pose thus generated could be used to update a VR environment, changing it based on the users position.

Issues:

VSLAM or Fast SLAM based approaches were widely developed for autonomous robots with dead reckoning and localization capabilities while exploring and mapping new environment. They are equipped with sensors like IMUs and magnetometer, wheel encoders and GPS that can help them estimate relative displacements while using monocular or stereo camera, ultrasound sensors, LIDARs

and SONARs to perceive and extract visual features. For smartphone based VR systems fitting up encoders or displacement sensors is not possible and localization using just IMUs is very erroneous due to drift or noise in IMUs. Also LIDARs, Sonars are expensive hardware along fast processors, GPUs to process the maps and filter sensor data for state estimation. Thus VSLAM was ruled out as a possible approach to solve our problem of positional tracking for VR system.

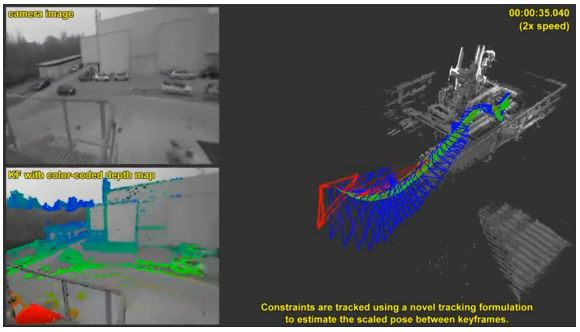

Fig. 1. Real time VSLAM implementation

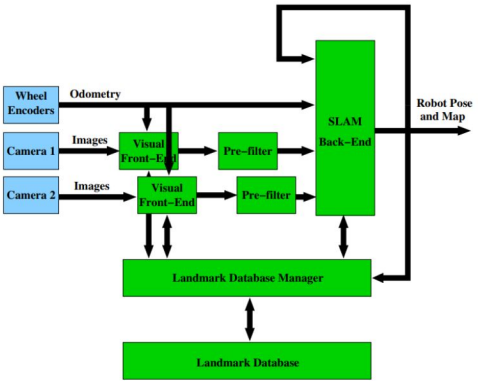

Fig. 2. Extended VSLAM architecture

C.2. Perspective-n-Point

Approach:

Perspective-n-Point (PnP) is the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. This algorithm uses a fixed number of points to generate mathematical solutions to estimate the camera pose. For a small number of points, multiple solutions are generated, leading to ambiguity. Increasing the number of points reduces the number of possible solutions and improves the accuracy of the position estimate. Generally, the use of a minimum of 6 points is recommended, to balance both computational complexity and accuracy. HTC Vive uses a combination of more than 20 IR markers on the HMDs and two expensive and sophisticated IR sensors to find relative distance between markers and these two sensors. These distances are then used to estimate camera pose using Outside-in architecture over a modified PnP model.

Issues:

Solving PnP problem using visual methods over an Outside-in architecture can prove to be very difficult or erroneous due to limitation in computer vision capabilities. They are very sensitive to lighting conditions, field of view and other factor. Line of sight visibility of markers is also a huge limitation to Outside-in technique. Similarly Inside-out visual tracking for indoor environments performance is also drastically affected by Line of sight, Field of View, lighting, computation resources. Positioning of markers and their unique identification all over the indoor space will prove to be challenge for estimating position of Human head position. These challenges were considered before ruling out this approach.

C.3. MONOCULAR TRACKING

Approach:

Monocular image sequences using a single RGB Camera can be used to track the position of a person, using the coordinates of different features on the image. A popular research work by Dushyant Mehta, et. al (http://gvv.mpi-inf.mpg.de/projects/VNect/) uses a Convolutional Neural Network based pose regressor, for full body pose estimation and character control (VNect). The person's joints are used as features, which are detected by the neural network, in real time. They have been able to achieve superior results as compared to existing CNN models or KINECT based tracking system. A real-time kinematic skeleton fitting method uses the CNN output to yield temporally stable 3D global pose reconstructions on the basis of a coherent kinematic skeleton. This can be leveraged to track shoulder joints or facial features of human wearing a head mounted display or Smartphone VR headset and estimate position of human in the indoor space/room. We implemented a MAT Caffe trained model of VNECT to understand its working and performance of tracking a human.

.

Fig. 3. 3D Real time pose estimation for various postures

Fig. 4. Side-by-side pose comparison with our method (top) and Kinect (bottom)

Issues:

All though the real time performance of tracking using a monocular camera was very satisfactory, the model was trained such that 3D joint coordinates were estimated with origin fixed at root joint (pelvis joint). So this CNN model can't be used to estimate absolute coordinates of human inside a closed room.We can only estimate relative locations of joints w.r.to pelvi of human. Also this model was found to fail with humans wearing HMDs. So this approach was ruled out despite implementing it on out computers due to above discussed reasons.

D. Final Approach - Outside-In method for Object Tracking

Building on the VNect model, we ideated on a similar outside-in, CNN based approach, using two cameras for tracking the person's motion in real time. Here we assumed a constrained space in a single rectangular room as region of interest for tracking the human head for VR position tracking system.

There are 2 possible variations of this approach. First one being two cameras placed on approximately centers of 2 adjacent walls to individually track human along 2 axes (X & Y) using first camera and other axis (Z or depth) using camera. This technique can prove to be more difficult in implementation and easy setup due to need for extensive camera calibration every time it is used in a new place or environment. On the other hand, two cameras can be places very close and adjacent to each other to achieve stereo tracking using 2 camera outputs. This is similar to depth perception made possible by the two human eye configuration or standard stereo cameras like ZED Camera. Images from left and right cameras are given as inputs to CNN model which will extract pixel coordinates of 17 joints like neck, ears, eyes, shoulders, ankle, hip, knee etc. A trained model of CMU Open Pose written using Tensorflow frameworks was implemented over Python for real time USB camera based keypoint (feature) extraction. Also this model was found to work excellently even with the human eye occluded with HMDs or mobile VR headsets. This model works for each and every person. This can be leveraged for our problem by extracting pixel coordinates of left and right shoulders from both images and then fed into disparity calculation for estimating real time 3D world coordinates w.r.to suitable origin. Shoulders were decided as a reference point for tracking human head as they are detected when the person is rotating or not facing the cameras.

The fact that shoulders are more visible and not occluded even when human is in difficult or weird poses while others joints are not visible.

D.1. CNN Architecture

Convolutional Neural Networks are widely used in image based applications, as they require relatively lesser preprocessing, as compared to that of other image processing applications. A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

We used OpenPose, a trained CNN model for object tracking (https://github.com/CMU-Perceptual-Computing-Lab/openpose). This deep neural network model incorporates both 2D multi person detection and 3D single person detection.The network consists of 57 layers, divided as follows:

- 18 layers for body parts location

- 1 layer for the background
- 19 layers for limbs information in the x direction
- 19 layers for limbs information in the y direction

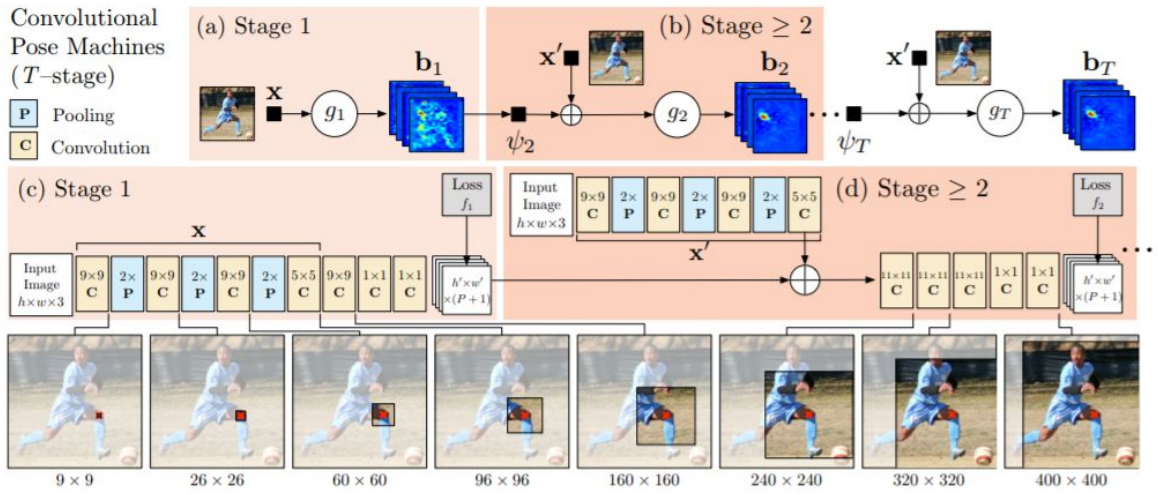

Fig. 5. CNN Architecture

Fig. 6. Results for various postures

D.2. Setup

Hardware Specifications:

- Ubuntu 16.04 laptop with NVIDIA GPU to run the python CNN tracking code
- Windows laptop to generate and update Unity virtual environment
- TP Link Wifi router to establish communication between the two laptops over TCP based Sockets
- Two Logitech C310 USB Cameras with 4.4mm focal length, 720p resolution and 60° Field of view

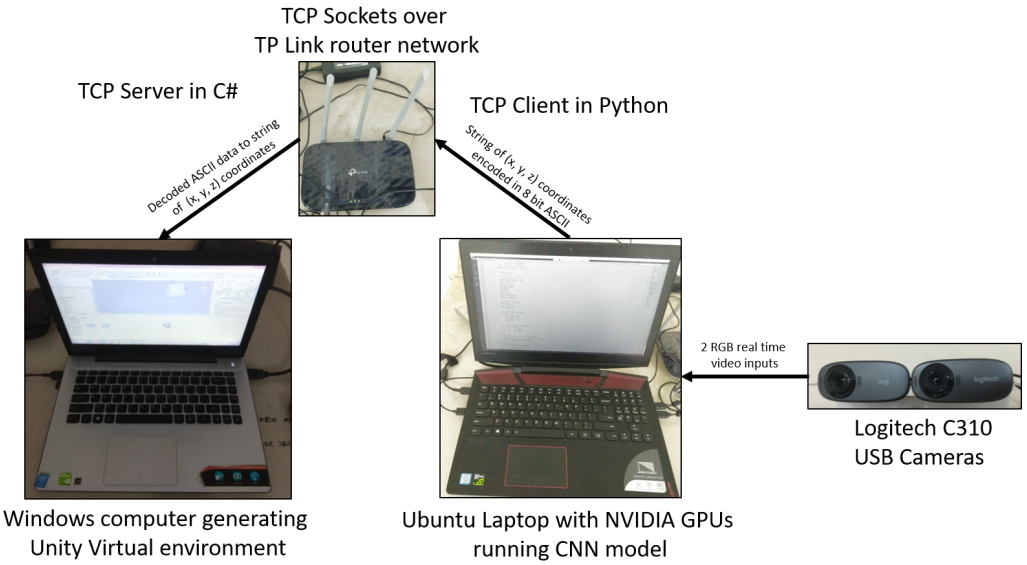

Fig. 7. Architecture of the setup

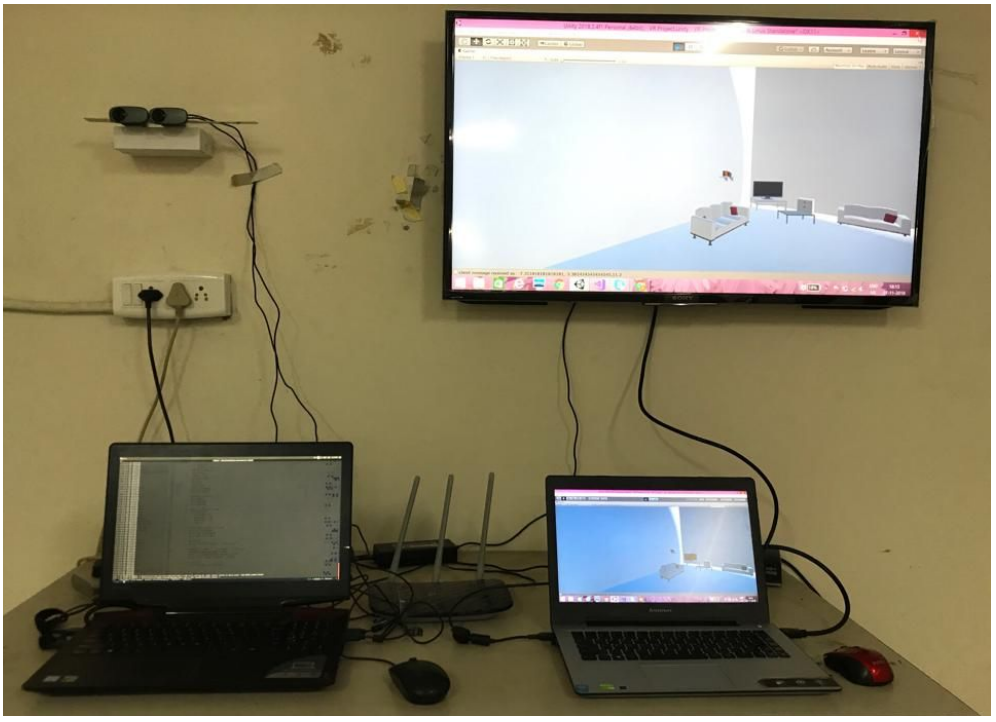

Fig. 8. Full setup in working during the testing of VR tracking system

D.3. World Coordinate Estimation

The two camera images were used to find the final coordinates of the person as follows:

The 2D image coordinates of the left and right shoulders were determined independently from both the left and right cameras by the CNN code. These

separate pairs of left-right shoulder coordinates were averaged separately for each camera, thus generating two sets of coordinates of a single point, which would be the centre of the neck. The disparity relations were then used to generate the 3D world coordinates. These coordinates were suitably transformed to shift the reference point to the centre of the eyes by increasing the y-coordinate by a distance of 300mm (average human value). The coordinates obtained from the first frame were fixed as the origin and were subtracted from subsequently obtained coordinates. This eliminates the need for calibration of origin of virtual environment. Triangulation is the method of determining depth from disparity. The following triangulation relations can be used to transform two sets of 2D pixel coordinates of a feature into 3D world coordinates.

$$z = f^{*}b / (xl - xr) = f^{*}b/d$$

$$x = xl^{*}z/f \quad or \quad b + xr^{*}z/f$$

$$y = yl^{*}z/f \quad or \quad yr^{*}z/f$$

Here,

f = focal length in mm

b = baseline distance between two cameras (lens to lens centre distance)

(xl, yl) = X and Y pixel coordinates of feature from left camera

(xr, yr) = X and Y pixel coordinates of feature from right camera

(x,y,z) = 3D world coordinates

d = disparity between two images

D.4. Filtering

The stream of coordinates determined in each frame shows some fluctuations, which cause disturbances in the Unity environment and create rendering problems. These fluctuations could be caused by small detection errors, fast, difficult motion or random noise. To safeguard against the rendering issues caused by these disturbances, we used a Median filter. The Median filter takes a set of measurements which are updated in each frame, sorts them and outputs the median measurement value. This value was taken as the measurement determined for that frame. This low pass filtering technique pushes the measurements with high variability to the edge of the set, where they are subsequently ignored. Using such a filter causes a small lag, particularly for fast motions, but generates a far smoother and stable Unity environment. The code used for the Median filter is shown in Fig. 9.

```
humansS = e.inference(imageS)
imageS = TfPoseEstimator.draw_humans(imageS, humansS, imgcopy=False)
centersS =TfPoseEstimator.joint_estimate(imageS, humansS, imgcopy=False)

#World Coordinate Estimation
try:
    ls_l = centersF[2]
    rs_l = centersF[5]
    xl = (ls_l[0] + rs_l[0])/2
    yl = (ls_l[1] + rs_l[1])/2
    #print("Left Cam: ",(xl,yl))
    ls_r = centersS[2]
    rs_r = centersS[5]
    xr = (ls_r[0] + rs_r[0])/2
    yr = (ls_r[1] + rs_r[1])/2
    #print("Right Cam: ",(xr,yr))

    if(xl == xr):
        continue
        print("xl == xr !!")
    zw0 = foc*base/(xl - xr)
    #Median Filter
    ZA.append(zw0)
    ZA.pop(0)
    ZA.sort()
    zw0 = ZA[2]
    xw0 = xl*zw0/foc
    yw0 = yl*zw0/foc
```

Fig. 9. Code snippet showing the implementation of median filtering using an python list 'ZA' of size 5 after which world coordinates are estimated

D.5. TCP Socket Programming

TCP Socket was used with the TP Link WIfi router to send final transformed coordinates from the laptop running the Python CNN Tracking code to the laptop running Unity. The Windows laptop with Unity behaved as the server, taking data from the Ubuntu PC, running the CNN code. The Python and C# scripts used for client and server side communication respectively, are given in Fig.6 and Fig.7

```
private void ListenForIncommingRequests()
{
    try
    {
        // Create listener on localhost port 8052.
        tcpListener = new TcpListener(IPAddress.Parse("192.168.1.109" + ""), 52595);
        tcpListener.Start();
        Debug.Log("Server is listening");
        Byte[] bytes = new Byte[1024];
        while (true)
        {
            using (connectedTcpClient = tcpListener.AcceptTcpClient())
            {
                // Get a stream object for reading
                using (NetworkStream stream = connectedTcpClient.GetStream())
                {
                    int length;
                    // Read incomming stream into byte arrary.
                    while ((length = stream.Read(bytes, 0, bytes.Length)) != 0)
                    {
                        var incommingData = new byte[length];
                        Array.Copy(bytes, 0, incommingData, 0, length);
                        // Convert byte array to string message.
                        clientMessage = Encoding.ASCII.GetString(incommingData);
                        Debug.Log("client message received as: " + clientMessage);
                    }
                }
            }
        }
```

Fig. 10. Code snippet of C# Server

```
import socket
HOST = '192.168.1.109' #SERVER IP
PORT = 52595
BUFFERSIZE = 1024
# TCP Socket client
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((HOST,PORT))
print("Socket Connection Established")
while True:
    s.sendall(campose.encode("utf-8"))
s.close()
```

Fig. 11. Code snippet of Python Client

D.6. Environment Generation in Unity

The 3D world coordinates obtained over TCP from the python client are used to move the camera in the environment created in Unity. The initial camera coordinates were set to (0, 0, 0) and a 45° rotation about the y-axis was preset to accommodate the complete environment. The frequency of the void update function was changed to 100/s, to allow for easier updating of frames in Unity. Also, the coordinates are received encoded in ASCII and require transformation to the Vector3 type that allows Unity to use them to change camera position.

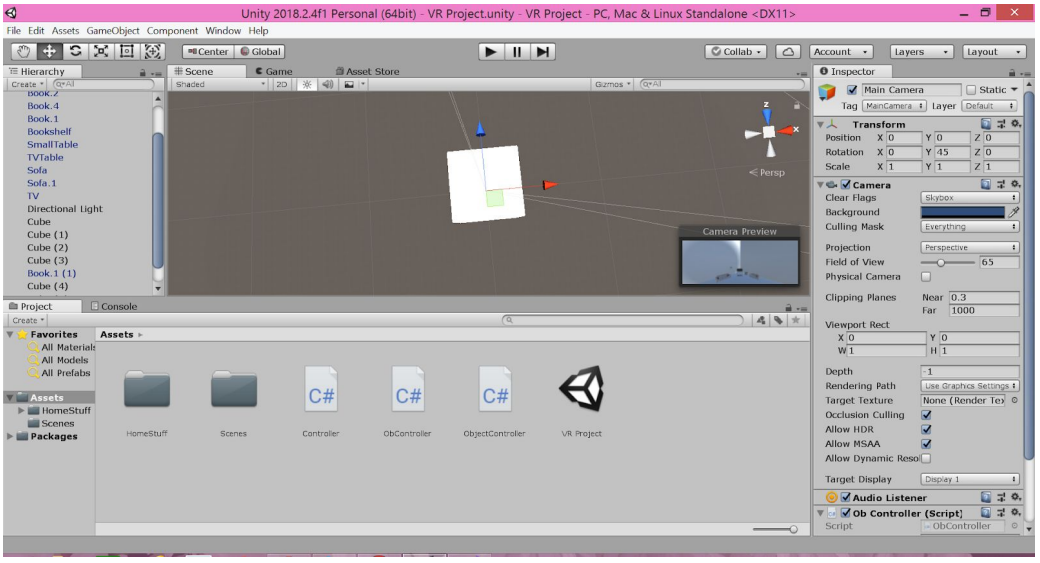

Fig.12. Unity Environment showing camera transforms

```
void Update()
{
    time += Time.deltaTime;

    if (time >= interpolationPeriod)
    {
        time = time - interpolationPeriod;
    }
    string[] data = clientMessage.Split(',');
    Vector3 pose = new Vector3(Single.Parse(data[0]), Single.Parse(data[1]), Single.Parse(data[2]));
    Cam.transform.position = pose;
    Cam.transform.rotation = Quaternion.Euler(0.0f, 45.0f, 0.0f);

}
```

Fig.13. C# script showing update function

E. RESULTS AND DISCUSSIONS

The CNN model was performing extremely well with an frame per second (fps) more than 15 even while processing both camera inputs as shown in Fig. 14. The CNN model was to found to perform well even when person was not facing the camera as shown in Fig. 15. It performed well even when the eyes were

occluded as shown in Fig. 16. Whenever the person goes out of the field of view or too close to camera shoulders are not detected which is printed on the Ubuntu terminal. There is no update in the coordinates in this case.

The developed VR system can work for both indoor and outdoor locations not affected by ambient lighting conditions or different people. For the developed VR system accuracy was found to be around the range of 20mm. Depth tracking was found to work smoothly with slow movements which can be observed from the attached video. It was tested to work even in case of jumping motion of human evident in the video.

Fig. 14. Joint tracking for left and right cameras implemented on a Ubuntu laptop with NVIDIA GPUs

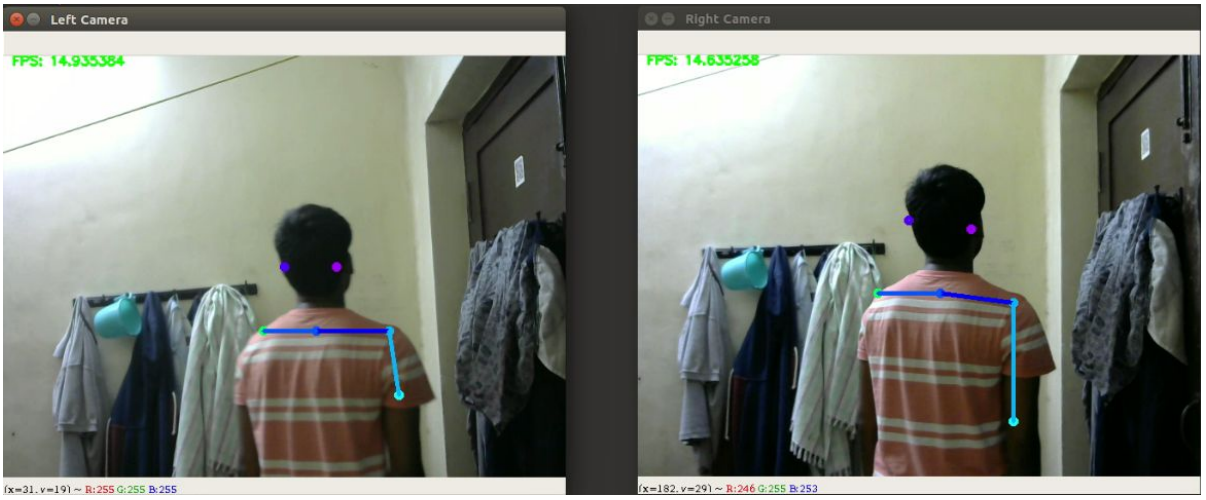

Fig. 15. Shoulder tracking performance with the human not facing the camera justifies choice of shoulder joint for position tracking of human body for VR

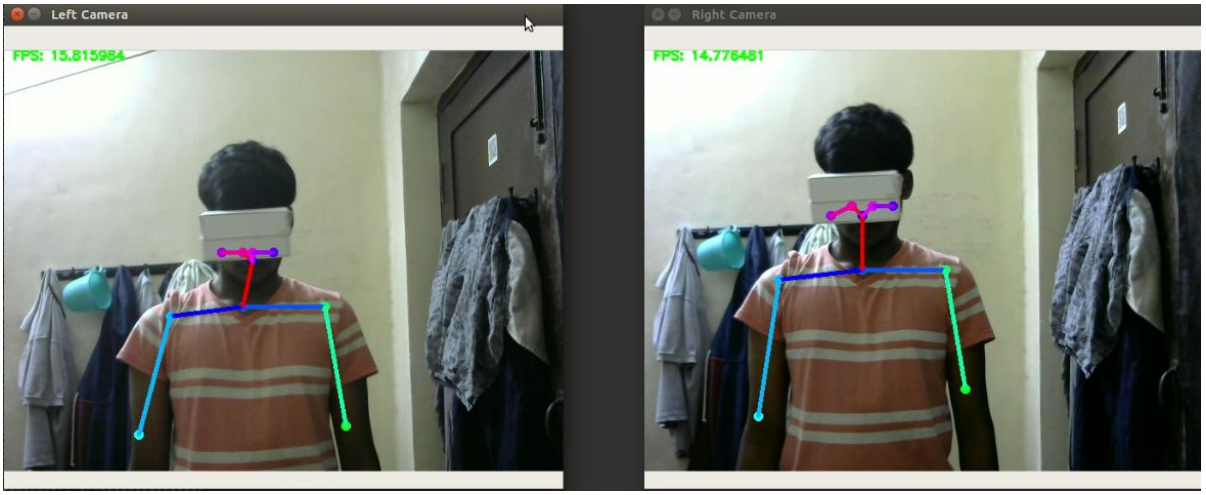

Fig. 16. Shoulder tracking performance even with the eyes occluded by white box which can be assumed to be a Head mounted display (HMD)

F. SUMMARY

A cost-effective and modular position tracking system using CNN to track the person and Unity to generate the virtual environment, using TCP for communication, was developed.

G. FUTURE WORK

- Integration of Orientation Tracking with the developed position tracking system for complete immersive experience
- Integration with virtual environment in smartphone, to enable position based environment change in mobile VR headsets
- Existing logitech C310 cameras can be replaced with higher field of view cameras and better resolution camera to be able to increase the visible region of tracking hence achieve improved mobility for VR users
- Cameras can be packed into compact and aesthetically pleasing mounts or inbuilt Stereo camera products like ZED camera can be used to eliminate issue with precise mounting of two cameras

H. CHALLENGES FACED

- Computationally expensive CNN models were used which demanded for high computational resources/processors and GPUs
- Precise mounting of camera is essential for accurate estimation of world coordinates, any lateral and transverse offset between camera centers can have drastic effect on depth estimation (z-coordinate) and tracking accuracy
- Noise in CNN based shoulder joint location estimates in both cameras also has an adverse effect on tracking accuracy. Fluctuations in shoulder joint

location estimates in both cameras led to fluctuations in the Unity virtual environments which was later eliminated by Median filtering technique
- Issue in interfacing TCP Sockets between Ubuntu and C# computers and parsing ASCII encoded data in Unity which caused a lag in unity environment updates
- VR tracking system was lagging in case of rapid movements

VR CONCEPTS USED

- Rotations and Translations, Quaternions
- Outside-In tracking system
- Computer vision based pose estimation techniques (VSLAM, PnP)
- Depth perception, stereo transform and eye transform
- Canonical view transform and viewport transform
- Drift errors, noise and fluctuations in tracking system

REFERENCES

[1] https://github.com/CMU-Perceptual-Computing-Lab/openpose
[2] http://gvv.mpi-inf.mpg.de/projects/VNect/
[3] https://courses.cs.washington.edu/courses/cse576/book/ch12.pdf
[4]https://gist.github.com/danielbierwirth/0636650b005834204cb19ef5ae6ccedb
[5] https://nptel.ac.in/courses/106106138/
[6]http://www.vudream.com/slam-inside-out-positional-tracking-solution-for-vr-and-ar/
[7]https://dspace.mit.edu/bitstream/handle/1721.1/36832/16-412JSpring2004/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring2004/A3C5517F-C092-4554-AA43-232DC74609B3/0/1Aslam_blas_report.pdf
[8] Eric Marchand, Hideaki Uchiyama, Fabien Spindler. Pose Estimation for Augmented Reality: A Hands-On Survey. IEEE Transactions on Visualization and Computer Graphics, Institute of Electrical and Electronics Engineers, 2016, 22 (12), pp.2633 - 2651.

ACKNOWLEDGEMENTS

We thank Prof. M. Manivannan associated with Applied Mechanics department of IIT Madras for providing us with this wonderful opportunity to gain an insight into this trending field of Virtual Reality. We are grateful to him for his valuable inputs and guidance which helped us in channeling our ideas in the right direction and shaping this into an interesting problem statement. We thank Joseph Hosanna for his timely support and suggestions. We are grateful to Radha Krishna Jalamanachili for his help with video editing and testing. We thank Sidharth Ramesh for providing his sophisticated laptop and Abhishek Peri for helping us out with CNN code.