PES INSTITUTE OF TECHNOLOGY

(An Autonomous Institute under VTU, Belgaum)

PROJECT REPORT ON

TITLE: 3D PERSPECTIVE ON A 2D SCREEN USING FACE TRACKING

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR THE TWO CREDIT COURSE

MINI PROJECT (12EC309D)

SUBMITTED BY

Akshay Varun S (1PI12EC006)

Naman Nandan (1PI12EC061)

Sumukh Desu (1PI12EC117)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PES INSTITUTE OF TECHNOLOGY

100 FEET RING ROAD,

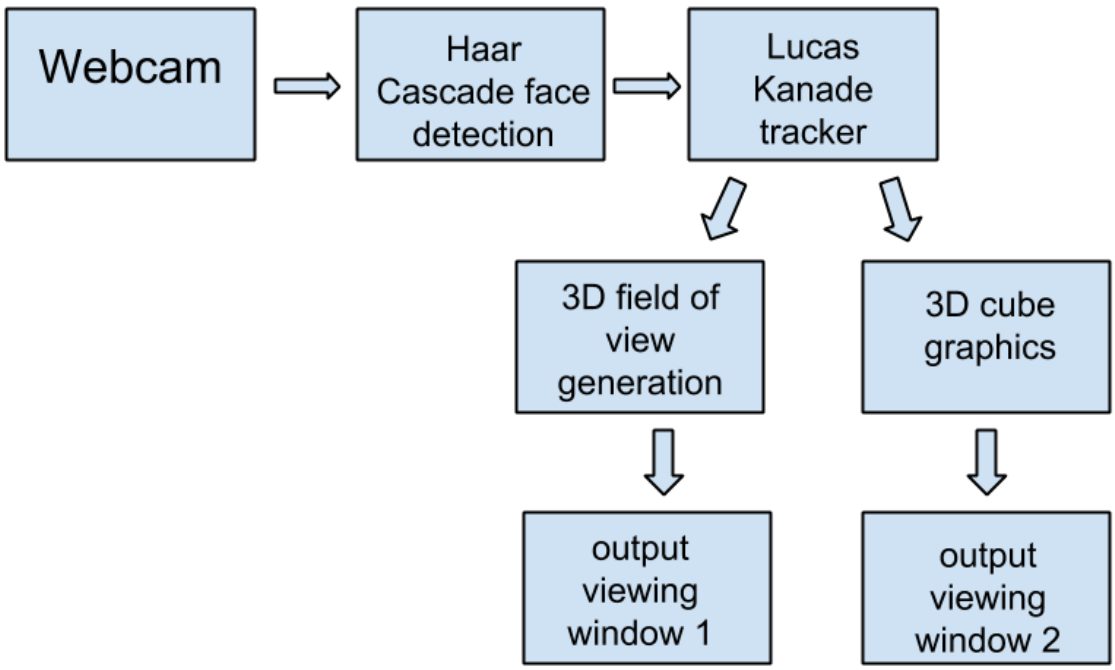
BSK III STAGE,

BANGALORE - 560 085

2014

Contents

ABSTRACT :-



This project aims to deliver a 3D viewing experience on a 2D screen by initially identifying and subsequently tracking the user's face. Based on the angle at which the user is viewing the screen, the orientation of the virtual objects displayed on the screen is suitably modified. This gives an impression that the user is viewing a real life object although only an appropriate projection of the object is displayed on the screen. This is achieved through face tracking performed on a webcam feed using the opencv library with python bindings. This can potentially be used to deliver a more realistic first person gaming experience.

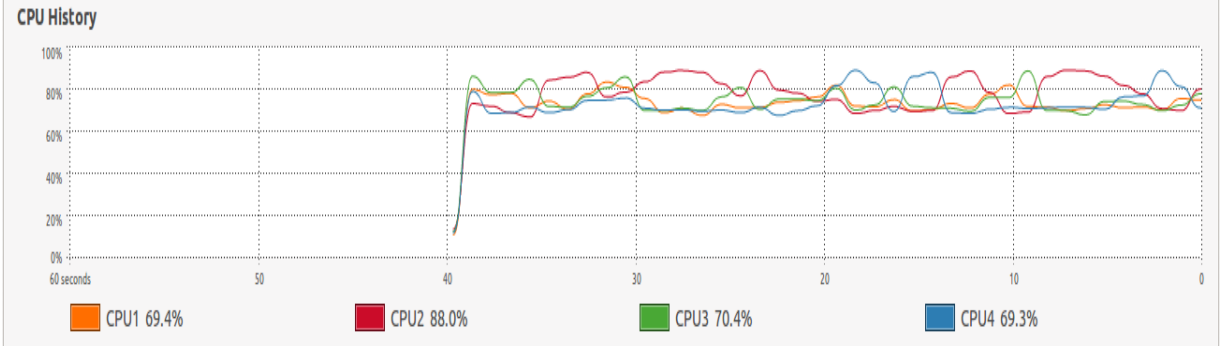
OBJECTIVES :-

→ User face detection from webcam feed.
→ Efficient and Robust tracking of the user's face.
→ Perspective transformations on a cube based on the user's viewing angle.
→ Generation of graphics to simulate a 3D field of view effect.

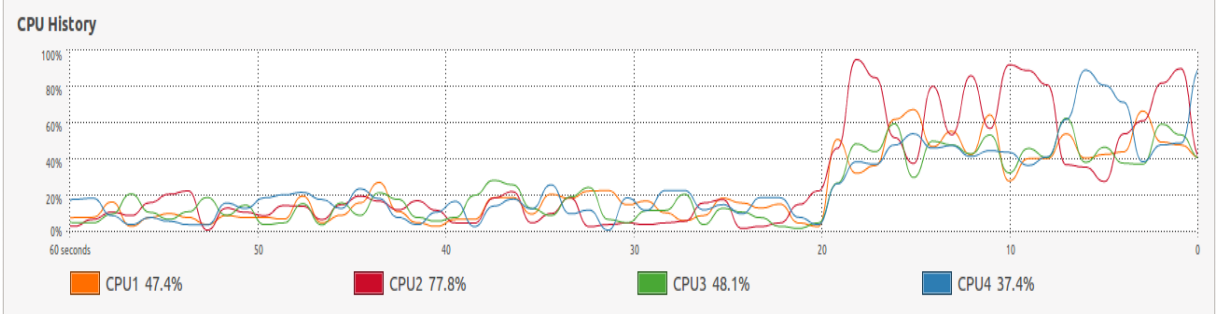
BLOCK DIAGRAM :-

IMPLEMENTATION :-

Initially, face detection and tracking was implemented using the haar-cascade method in which a frame obtained from the camera feed is processed in order to detect faces. This process was repeated on every subsequent frame obtained from the camera. Since the haar-cascade algorithm is a CPU intensive task and face detection is performed on every frame obtained from the camera, the usage of system resources was very high as shown below.

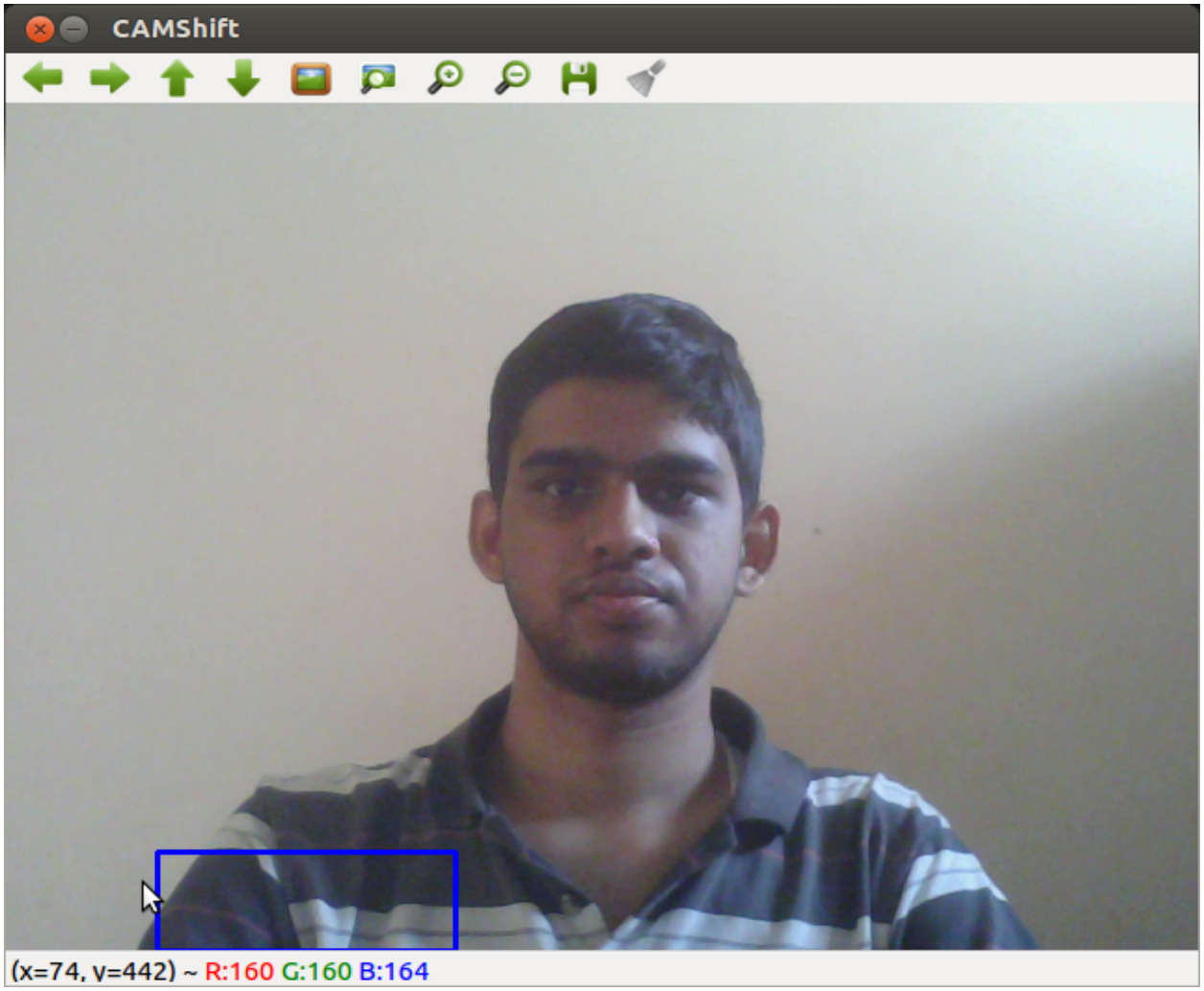


Therefore, in order to reduce the CPU load, we used haar-cascade algorithm for face detection and the CAMShift method for face tracking. Here, haar-cascade face detection is performed once every ten frames. The duration in which face detection is not performed, the identified face is tracked using the cam-shift method. This reduced the CPU load significantly as can be seen from the graph below.



On the contrary, Cam-shift algorithm combined with histogram based tracking was inefficient since it was colour based and not very robust. The algorithm is simple, a bounding box which contains the ROI (Region of Interest) is provided in the initial frame. The Histogram is then calculated. In the subsequent frames, using Backprojection, the

region which is similar to the ROI is found and then using the CAMshift algorithm, the position of the bounding box is updated. This method was an improvement compared to the initial face detection but as mentioned above was not very robust and highly inefficient. We encountered problems such as the bounding box disappearing and also tracking objects which were similar in colour to the skin as can be seen from the image below.

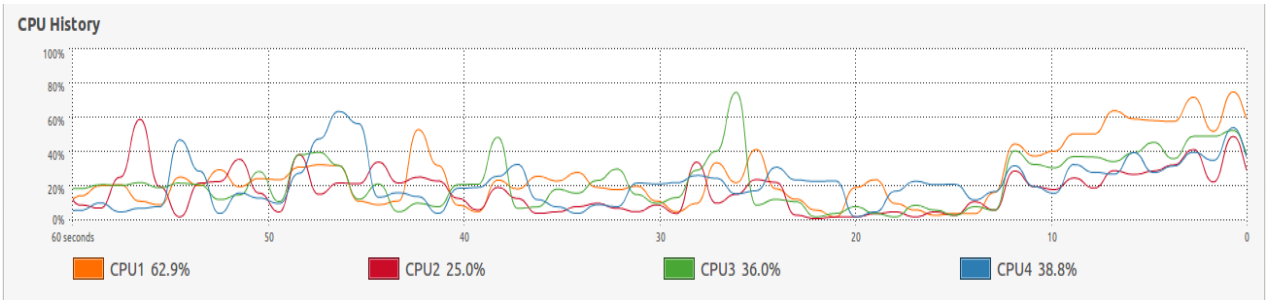


Therefore, we decided to use the Lucas-Kanade optical flow method, which is a better tracking algorithm as compared to cam-shift.

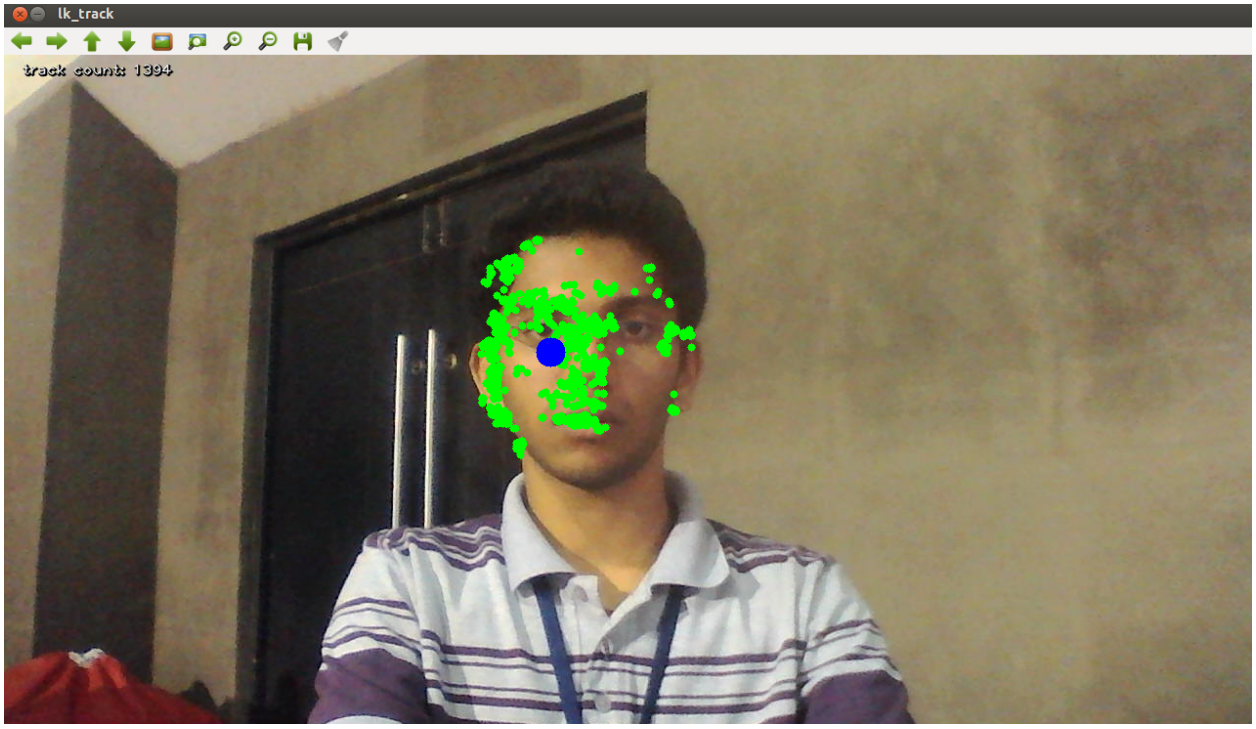
In this implementation, face detection using haar-cascade is performed once every twenty frames. The duration in which no face detection is performed, the identified face is tracked using the Lucas-Kanade optical flow method.

Initially, using haar-cascade, the bounding rectangle around the user's face is obtained and this is considered to be the ROI (Region of Interest). Using the "goodFeaturesToTrack" function available in the opencv API, the distinct feature points that can be tracked in the ROI are identified.The features obtained so,are the ones that can easily tracked and are the most prominent corners of the ROI. Then, the location of these feature points are updated in each frame using the Lucas-Kanade optical flow method. When the haar-cascade face detection is performed again, if any new feature points are identified in the ROI, they are added to the list of feature points to track and if any of the old feature points are outside the ROI, they are deleted from the list of feature points to track and hence are no longer tracked. This in turn results in a robust yet efficient method to detect and track the user's face.

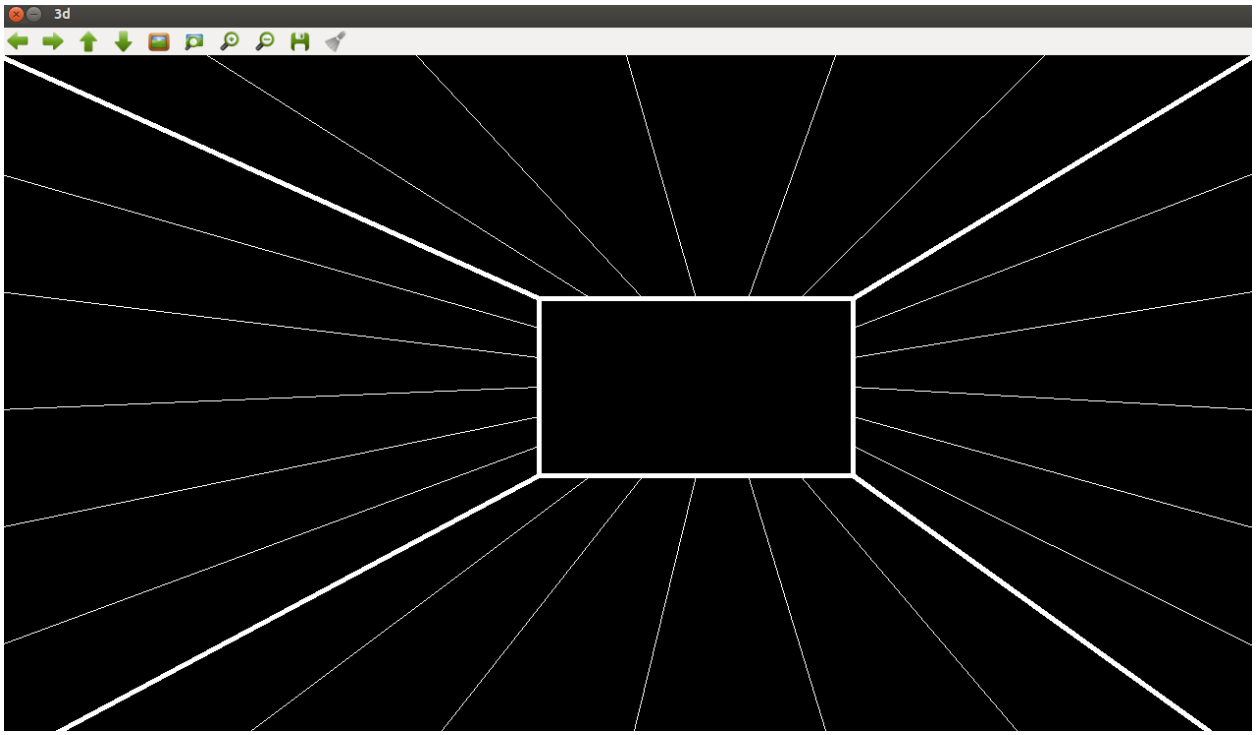
The graph below shows the CPU usage for the above method



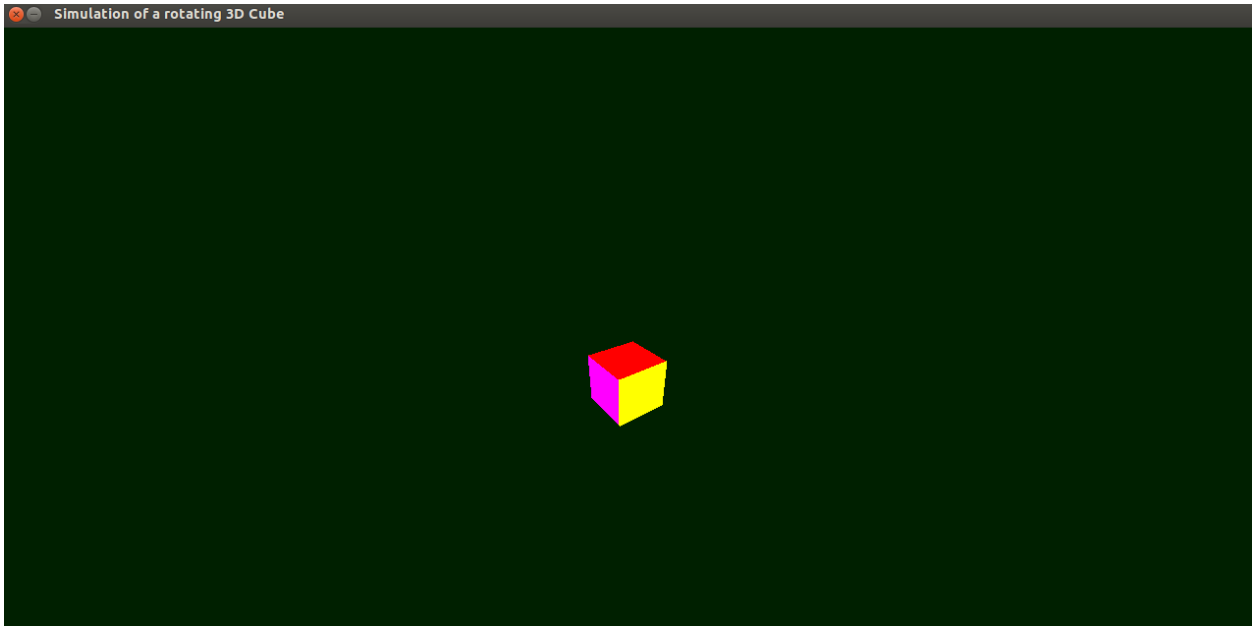
→ Face detection and tracking using haar-cascade and Lucas-Kanade method
(In the image shown below, the green points represent the feature points that are being tracked and the blue point represents the mean of the feature points)



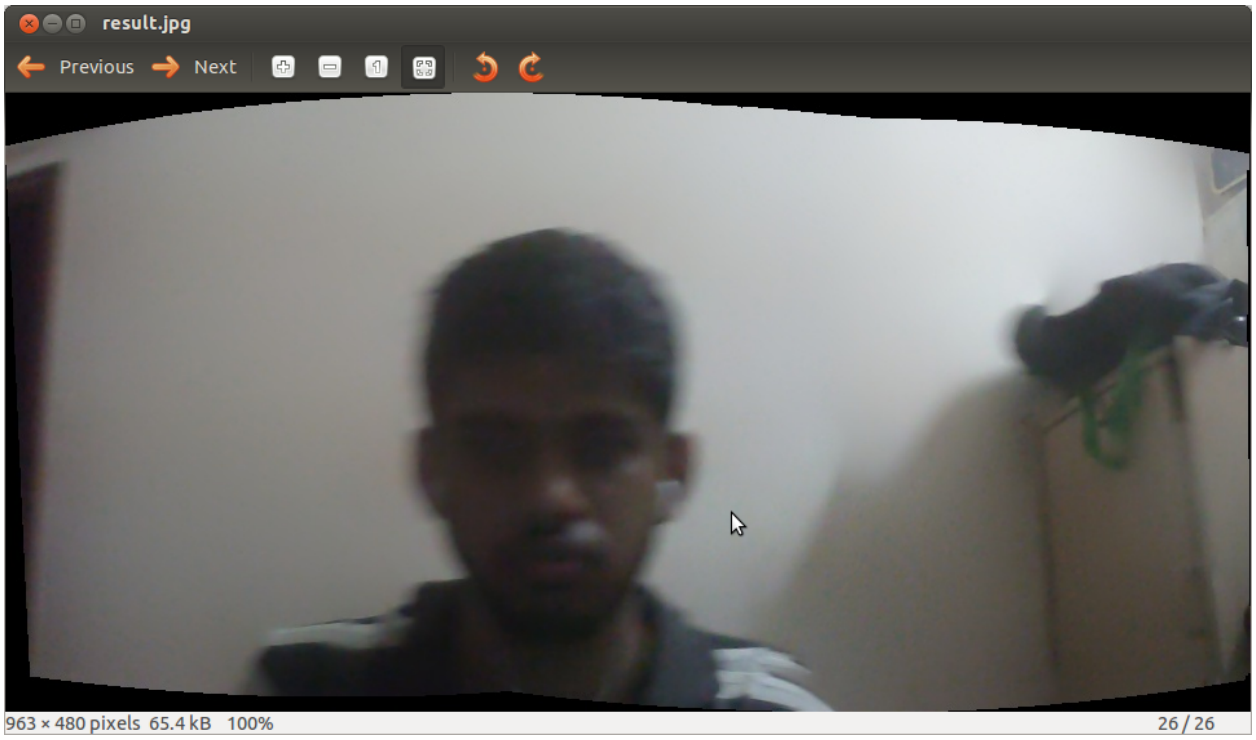
→ 3D scene generation



→ 3D cube graphics



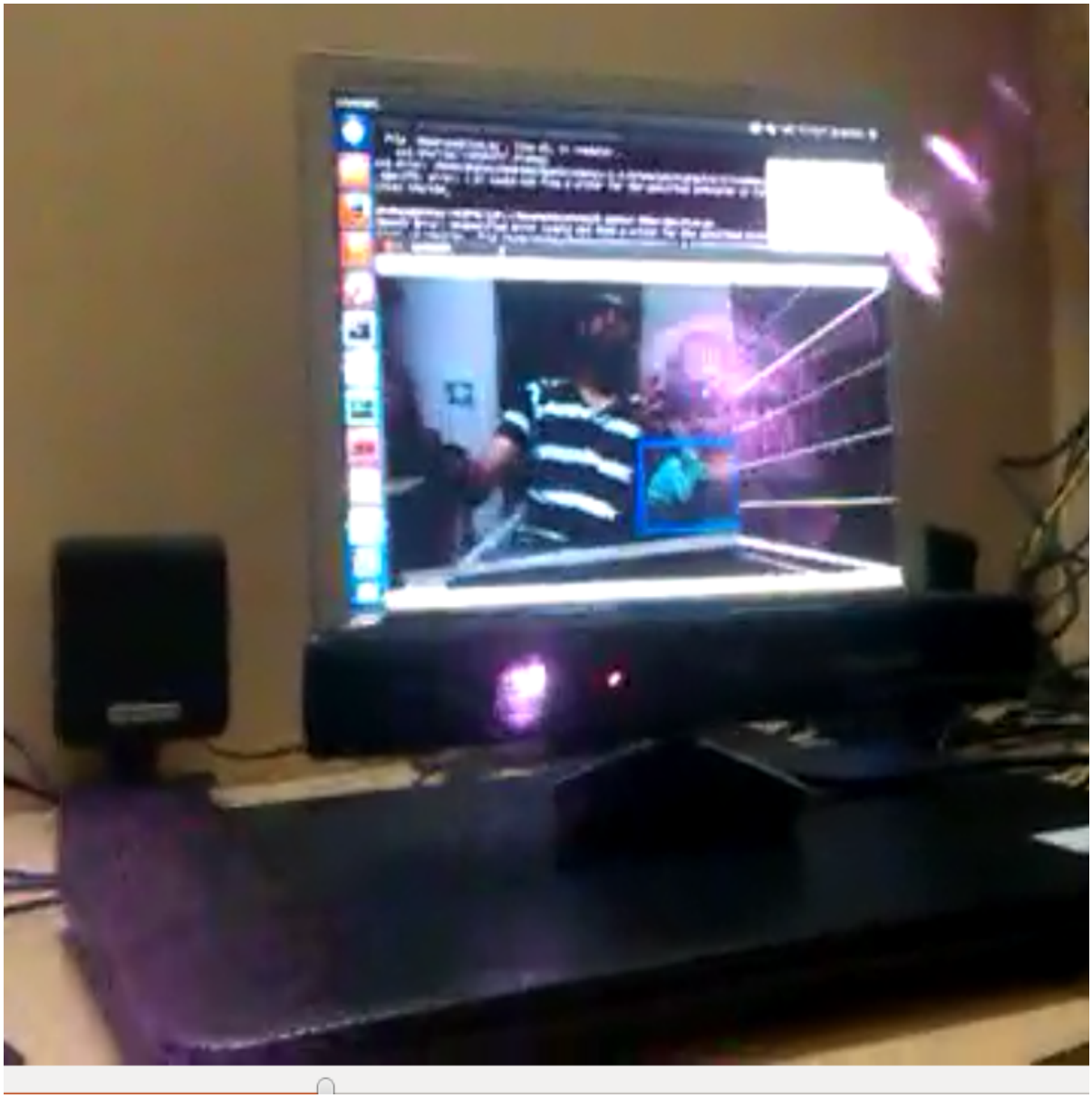
→ Panoramic Image Created using the Image Stitcher Class of OpenCV



This panorama was created using the OpenCV Stitcher Class. We had to supply images that should be a part of the panorama as arguments. The stitcher class finds the

features between the images and the similar parts in the two adjacent images. After finding the necessary features, it ‘stitches’ the images into one panorama. The main reason we wanted to create this is because we wanted to implement the perspective projection on an actual image and not just a graphical representation. Based on the position of the user’s face, only a certain portion of the panorama is displayed.

→ Using the Xbox Kinect Depth Camera,we were able to obtain depth data so as to add zoom in-out functionalities to the image.



The main reason we wanted to use the Kinect camera is because we wanted to add the depth dimension to the system, so that as the user walks towards/away from the screen, the virtual object becomes larger/smaller. This enhanced the 3D viewing experience of the virtual object. The LibFreenect library with python wrappers was used to obtain data from the kinect. This data was then processed using opencv.

FUTURE WORK :-

→ Implementation of reliable depth tracking of the user's face using the data from the depth sensor in the kinect.

→ Implementation of a gaming interface using the work done so far.

REFERENCES :-

1.OpenCV Python Documentation-
http://docs.opencv.org/trunk/doc/py_tutorials/py_tutorials.html

2.OpenCV Image Stitcher Class-
http://docs.opencv.org/modules/stitching/doc/stitching.html

3.Pygame 3D cube Demonstration-
http://www.pygame.org/project-Rotating+3D+Cube-1859-.html

4.Johnny Lee Wii Hack website
http://johnnylee.net/projects/wii/

5.OpenCV Feature Detection.
http://docs.opencv.org/modules/imgproc/doc/feature_detection.html

6.OpenCV Lucas-Kanade Optical flow method description
http://docs.opencv.org/master/doc/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html

7.Lucas-Kanade algorithm description
http://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method