PROJECT 1- ENPM 673

Submitted by :
Srujan Panuganti : 116302319
Mrinali Vyas : 116189866
Akshitha Pothamshetty: 116399326

Collaborated with: Sanchit Gupta, Aaradhana KS.

Pre-requisites -

The following Project is developed in python 3 and the packages used in this project are the following- cv2, numpy, copy
This project uses OpenCV 3.4.x

Instructions to run the code:

Input :
Enter one the option below:
Enter 0 for for Tag0 video
Enter 1 for for Tag1 video
Enter 2 for for Tag2 video
Enter 3 for for MultipleTag video :

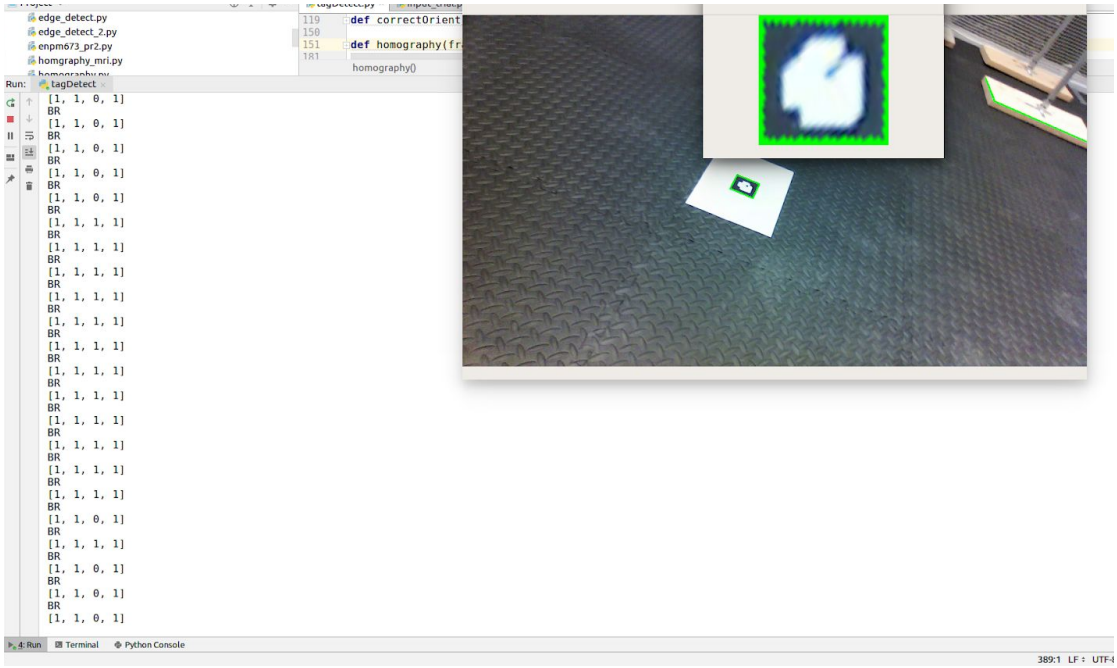
Introduction -

The aim of this project is to detect a custom AR Tag, that is used for obtaining a point of reference in the real world, like in augmented reality applications. The project is divided into 3 parts.
Part 1- It comprises of the detection of the encoded ID in each of the AR tags.
Part 2- It is a masking problem where the AR tag is to be masked with another reference image.
Part 3 - It is an AR application of placing a cube over the AR tag.
There are 4 test videos provided and a reference image for this problem.

Part 1 - ID detection

1. The image dimension is converted to 200 X 200, we defined the function tagDetect() that thresholds image into a BINARY_THRESHOLD.The image is assumed to be 8 X 8 grid and is then cropped to remove the padding.
2. The inner 4 X 4 image gives the orientation and the tag id.
3. The corner pixel of the outermost layer of 4 X 4 matrix gives the orientation of the image. Detecting white pixels as 1 and black pixels as 0.
4. The inner 2 X 2 matrix gives the Binary Code, which is read in a clockwise direction with the least significant bit at the top left.
   Below figure shows the detected tag

Figure : *Results of Tag Detection of a frame in Tag*



Part 2- Superimposing Image on TAG

1. Detected the corners of the tag using the edge detector function cv2.canny() and contour detection function cv2.findContours(). We have defined a function which uses the cv2.RETR_TREE, which generates the tree of contours. Using this tree, we have detected the parent contour of the ID contour. This parent contour will be the contour which encloses our AR Tag. The corners of this parent contour are used to form the homography matrix which transform the image from the world frame to the camera frame.
2. Depending on the orientation of the tag, we have corrected the orient of the lena image and superimposed on to the tag.
3. We have found the homography matrix that is needed to project the points from the world frame to the camera frame. We have defined the function called calculateHomograph() to construct the A matrix given as below

$$\boldsymbol{Ah} = \mathbf{0}_{8\times 1}$$

$$\begin{bmatrix} {x_1}^{(w)} & {y_1}^{(w)} & 1 & 0 & 0 & 0 & -{x_1}^{(c)}{x_1}^{(w)} & -{x_1}^{(c)}{y_1}^{(w)} & -{x_1}^{(c)} \\ 0 & 0 & 0 & {x_1}^{(w)} & {y_1}^{(w)} & 1 & -{y_1}^{(c)}{x_1}^{(w)} & -{y_1}^{(c)}{y_1}^{(w)} & -{y_1}^{(c)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ {x_4}^{(w)} & {y_4}^{(w)} & 1 & 0 & 0 & 0 & -{x_4}^{(c)}{x_4}^{(w)} & -{x_4}^{(c)}{y_4}^{(w)} & -{x_4}^{(c)} \\ 0 & 0 & 0 & {x_4}^{(w)} & {y_4}^{(w)} & 1 & -{y_4}^{(c)}{x_4}^{(w)} & -{y_4}^{(c)}{y_4}^{(w)} & -{y_4}^{(c)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The A matrix is then decomed using singular value decomposition technique as **USV.** where the last column of the V matrix gives the **h** vector given by

$$\boldsymbol{h} = \frac{\left[v_{19}, \ldots, v_{99}\right]^T}{v_{99}}.$$

The elements of this **h** matrix form the homography matrix.
This homography matrix is then used to superimpose the template image(lena.png) on to the tags. We have used the function cv2.warpPerspective() function to obtained the superimposed image based on the computed homography matrix.

Figure : *Results for Lena superimposed on a frame in Tag2*

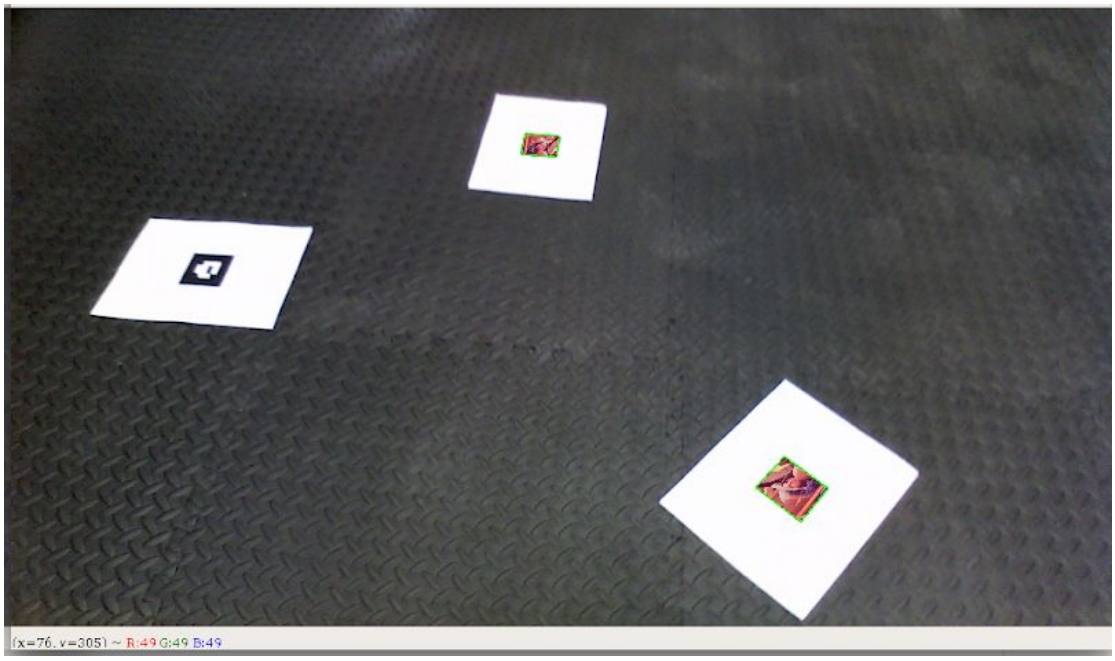


Figure *: Results for Lena superimposed on a frame in Multiple Tag*

Part 3- Cube Placement on the TAG

Steps involving the cube placement:

1. The inverse of the homography calculated above is used to calculate the new projection matrix to transform a 3D object to the image plane given by

$$P = K[R \mid t].$$

Where the K matrix is the Calibration matrix which is the intrinsic parameter of the camera.

2. From the homogeneous matrix the camera's pose is identified by the Rotation matrix **R** and the translation matrix **T**
3. The **R** and **T** matrices are used to define a new matrix **B** = [b1, b2, b3]. Where r1 = (scale parameter) * b1 , r2 =(scale parameter) * b2, r3 = r1 X r2 and t = b3. r1 , r2 and r3 will form the new rotation matrix **R** and new translational matrix **T** will form the new homography matrix **P** to transform 3D object to the 2D camera frame.
4. This has been accomplished by using the opencv function cv2.projectPoints(). This project the 3D points to the image plane. The parameters given to this function are cube coordinates in the world frame, rotational matrix, translational vector, the camera calibration matrix and the distortion coefficient which we have assumed as zero in our case.
5. We have used the draw function to draw contours and the edges of the cube.

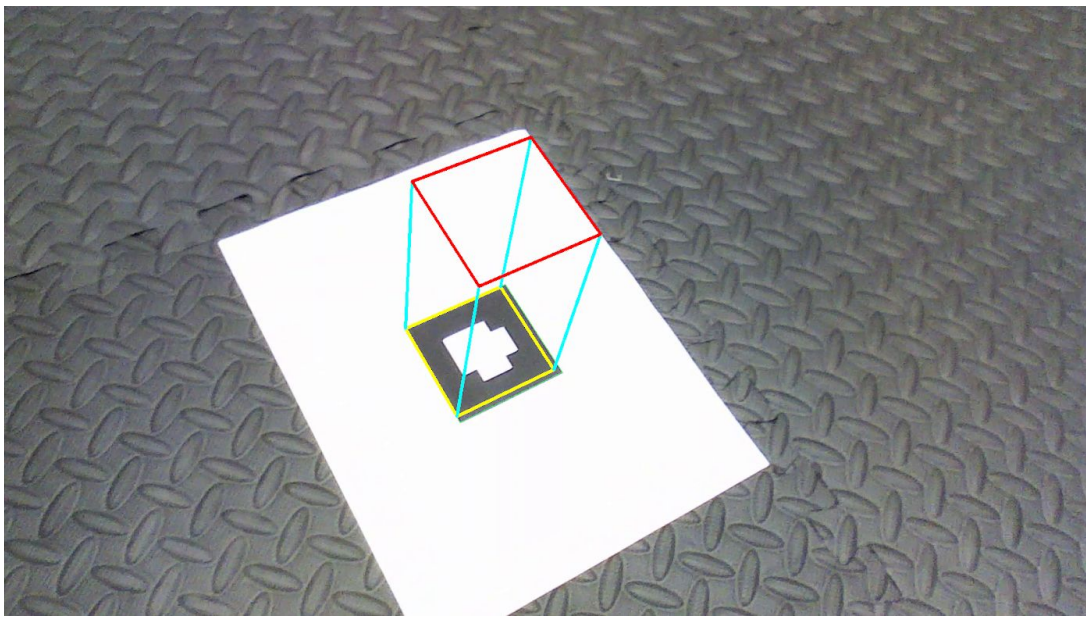


Figure : *Results for cube placement on a frame in Tag2*

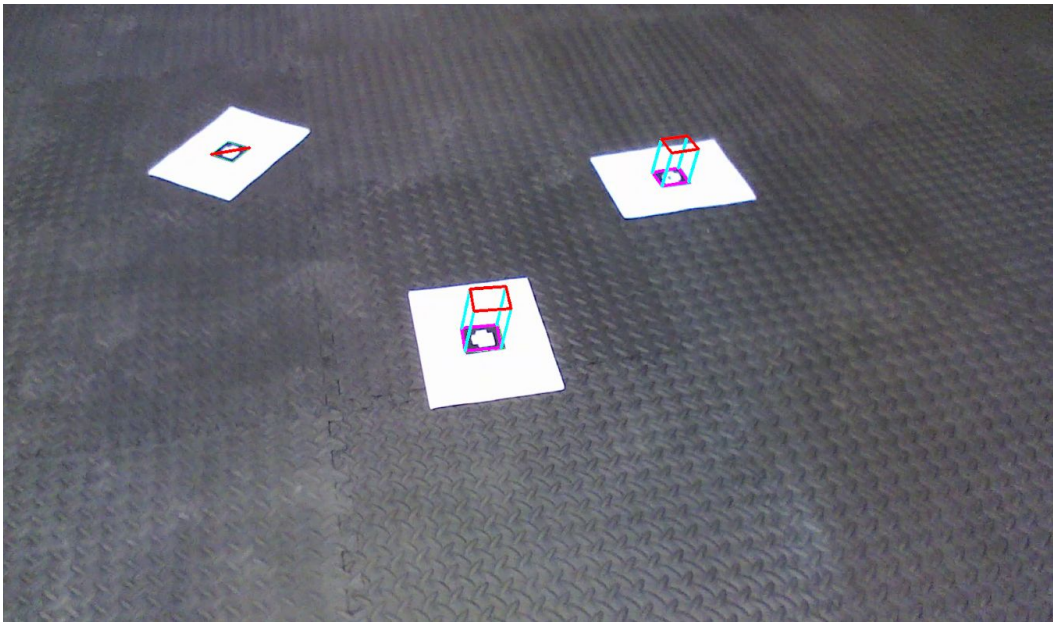


Figure : *Results for cube placement on a frame in multipleTag*