UNIVERSITY OF MARYLAND

ENPM 673 P1

SUBMITTED BY,

SAKET SESHADRI GUDIMETLA | CHAYAN PATODI | PRASANNA MARUDHU BALASUBRAMANIAN

INTRODUCTION

This project dealt mainly with the detection and tracking of an AR tag (or a fiducial marker) that acts as point of reference of in real world. The aim of the project was to detect an AR tag in a given input video sequence and keep track for the entirety of the video. We have understood and implemented new concepts in this project like corner detection, edge detection, homography, augmented reality etc. These concepts have been briefly described as follows –

1. **Harris Corner Detection algorithm**: - As the name implies this algorithm is used to find the corners of a given target. The working of this algorithm basically relies in the difference in intensity for a displacement of (u,v) in all directions. This is mathematically expressed as follows-

   $$E(u,v) = \sum w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

   Where w represents a rectangular window or a gaussian window that gives weight to the pixels underneath it. Here the term inside the square bracket represents the change in intensity for a random displacement in x-y direction. For corner detection to be successful this E function is supposed to maximized. For implementing it in the code we used the opencv function "cv.cornerHarris()".

2. **Canny Edge Detection: -** It is an edge detection operator that relies on a multi stage algorithm to detect a wide range of edges in images. The entire process is composed of the following steps –
   a. **Noise reduction** – Noise is removed with the help of a Gaussian filter.
   b. **Finding Intensity Gradient** – Sobel kernel is used to filter the smoothened image both vertically and horizontally. Gradient is just the combination of these two components.
   c. **Non-maximum Suppression** – Removal of unwanted pixels which may not constitute the edge.
   d. **Hysterisis Thresholding** – This stage separates the edges that can accurately be called edges from the ones that are not so accurate.

   Canny Edge Detection in OpenCV is implemented using cv.Canny().

3. **Homography: -** In projective geometry homography is defined as the transformation between two planes in projective space. It basically helps us transform the real-world coordinates into that of the coordinates understood by the camera. The homography is performed by multiplying the homography matrix (a 3x3 matrix) with the real-world coordinates to get the coordinates in the camera's frame of reference. It is given as,

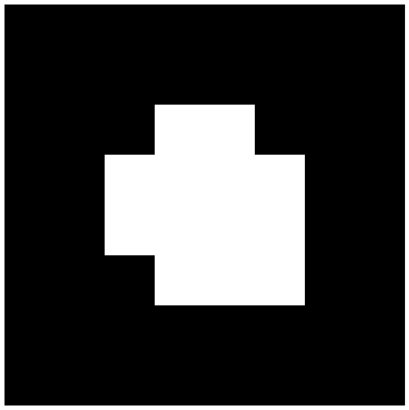
$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

Let us consider the first set of corresponding points — (x_1, y_1) in the first image and (x_2, y_2) in the second image. Then, the Homography H maps them in the following way.

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

The applications of Homography are in (but not limited to) augmented reality, image stitching, perspective correction etc.

4. Fiducial markers/ AR tags – A fiducial marker is an object placed in the field of view of an imaging system which appears in the image produced, for use as a point of reference. It looks as follows -



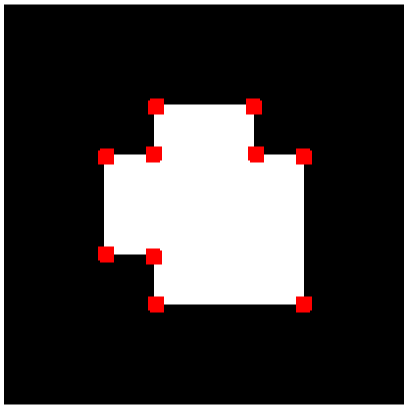
The fiducial markers are widely used as reference points in Physics, Geophysical surveys, Metrology, ECGs, Radiotherapy, Augmented Reality etc. As used in the project the fiducial markers are greatly useful in solving problems of integration between real world view and the synthetic images that augment it. Fiducials of known pattern and size can serve as real world anchors of location, orientation and scale. They can establish the identity of the scene or objects within the scene.

WORK DONE AND RESULTS

1. **Detection**: -

For detection portion of the code the task at hand was to detect the corner points of the tag. We explored the Harris and Shi-Tomasi corner detection algorithms respectively, out of which we opted Harris as it was easier to grasp as well as easier to code.

The initial step is to convert the image to gray-scale, as Corner Algorithms are used on Images that are gray-scale, as it is easy for them to detect the corners. The logic being it is easy to differentiate between shades of black and white, instead of all the colors. We gave the input to the algorithm, by adjusting the parameters such as block size and thickness and the output was stored in dst , which is the usual syntax. This dst image is than dilated for making the corners clearer, which becomes a more important factor for detecting the corners of the tag in the video. After dilation, we set a threshold, which varies the number of corners present, this threshold value may vary according to the image.



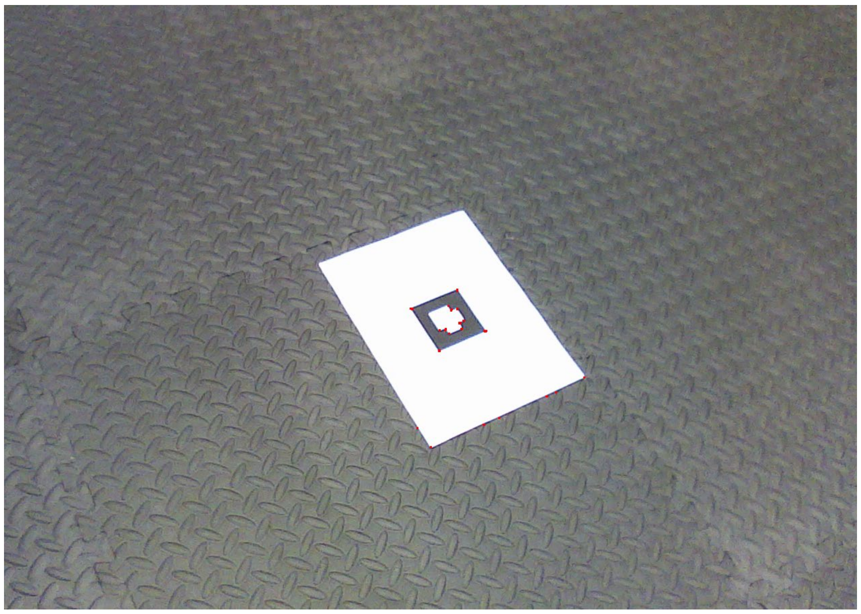
The code for the above-mentioned steps, can be found with the file name Harris.py, which is presented with this report.

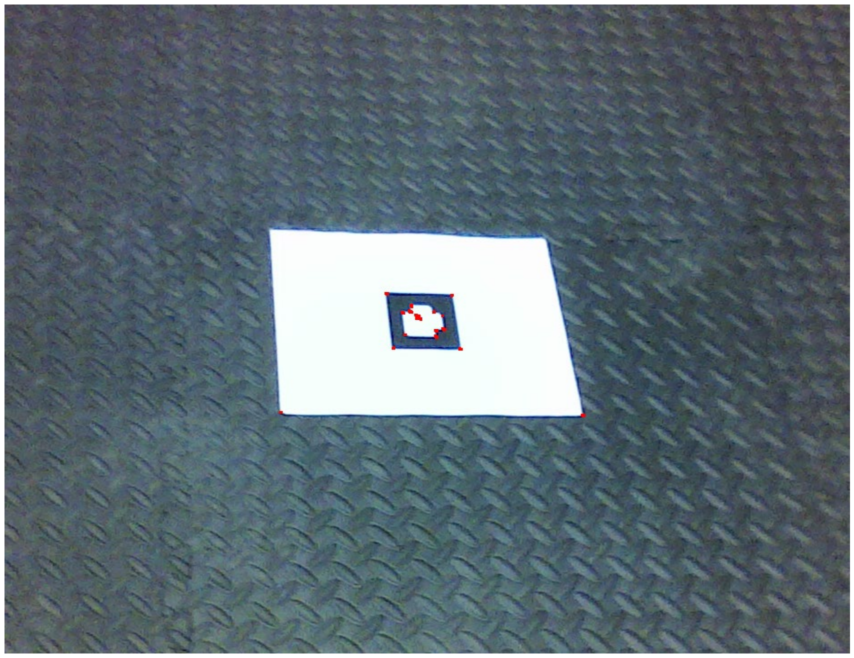
Finding corners of the tag in the video, is a more difficult job, because instead of finding corners in an image, we have to find the corners in continuously moving images (Frames).

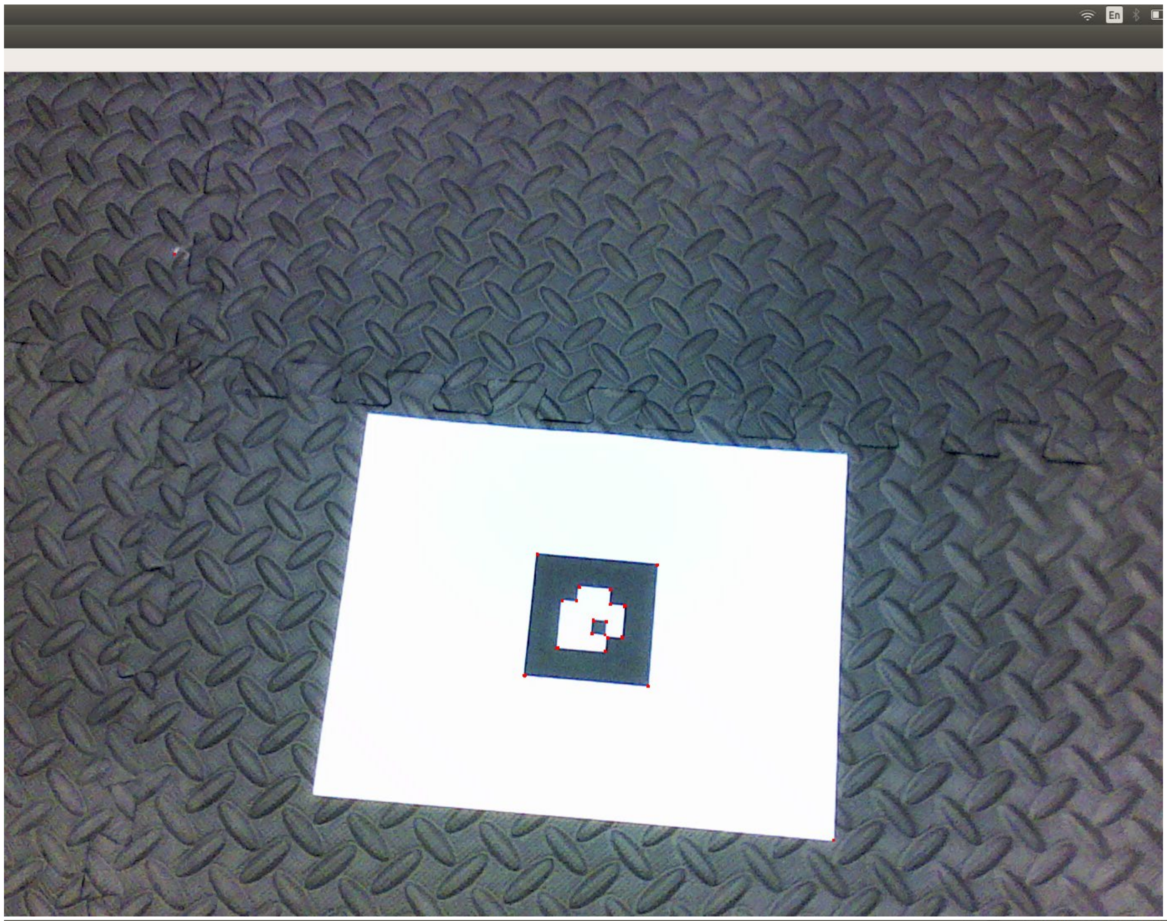
The logic and the flow remain the same, the only trick here is that you capture the video frame by frame and apply the corner Harris algorithm on each frame, following the steps.

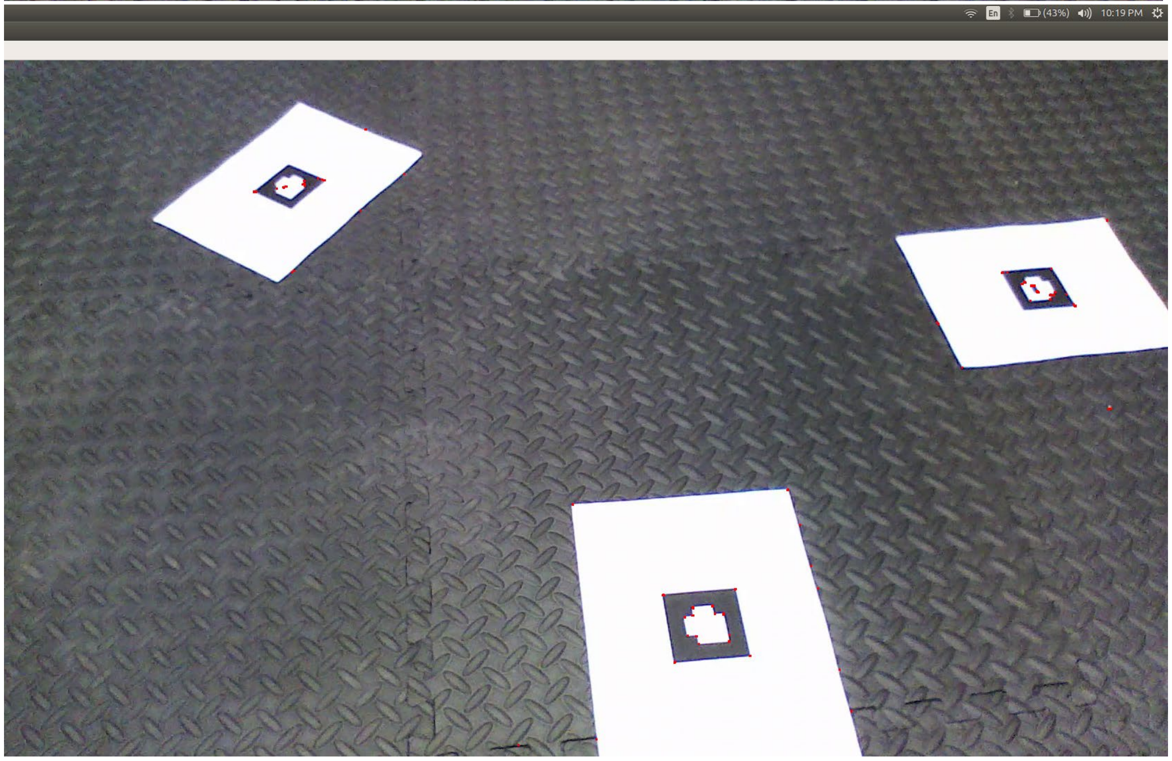
In our code, the user is asked which video he wants to detect the corners in and, is given 4 options. As the input is given, the video starts, and the corners are detected of the tag, the inner tag and the page. (Some random corners maybe detected at some point due to the motion, but they are negligible. They were reduced by adjusting the threshold value.)

The code for the above mentioned steps, can be found with the file name HarrisV.py, which is presented with this report. The outputs are shown below.






En


En
(43%)
10:19 PM

As we know that the tag is a 8x8 grid, which contains padding, the orientation and the ID. Padding is of no value to us, it is used so as to make it's detection easy on a white background. Eliminating the padding, we are left with the orientation and the ID, which is given by the data bits.

As mentioned in the project description, the inner 2*2 grid, represents the data bits. Each black grid denotes 0 and each white grid denotes 1. The 0 and 1 are the binary representation of the square blocks. To find the ID, what we did was identify the color of the grid and and appended the values in a clockwise direction, which gave us 1111 in binary form, which then converted gives us 15 in integer value. Tbe same thing is done with every video.

The logic behind the determining the ID in the videos, is the frames are captured, each frame is brought to the front plane using perspective transformation and is identified for black and white square blocks. The orientation is done in a similar fashion, but for a larger grid, i.e 4*4. The orientation helps us to identify the tags in the videos for superimposition of the template image. The 4x4 grid also denotes a binary representation, 0 for black and 1 for white.

While superimposing the template image, we had to search for the orientation, as even though the tags varied in the data bits, they had the same orientation bits, which helped us superimpose the template image on the AR tags. Also, their binary representation helped us to superimpose the template image in its original orientation, even if the tag was rotated or flipped, the template image was superimposed with the same orientation as of the original tag. We understood how tags / markers can be helpful for tracking and detection for future robots and certain other scenarios and how the orientation and data representation is important.

2. Tracking: -

The aim of this section of the problem was to perform superimposition of the given image on to the detected AR tag and keeping the orientation of the tag and the template image aligned throughout the entirety of the video. For solving the tracking part of the problem, we have used the OpenCV functions cv2.getPerspectiveTransform() and cv2.warpPersepctive().

For perspective transformation we require a 3x3 matrix. To find this we need 4 points on the input image and the corresponding points on the output image. The condition being that 3 of those 4 points must be collinear. This transformation matrix is found by using the cv2.getPerspectiveTransform(). The we apply the cv2.warpPerspppective along with this 3x3 transformation matrix to get a perspective transformation of the image.

3. Virtual cube placement: -

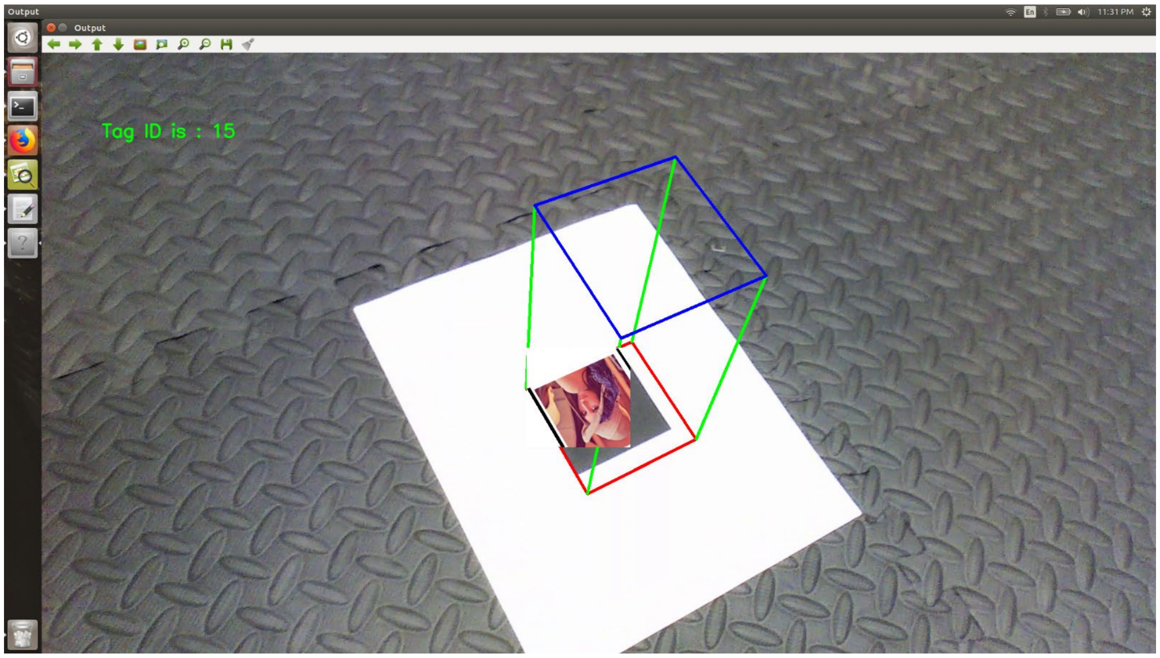
The supplementary data presented on the Homography was very much helpful and a detailed explanation of the steps, that were to be taken to generate the virtual cube. We generated the A matrix, we decomposed it using SVD, found the V matrix. Using that V matrix, we got a vector h, which was then reshaped to get the Homography Matrix.

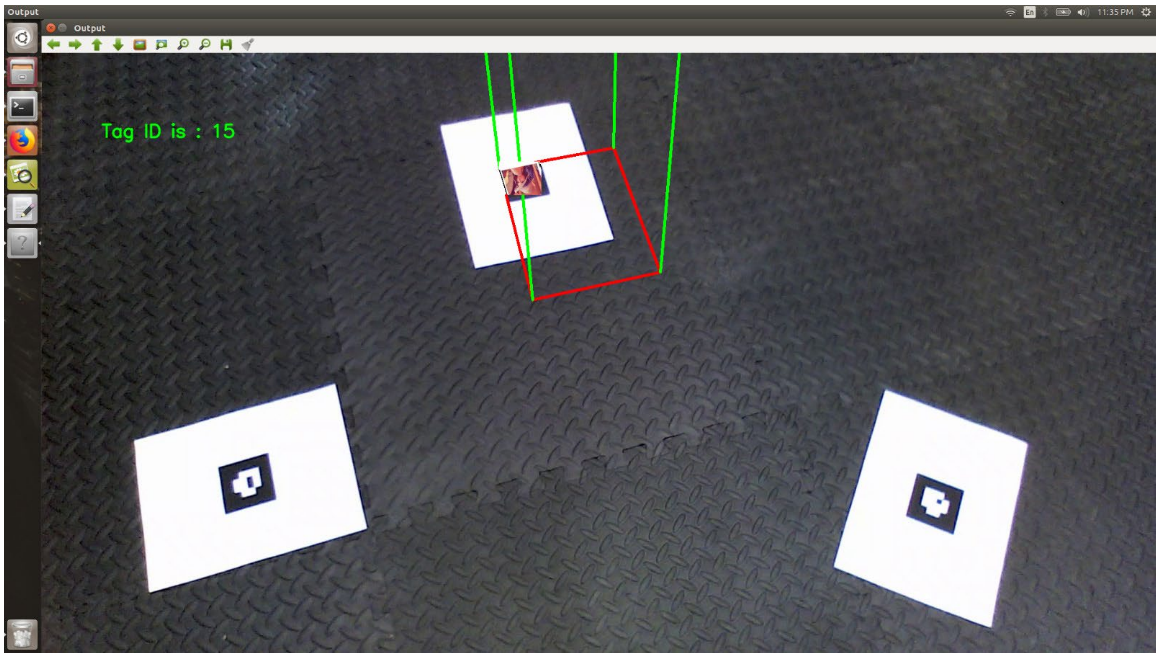
We were already given the Camera Calibration Matrix, which was then used inversed and multiplied by first two columns of H matrix, to obtain the Lambda, which is a scaling factor. We found the B matrix using Lambda, Homography matrix and the Camera Calibration Matrix. With the help of B matrix, we were able to find the Rotation Matrix and the Translation Vector.

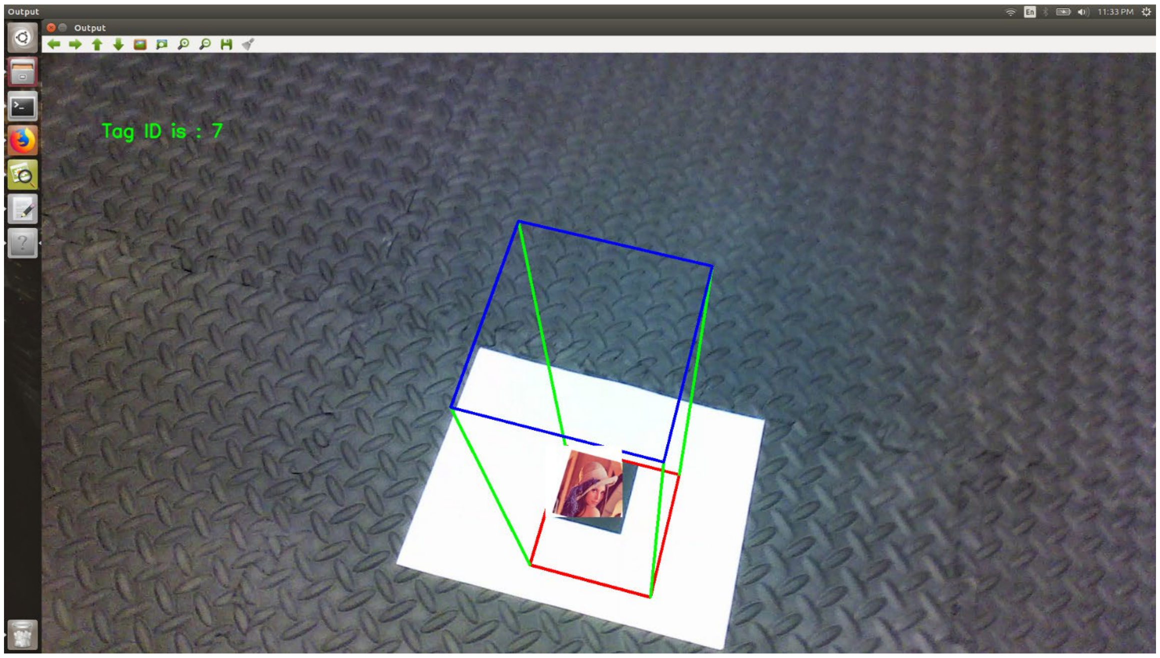
With every parameter calculated and known, we were able to project the cube onto the tag. Functions like numpy.linalg.norm , numpy.linalg.inv , numpy.dot , numpy.concatenate etc are used for deriving the values and matrices. This part of the project helped us understand how can the world cordinates and the camera coordinates can be related and how Homography estimation and Projection of a 3-D shape can be done on a 2-D Image. The outputs for this section of the code are as follows. The file ID_Cube.py shows this implementation.

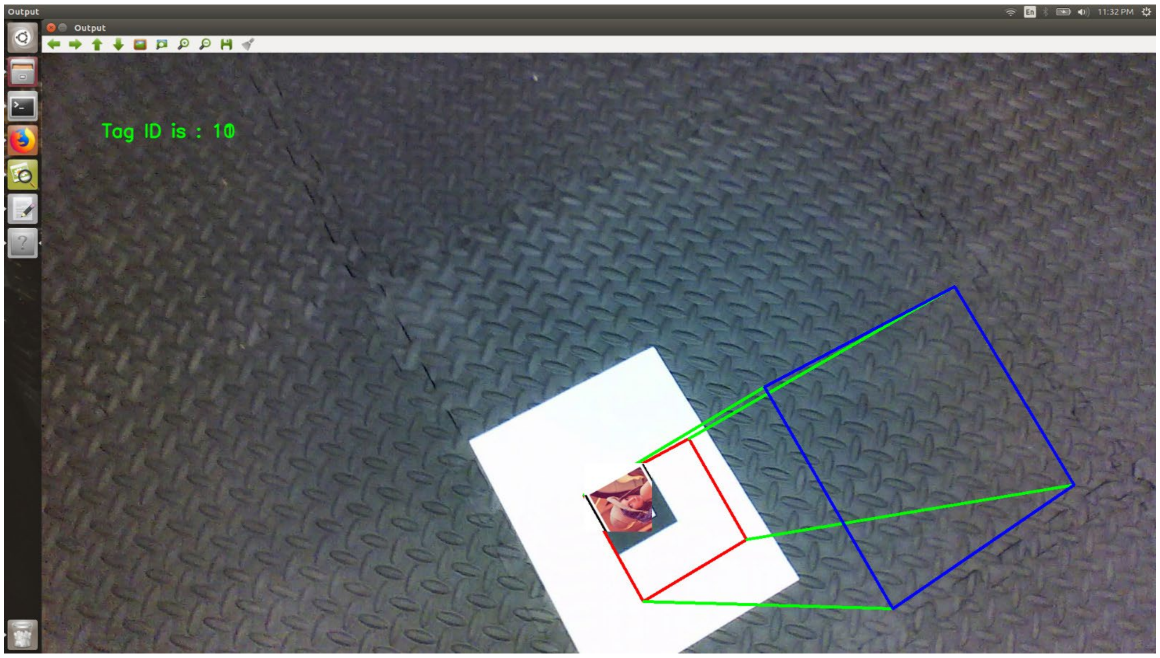
THE PYTHON VERSION USED WAS 2.7 HENCE IF THE CODE THROWS AN ERROR IN THE HIGHER VERSION JUST TRY TO RUN IT USING 2.7

GENERATING THE CUBE AND SUPERIMPOSING THE TEMPLATE IMAGE MIGHT TAKE A LITTLE BIT OF TIME BECAUSE OF THE MOTION BLUR AND LOW QUALITY OF THE VIDEO


Output
Output
11:31 PM
Tag ID is : 15


Output
Output
11:35 PM
Tag ID is : 15


Output
Output
11:33 PM
Tag ID is : 7


Output
Output
11:32 PM
Tag ID is : 10

REFERENCES

This section states the useful websites we found on the internet that helped us reach the goal of completion.

1. https://rdmilligan.wordpress.com/2015/07/22/glyph-recognition-using-opencv-and-python-mark-ii/
2. https://rdmilligan.wordpress.com/2015/07/31/3d-augmented-reality-using-opencv-and-python/
3. https://www.youtube.com/watch?v=Udz95pzobr4
4. https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/
5. https://docs.opencv.org/2.4/doc/tutorials/tutorials.html
6. https://www.wikipedia.org/