ENPM 673 - Perception for autonomous Robots
ARtag detection, Image Superimposition & 3D Cube Projection

Submitted towards completion of Project-1

Abhishek Banerjee
Arjun Gupta
Vishnuu AD

February 26^{th} 2020

1 Introduction

In this project we seek to detect sets of AR tags, either singular or in groups from an input video. The idea is to detect and then identify the tags subsequently on the basis of the data that they encode. Thereafter, we proceed to superimpose an image (lena.png) onto the tag(s) using the calculated homography. The final part of the project consists of placing a 3D cube on the tag(s). In this report we detail out the steps and our approach involved in completing each stage of the project.

2 Image Pipeline

The pipeline for each task has three components which is described as follows:

- **Video read :** OpenCV's Video capture ***obj = cv2.VideoCapture()*** library is used create a Video reader Object. This object is used to capture individual frames and store them in the buffer using ***obj.read()*** .
- **Image Process** Once the frame is available, our first task in any of the challenges below is to detect the corners of the tag. Once the corners are available, we estimate homography between the desired (rectified view) of the tag and the corners of the tag found from corner detection module. The homography is then used differently in each of the challenges below.
  - **Corner detection**: The Corner detection module returns a list of tag's corners. This list is iterated and for each tag, homography is estimated and subsequent tasks are performed.
  - **Homography Estimation**: Given 4 pairs of points i.e 4 from the tag in the given image and 4 in from the desired image, we can estimate Homography by solving $A * x = 0$, where x is a coloumn vector which are also the elements of Homography H.
  - **Task Specific Process**: There are three challenges:
    * Find the orientation and ID of the Tag: We first rectify the ARtag by using our equivalent of OpenCV's WarpPerspective, called getPerspectiveTransform. We then find the corners of the 4x4 grid. To do this, we've found the coordinates x_{min}, x_{max} and y_{min}, y_{max} of the white portion of the tag. The 4 points formed by these limits

are the corners of our 4x4 grid rather than just assuming that the corners of the 4x4 grid are known given the rectified image. This is because the white pattern in the image tends to distort when the camera moves with a high frequency. These corners points are then used as reference to find the orientation and ID of the tag.

  * Superimpose Lena on the tag: Having found the Homography matrix, we first find the orientation of the tag. Having found the orientation of the tag, we rotate our reference image(Lena image) as many times in the opposite sense so that when warped, we get the right orientation on our actual frame. To warp a rectified image on top of the tag, we use a slightly modified version of the getPerspectiveTransform known as getPerspectiveTransform_Lena in our code. After finding the corresponding points of Lena on the tag, we modify the original image with Lena of top of the tag.
  * 3D Cube projection: Given the homography matrix, we find the projection matrix and then thereby find the corresponding points of the entire cube on the image. Small circles and lines are drawn using basic OpenCV functions and hence the cube is visualised.

- **Write to File**: We use the OpenCV library **obj = cv2.VideoWriter()** to create a writer object. The method **obj.write()** is used to write frames into a file.

3 Task : Detection, Identification and Superposition

3.1 Corner Detection

The first step in our task is detect the AR tags from a video stream and determine it's identity through the information it encodes. The most common ways of determining the corners of an image is through the use of filters that extract the gradients in the image, and an intersection of these gradient features, which are nothing but lines, leads to a corner. Common algorithms in this domain are Shi-Tomasi and Harris corner detectors. In our current study, although the corner detectors work very well on account of the simplicity of the image, it becomes difficult to establish a relation between the detected corners and the ones that are there in the image for the AR tag.

Alternatively, we used the cv2.findContours() function to first extract the contours that exist in the image. Before this step we apply a Gaussian filter on the image to blur the image that enhances our corner detection task. This function gives us a set of contours, which are bunches of points which correspond to a single continuous feature, in this case a contour. Additionally the function gives us an hierarchical relation between the contours. We leverage this relationship to detect the corner points of our AR tag.

The hierarchy for a specific contour gives us information about it's "parent" and "child". In this context, if there exists a contour which is completely enclosed by the current contour, then the enclosed one becomes the "child" of the enclosing contour. Conversely, the circumscribing contour becomes the "parent" of the inner one. It is this property which aids us in extracting the AR tag for our purpose. The AR tag is the parent of the feature inside of it which encodes information, and hence gives it a unique ID. The AR tag is further a child of the contour formed by the sheet of paper on which it is printed. Hence, we filter out all contours that do not have both a parent and a child, i.e., we get a contour which is nested.

The contours which we get out of this step, consist of all the points that constitute the contour.

To extract the four points that form the contour, we calculate a convex hull of all points in that contour. A convex hull is a convex space which encloses all the points that are a part of it, by selecting a subset of points. We check for the extremum points of each convex hull, and pass them on. These points, in our case, come out to be the corner points of the AR tag.

Therefore, we detect the tag corners in the first step.

3.2 Homography

After we detect the corners of tag(s) in the image we want to find the ID associated with the tag. In order to find the ID we first must orient the tag in the specified manner to correctly calculate the tag ID. Thus, in order to do so we first transform the tag from image to a rectified image. This is done using the homography. We calculate the homography as described above between the pixel coordinates of the corners in the image and the rectified corners of the tag. The rectified image is 128×128 in size with it's center at the camera center. The homography being a similarity transformation, gives us the rotation and translation to get from one image orientation to the other, on account of the fact that it encodes a relation between corresponding points between the pair of images.

3.3 Tag ID and Orientation

The next step is to determine the IDs of the AR tags that are now in the camera frame. The first step is to iterate over the image, column and row-wise to determine the corners of the white-patch of the tag which encodes the information. Once we have the maximum and the minimum corner points for the central patch, we discretize it into a patch of size 4 x 4. The inner 2 x 2 patch designates the ID of the image, while the immediate padding of thickness 1 is what is used for determining the orientation of the image. In it's standard form, the bottom right corner which has a value of 255 is said to be the upright position. Depending on where that patch is, we apply rotations to the image to get it into the desired orientation, i.e., bottom right. Thereafter we look at the central 2 x 2 patch and perform a bit-wise binary OR operation, to determine the IDs.

3.4 Image Superposition

To superimpose a given image, in our case, 'lena.png', we first determine the corners of our AR tag. Considering that our image is of size 512 x 512, we resize it to 256 x 256, where the corners have coordinates, (0,0),(0,256),(256,0) and (256,256). With both the set of corners known, we calculate the homography the same way as has been detailed in the previous sections, and apply an inverse transformation to put the image on top of the AR tag. One point to note, is that before we apply the inverse Homography and warp transformation, we need to determine the orientation of the AR tag, so that the image can be superimposed in the right orientation.

Since we cannot use the in built function (cv2.warpPerspective()), we built our own function to achieve the same task. The idea is to warp Image A to Image B using the computed homography between the two image planes. One of the possible ways is to calculate the homography from A to B and use it to warp image A to image B. Since the values of the homography matrix can be of float (decimal point) type, we end up getting transformed values (after matrix multiplication) as the float values. When we round the values to nearest integer and assign it to the obtained

coordinates, the ultimate obtained image has lot of blank spots. In order to overcome this issue, we use the inverse warping. We assume a grid of the same size as that of Image A and then map every element of this grid to Image A by using the inverse of the homography matrix. Now, when the transformed values are in float we take the average of the neighboring pixel's intensities and assign the averaged value to the pixel in Image B. This way we map every pixel in image B and have no holes in the image.

4 Task : 3D cube projection

4.1 Projective matrix calculation

For projecting a 3D point from world coordinates to image we use the following projection equation:

$$\begin{bmatrix} \text{x} \\ \text{y} \\ 1 \end{bmatrix} = P * \begin{bmatrix} \text{X} \\ \text{Y} \\ \text{Z} \\ 1 \end{bmatrix}$$

where X, Y and Z are the world coordinates, x and y are the corresponding image plane coordinates and P (3X4) is the product of intrinsic and extrinsic matrix of the camera. Since we want to find the projection of the points of cube on the image plane we first solve for the bottom four points of the cube and we assume Z=0 for those four points. Thus, by simple arithmetic we can observe that we can remove the third column of P matrix and the remaining P matrix is equivalent to somewhat we see in Homography matrix. In this representation we represent the rotation matrix simply by r_1, r_2 and the reduced extrinsic matrix of camera is given by $[r_1, r_2, t]$. Thus we can define the P in above equation as H = K$[r_1, r_2, t]$ where K is the intrinsic matrix of the camera. We solve the obtained equation for H in the same way as we did in previous parts.

After we obtain the homography matrix, we want to calculate the top four points of the cube for which we assume the Z=-512. Using the obtained H matrix we obtain the full rotation matrix and translation vector of the extrinsic matrix as follows:

$$\lambda H = K\tilde{B}$$

$$\tilde{B} = \lambda K^{-1} H$$

$$B = \lambda \tilde{B}(-1)^{|\tilde{B}|<0}$$

where $\tilde{B}$ is given by $[r_1, r_2, t]$, λ is the scale factor that needs to explicitly computed. The power term in the last equation tells that consider negative of $\tilde{B}$ if its determinant is negative. r_1, r_2 of the extrinsic matrix are proportional to that of $\tilde{B}$ and we calculate r_3 by calculating the cross product of r_1 and r_2. In real sense the column vectors of the rotation matrix must be of the unit length but the estimated vectors are usually not of unit length. Thus we estimate the λ parameter by taking average of the first two columns of B i.e.:

$$\lambda = \left(\frac{\left|\left|K^{-1}h_1\right|\right| + \left|\left|K^{-1}h_2\right|\right|}{2} \right)^{-1} \tag{1}$$

Now we can calculate all the elements of the extrinsic matrix as:

$$r_3 = \lambda(r_1 \times r_2), \quad r_1 = \lambda r_1, \quad r_2 = \lambda r_2, \quad and \quad t = \lambda t$$

we utilize the above obtained information to calculate the projection of top four points of the cube on image plane simply by solving:

$$\begin{bmatrix} \text{x} \\ \text{y} \\ 1 \end{bmatrix} = K * [r_1, r_2, r_3, t] * \begin{bmatrix} \text{X} \\ \text{Y} \\ \text{-512} \\ 1 \end{bmatrix}$$

5 Results

Here below we show sample results of a particular tag given in the data set : **Tag 1** and also of **multiple tags**. The complete output videos can be found in the link here. and the code can be found here.

Tag 1:

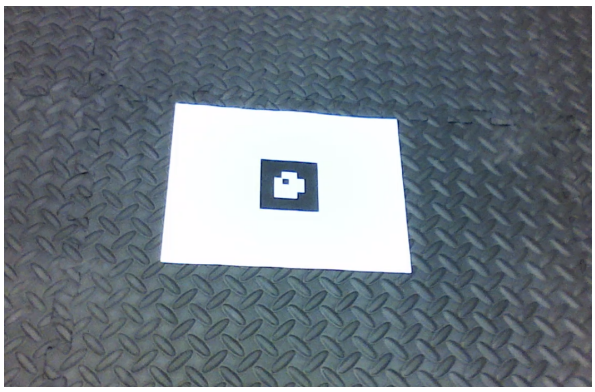

Figure 1: Raw Image with single tag

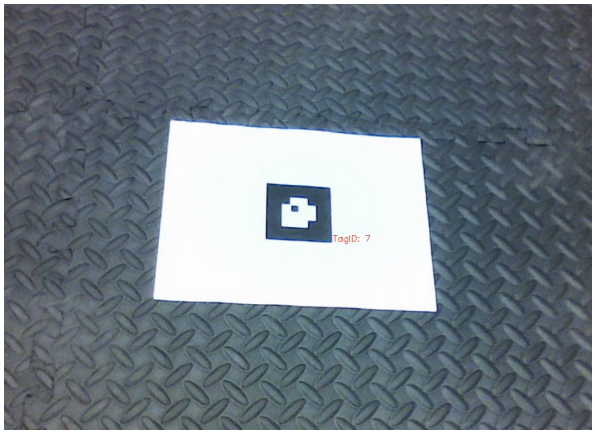

Figure 2: Image with single tag and ID

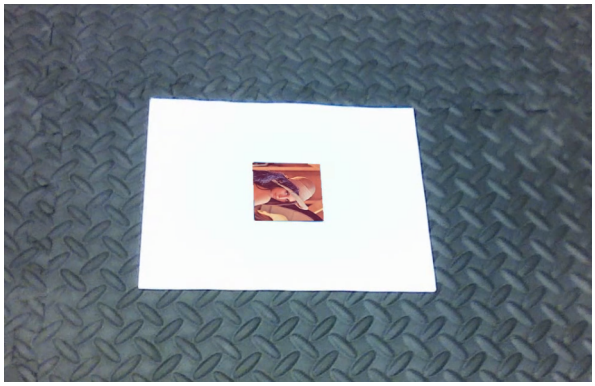

Figure 3: Image with single tag and Lena image superimposed over the tag

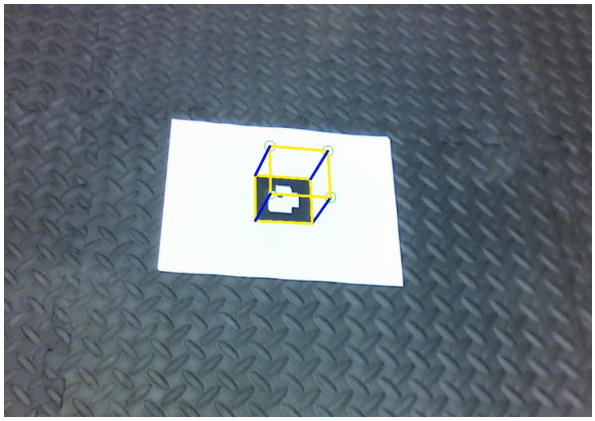

Figure 4: Image with single tag and cube projected over the tag

Multiple Tags

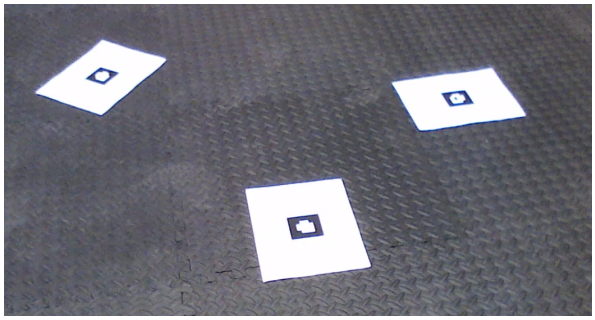

Figure 5: Raw Image with multiple tags

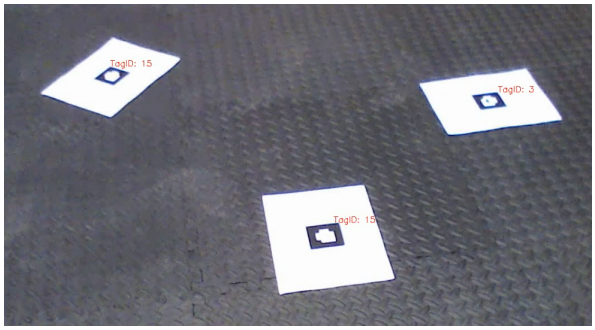

Figure 6: Image with multiple tags and corresponding IDs

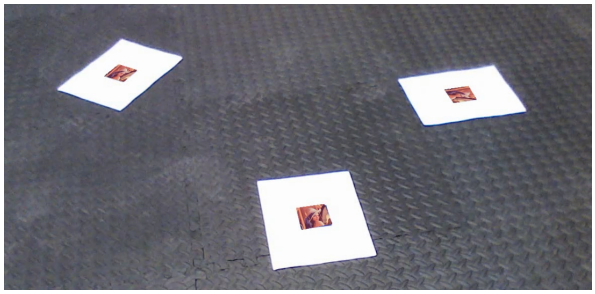

Figure 7: Image with multiple tags and Lena image superimposed over the tags

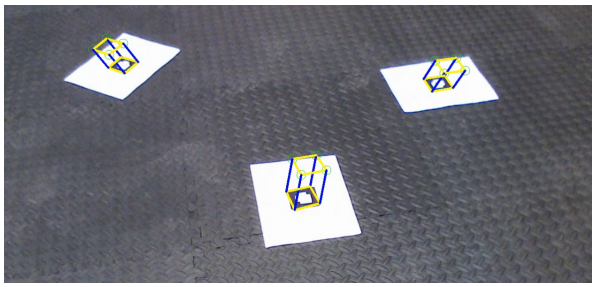

Figure 8: Image with multiple tags and cube projected over the tags