University Of Maryland

College Park

Project Report

673-Perception for Autonomous Robots

Project 1

Aditya Goswami
116699488

Mahmoud Dahmani
116699488

Sukoom Sarin
116699488

February 27, 2020

Contents

List of Figures

1 Introduction

This project emphasises on computer vision and augmented reality. Following points entail more information on this project.

- The major task given in this project is to detect a custom AR-Tag in different videos. Four videos are used as input and AR-Tags are detected from the videos.
- First the corners of the Tag are found which are then used to find the homography matrix.
- Homography matrix is then used to get the AR-Tag upfront using warping where a self-made function is used.
- The self-made function called *warp* is programmed using the logic that one can compute the homography matrix H with eight points with a matrix system such that the four corresponding points *(p1,p′1),(p2,p′2),(p3,p′3),(p4,p′4)* are written as 2×9 matrices such that SVD can be used on it. This gives the last singular vector of 9 elements which is given as *3x3 homography matrix*.
- Using the homography matrix, *warp* function, a new warped image is generated which gives the upfront render of the Tag.
- Then the decoding of the tag is done where the AR-Tag is used to get its binary value (tag id value) and its orientation.
- Using inverted homography, superimposition of Lena image is done on the tag.
- A function is created to produce the projection matrix.
- Projection matrix is further used to know how determination of point of world frame onto the tag in the frame.
- Later this is used in the space to draw the virtual cube.

2 AR-Tag Detection

In this phase the goal is to extract the tag from the video and then calculate required matrices for the superimposition of image and projection of 3D virtual cube.

- First the input is received in the form of a video.
- While iterating over the video, each frame is stored.
- Again iterating over each frame in the video, corners of the tag in the frame along with the contours and the threshold image of the frame is obtained by calling the self-made function `findCorners` which uses two functions to find the contours and the corners of the AR-Tag.
- The two functions used in the above steps are: `preprocess` and a built-in function from opencv `findContours`.
- The preprocess function processes individual frame image through operations like Grayscaling, blurring (using GaussianBlur) and the binary image using threshold function in opencv.
- `cv.findContours` function is given binary image as input which then generates contours and hierarchy of all the contours which is then manipulated to give corner coordinates of the quadrilateral. Thus giving us the contours, corners and binary image.
- Contours are simply curves joining all the continuous points (along the boundary), having same color or intensity. Here, in our code we have used the parameter `cv.CHAIN_APPROX_NONE` in the function `cv.findContours` that compresses horizontal, vertical, and diagonal segments and leaves only their end points. However, we get a set of points along the boundary.
- To get four corners of the tag we applied `cv.approxPolyDP` which selects the corners out of the multiple points generated. This gives us the corners of the black region of our AR-Tag.
- Thus we are able successfully detect the tag in the video (in each frame).

3 Converting Tag into Homography Matrix

After the dection of the AR-Tag, we need to extract the tag only as an upfront image that will show how the position of the tag or the orientation in each frame of the video (since the camera is rotating and we need to find the change in orientation of the Tag as well while the camera changes its orientation in the video).

For that first of all we need the homographical transformation of the image which is given by the homography matrix. Therefore following steps are taken to find the same.

1. While detecting the Tag, we are able to get corners of the Tag which are then shown as circle in the video using the `cv.circle`.

2. Now that we have all the corners, iterate over each corner and to find the homography matrix for each corner using the self-made function (`estimate Homography`).

3. In the function `estimateHomography`, four points from the source image are taken. Similarly four points are taken from the destination frame.

4. Theory: You can compute the homography matrix H with your eight points with a matrix system such that the four correspondance points $(p1, p'1), (p2, p'2), (p3, p'3), (p4, p'4)$ are written as 2×9 matrices such as:

$$p_i = \begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x'_i & y_i x'_i & x'_i \\ 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y'_i & y_i y'_i & y'_i \end{bmatrix}$$

5. It is then possible to stack them into a matrix P to compute:
   $PH = 0$
   Such as:

$$PH = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1 x'_1 & y_1 x'_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1 y'_1 & y_1 y'_1 & y'_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2 x'_2 & y_2 x'_2 & x'_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2 y'_2 & y_2 y'_2 & y'_2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3 x'_3 & y_3 x'_3 & x'_3 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3 y'_3 & y_3 y'_3 & y'_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4 x'_4 & y_4 x'_4 & x'_4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4 y'_4 & y_4 y'_4 & y'_4 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0 \tag{1}$$

6. While adding an extra constraint $|H| = 1$ to avoid the obvious solution of H being all zeros. It is easy to use $SVDP = USV^\top$ and select the last singular vector of V as the solution to H.

Note that this gives you a $DLT(directlineartransform)$ homography that minimizes algebraic error. This error is not geometrically meaningful and so the computed homography may not be as good as you expect. One typically applies nonlinear least squares with a better cost function (e.g. symmetric transfer error) to improve the homography.

Once you have your homography matrix H, you can compute the projected coordinates of any point $p(x, y)$ such as:

$$\begin{bmatrix} x'/\lambda \\ y'/\lambda \\ \lambda \end{bmatrix} = \begin{bmatrix} h_1 1 & h_1 2 & h_1 3 \\ h_2 1 & h_2 2 & h_2 3 \\ h_3 1 & h_3 2 & h_3 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2}$$

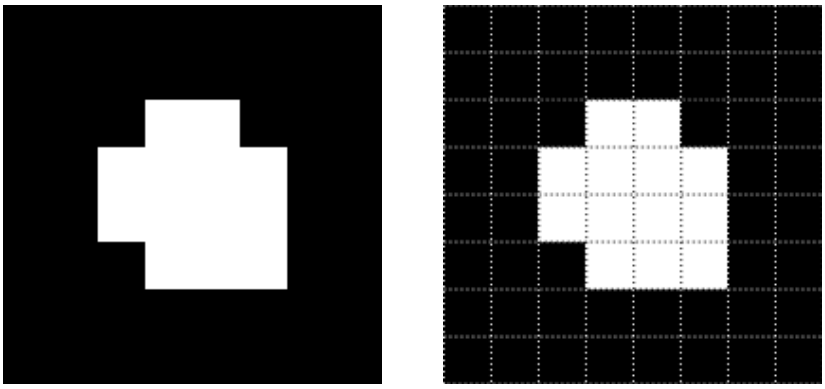
7. In the code, the P matrix is created and its SVD singular value decomposition is done using the inbuilt function `(np.linalg.svd`. This gives us the homography matrix.

Note:

- Keep in mind about reshape function such that it returns the value of the SVD in $3x3$ matrix.

4 Decoding the tag

1. After using homography matrix, we are able to extract the Tag from the video i.e. Tag-upfront image.
2. The extraction is finally done with the help of warping.
3. After finding homography from source coordinates to warped coordinates, we perform warping by multiplying the source coordinates to the obtained homography matrix.
4. We observed that sometimes due to scaling of the 2 image coordinates, our warped image develops some blank pixels.
5. This happened because after multiplication the obtained coordinates were rounded to the nearest integer and if the warped image size was larger than source image not all pixels were covered, leaving blank pixels.
6. We solved this problem by multiplying the warped image coordinates by the inverse of the homography matrix instead. This was done to find which coordinate of the warped image matches with which coordinate of the source image, and then using the pixel value from the source image and pasting it back on the warped image. This way we ensure that all the pixels of the warped image are covered.



(a) Reference Marker (b) Reference Marker Grid

Figure 1: AR-Tag - Divided in 64 squares

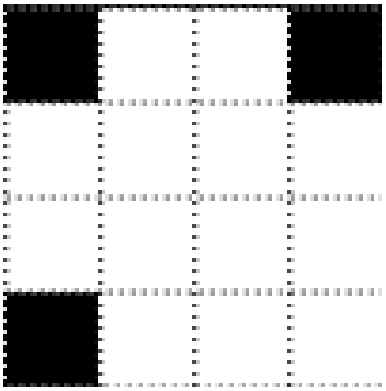


Figure 2: 4x4 Grid Tag

7. Once we have the upfront images of the Tag, we divide it into 64 equal squares.

8. We further convert it into a 4x4 matrix which gives us the internal white area with some black squares at some corners by determining if the squares has white pixels or black pixels.

9. Here 1 represent white colour and 0 represents black colour.

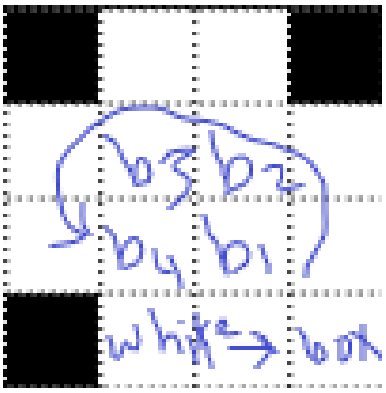


Figure 3: 4x4 Grid Tag

5 Finding the orientation and Binary Value

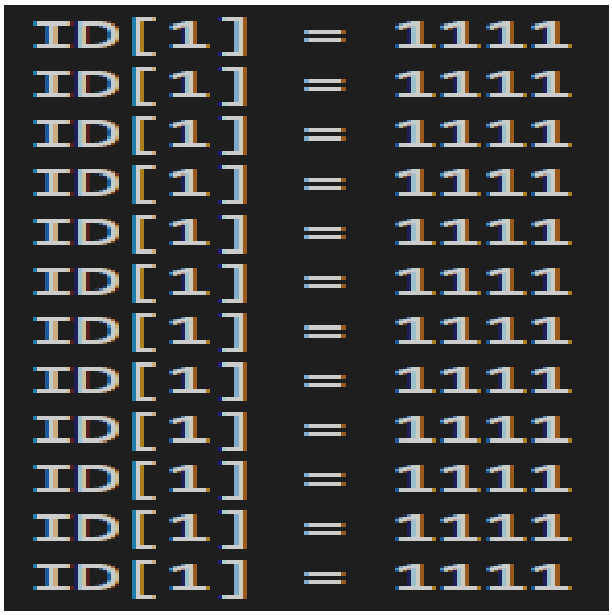
1. Once we have the 4x4 matrix, we try to find the position of the white box present at the bottom right corner which gives us the orientation of the Tag.

2. The position of the bottom right corner of 4x4 matrix helps us to determine how much the tag has been rotated.

3. When the bottom right corner of 4x4 matrix has been identified then the internal white area (2x2 matrix at the center) gives us the binary form of the id (as in white colour is 1 and black colour is 1).

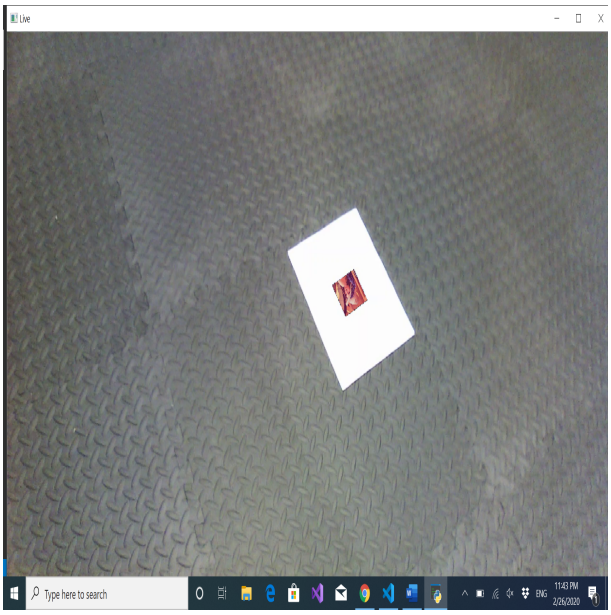
4. The nearest square in 2x2 matrix to the white box in 8x8 matrix is the starting bit and then all the bits are read in anti-clockwise.

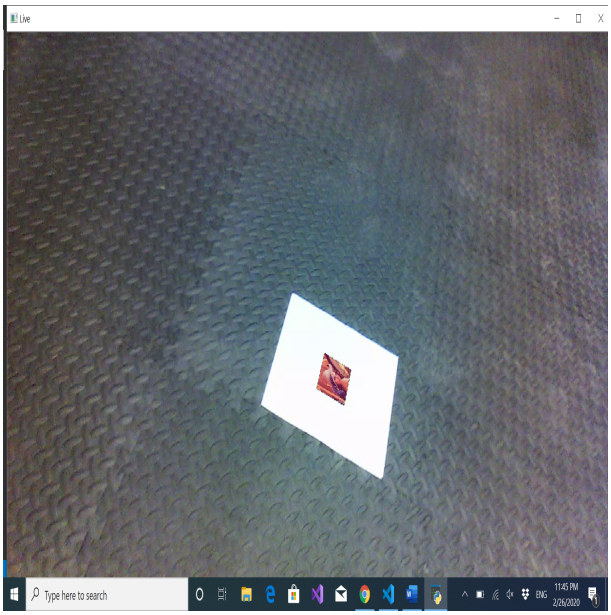
5. The method is shown above in figure 3: 4x4 Grid Tag.

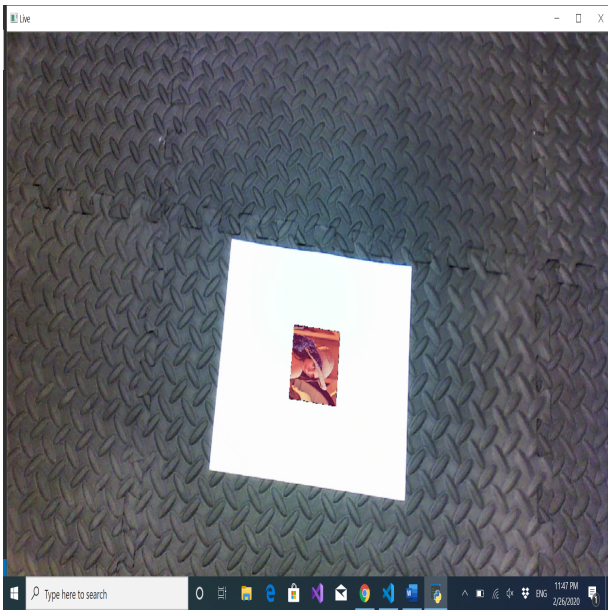

ID[1] = 1111
ID[1] = 1111
ID[1] = 1111
ID[1] = 1111
ID[1] = 1111
ID[1] = 1111
ID[1] = 1111
ID[1] = 1111
ID[1] = 1111
ID[1] = 1111
ID[1] = 1111
ID[1] = 1111

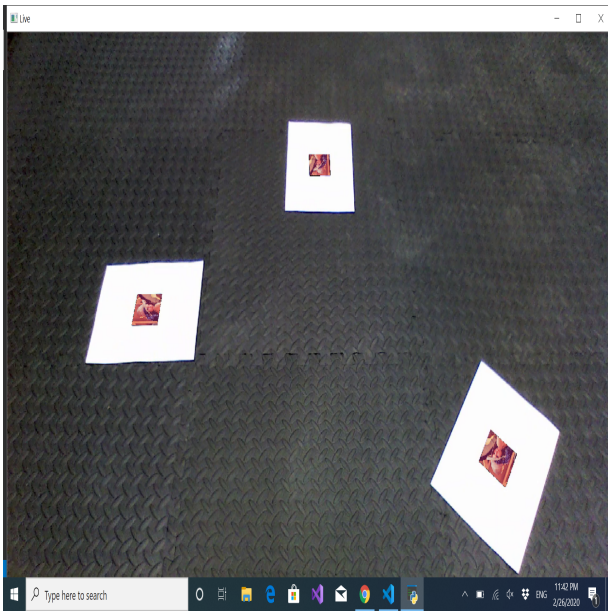
6 Superimposing Lena on the tag

1. The corners of the AR tag in the videos from section-1 are used in this section.
2. The corners are updated according to the orientation of the AR-Tag.
3. A homography function is defined to `estimate the homographic` matrix between the given template, image of Lena(world frame).
4. When the self-made homography function is called it takes corners and the corners of the template are given as the input.
5. Then a new image (frame1) of the frame is created in which the warped image is stored as the self-made function of warping used earlier is called.
6. Again a new image is created where self-made function `renderMask` in which frame and contours are passed.
7. This gives us an image (frame2) of the frame where the area of whole Tag is rendered black.
8. Now the opencv function `bitwise_or` is called where both images (frame1 and frame2) are called and the new image is stored (in frame).
9. Finally, this new image (frame) has the image of Lena superimposed onto the Tag.




Live
Type here to search
ENG
11:45 PM
2/26/2020


Live
Type here to search
ENG
11:47 PM
2/26/2020



7 Superimposing a virtual cube on the tag

1. The objective is to place a virtual cube on the AR tags in the video by the process of projecting a 3-D shape on to a 2-D image.

2. The cube consists of 8 corners and 12 corresponding edges joining them. Out of the 8 corners and 12 edges, 4 corners and edges lie on AR tag. These four corner points can be written as (0,0,0), (0,200,0),(200,0,0) and (200,200,0) in the world frame.

3. As the positive Z axis is assumed in the downward direction the other four corners could be considered as (0, 0, -200), (0, 200, -200), (200, 0, -200) and (200, 200, -200).

4. First, the homography between the AR tag from the video in the camera frame and the world frame is calculated. The corners of the AR tag are considered as camera frame and the corners (0, 0, 0), (0, 200, 0), (200, 0, 0) and (200, 200, 0) are considered as the world frame. In order to calculate homography a user-de[U+FB01]ned function homography is called.

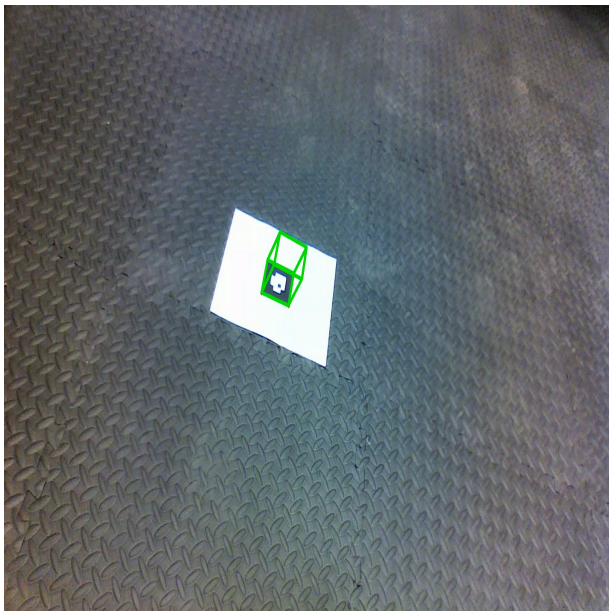
5. Secondly, in order to project 3-D points from world frame to camera frame, we require a projection matrix. For calculating projection matrix, a user de[U+FB01]ned function projectionMatrix is created, which consists all necessary mathematical equations. projectionMatrix requires camera calibration matrix and homography matrix as its two arguments.

6. Again a new image is created where self-made function `renderMask` in which frame and contours are passed.

7. Then the 2-D camera coordinates are calculated using 3x4 projection matrix and 3-D world coordinates [(0,0,0), (0,200,0), (200,0,0), (200,200,0), (0, 0,-200), (0, 200,-200), (200, 0,-200) and (200, 200,-200)].

8. The formulas and equations required for de`[U+FB01]`ning the homography and projectionMatrix functions are referred from the supplementary provided.

9. Finally, the 3D virtual cube is placed on the Tag.



8 Results and conclusion

1. All the three parts of question statement have been achieved.
2. We faced problems for finding the real corners but we were successful after checking several approaches.
3. We faced loss of information during the use of homography and warp function which we overcame by using the inversion of homography.
4. We came across many challenges while projecting 3D cube due to some hyper parameter. But finally we solved most of them later by tuning those hyper parameters, except jittering of cubes.
5. We also noticed that without using the in built warp perspective function the frames are quite slow to be processed since every single pixel has to be multiplied.