ENPM673

Balaji Selvakumar – 118545745

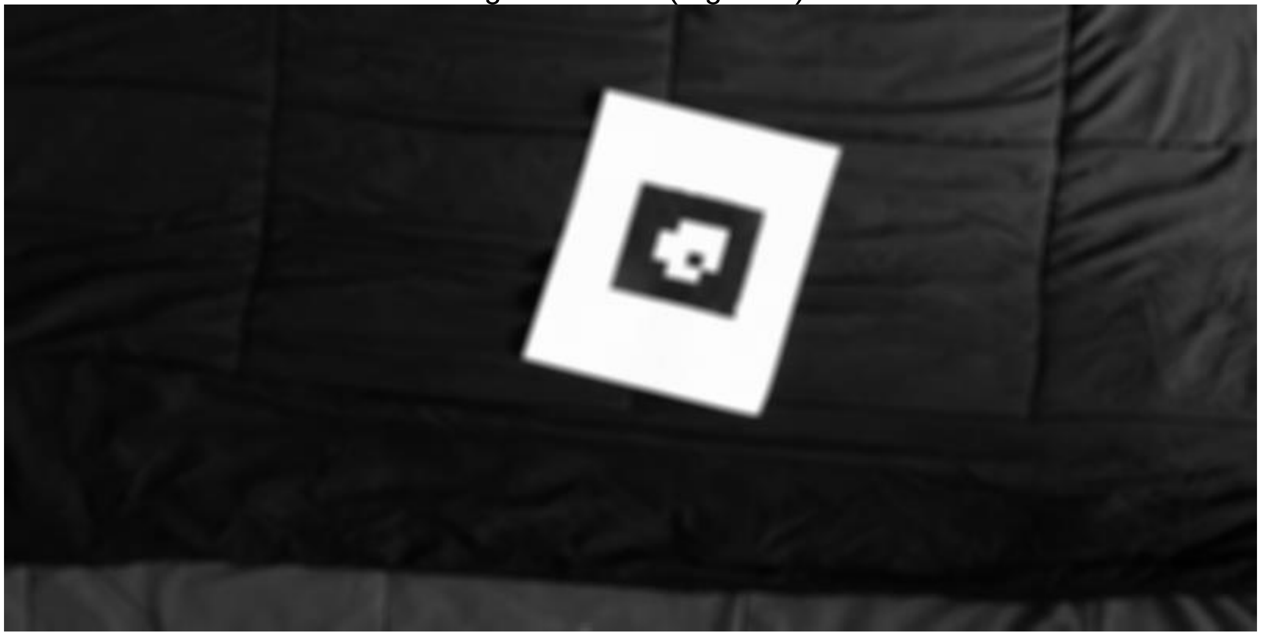
ABSTRACT

In this project we focus on detecting and tracking a custom AR Tag in a video sequence. The detection part of the project focused on finding the location and identity of the AR Tag. The tracking part of the project involved superimposing an image and placing a 3D cube over the tag. For improving the estimated homograph, Harris corner detection was applied to keep a track of four corners of the AR Tag

1. PART 1 – DETECTION

1.1 Initial Filtering

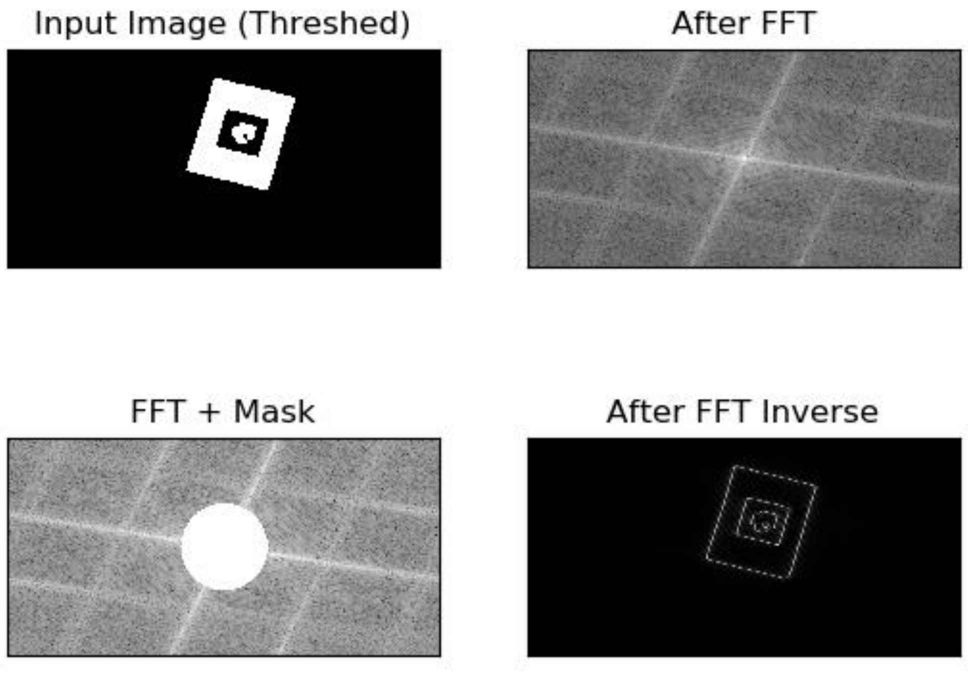
The video was evaluated frame by frame, treating each frame as an individual static image. Converted the frame to a gray-scale color space. To make the detection smoother, the gray-scale image was then blurred out to remove the unwanted noise and to just focus on the edges. Gray-scale threshold was experimentally determined that made just the AR tag clearly visible in the frame and eliminated all undesired image elements (Figure 2).



1.2 Fast Fourier Transformation and Edge Segmentation

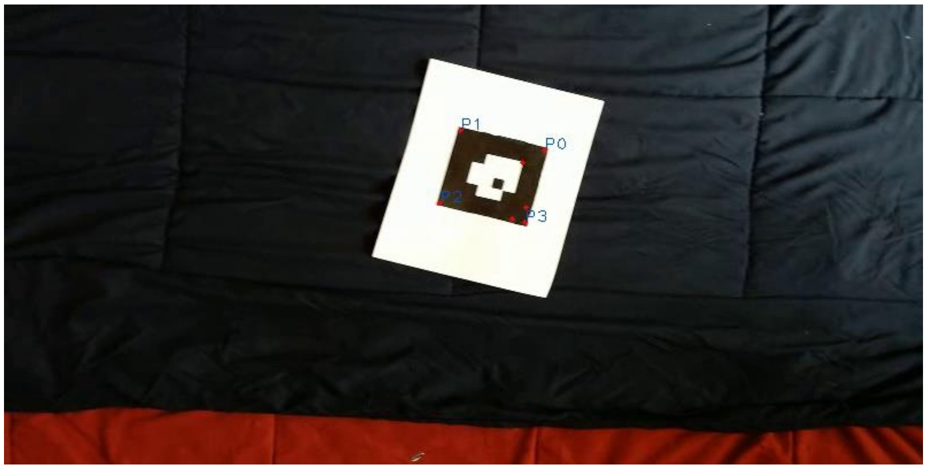
The edges of an image are high frequency signals. If we could filter out the low frequency signals from an image, we can obtain the edges. Inbuilt FFT is used to calculate FFT and Inverse FFT. A masking (High pass filter) is passed to the fourier transformation and the resultant output is converted back to spacial domain using Inverse FFT function. The following steps were followed to do the FFT and detect the edges:

1) Compute FFT for a gray-scale(blurred then threshed) image.
2) Define a circular mask(High Pass Filter) and multiply it with the FFT from step 1. to filter out low-frequency signals.
3) Compute the inverse of FFT to get the edges.



1.3 Getting Corners

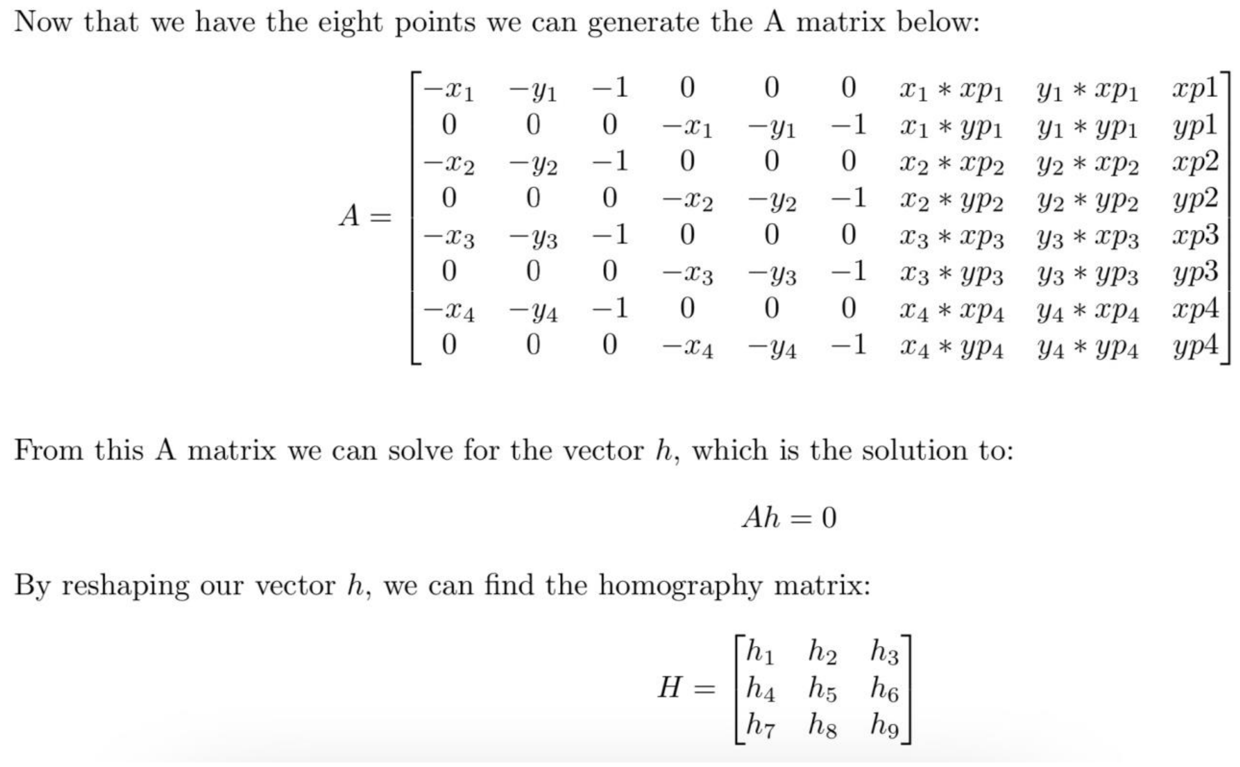
Once the edges are detected and all the noises are avoided using the above functions, I applied Harris Corner detection to detect the corners of the video/frame. But this function gives all the points/ Edges from the video. Since I just need the four corners of the AR Tag, I created a logic for getting the extreme points from the output of Inverse FFT video(Since it’s a binary video it has only two values)



2.Decoding AR Tag

2.1 Homography Computation:

To find a homography matrix that will allow us to convert between our camera coordinates and the square coordinates we need eight points. Four of the points are the corners of the tag that we have from detection. The other four points are the corners of our square destination image. The dimensions of the destination are somewhat arbitrary but if it is too large we will start to have holes, since there are not enough pixels in the source image, and if it is too small we will have a lower resolution and will start to lose information. To fix this we found the number of pixels that were contained within the four corners of the tag and took the square root of that number to get an ideal dimension dim. The four corners are then [(0,0), (0, dim), (dim, dim), (dim, 0)



Now that we have the eight points we can generate the A matrix below:

$$A = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1 * xp_1 & y_1 * xp_1 & xp1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1 * yp_1 & y_1 * yp_1 & yp1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2 * xp_2 & y_2 * xp_2 & xp2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2 * yp_2 & y_2 * yp_2 & yp2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3 * xp_3 & y_3 * xp_3 & xp3 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3 * yp_3 & y_3 * yp_3 & yp3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4 * xp_4 & y_4 * xp_4 & xp4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4 * yp_4 & y_4 * yp_4 & yp4 \end{bmatrix}$$

From this A matrix we can solve for the vector h, which is the solution to:

$$Ah = 0$$

By reshaping our vector h, we can find the homography matrix:

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

To solve for the vector h, I used Singular Value Decomposition .This will decompose A matrix into three matrices U, Σ,V. Inbuilt numpy function is used to calculate SVD and the corresponding homography for the four corners were found.

2.2 Creating the Square Image

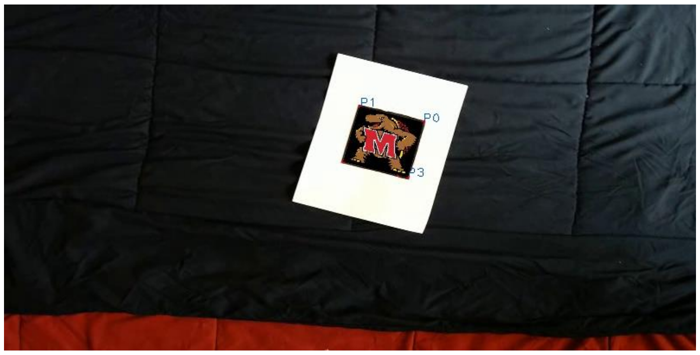
After the homography matrix has been computed,we need to generate a new set of points in the square frame, and then create a new image using the pixels from source image in their new location. For any point in the camera coordinates Xc (x,y), we can compute that point in the square coordinates Xs (x′,y′)

We can generate new points for all (x,y) in a given frame. The points generated are floats with multiple decimal places. There are several possible approaches to remedy this, including averaging the pixel values around the closest four pixels, picking the closest neighbor, or even simply truncating the floating point value as an integer. We decided to use this last approach for its simplicity and because the grid size of the tags is so large compared to any pixel that it this loss of resolution does not affect the decoding of the tag.
It is also important to note that any points outside of the boundaries of the corners of the tag will be outside of the dimensions of our destination image. We can generate a new image by taking only the points that lie in the range of our destination image and reconstruct an image pixel by pixel in the new coordinates

2.3 Warping:

After finding homography from source coordinates to warped(destination) coordinates, we perform warping by multiplying the source coordinates to the obtained homography matrix. It's been observed that sometimes due to scaling of the 2 image coordinates, warped image develops some blank pixels. 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.
This problem was solved by multiplying the warped image coordinates by 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, Thus pixel is left without empty.

3. Superposing Testudo Image
To place the Testudo image on top of the tag, we recalculate Homography matrix by using Testudo corners as source coordinates and the detected AR tag corners as destination coordinates. We apply this Homography on the Testudo image to obtain transformed coordinates. To superimpose Testudo on our original image, we overwrite the pixel values of our original image at the transformed pixels with the corresponding Testudo pixel values.



After we have determined the position, orientation, and ID of the tag in a given frame, the Testudo can be superimposed onto the tag. The steps to do this are as follows:

1). Match the tag ID to an image.
The IDs of the tag and Testudo photo is hard coded at the start of the program.

2). Rotate the image
The function we use to decode a tag returns a number of rotations necessary to match the rotation of the tag. This number is either 0,1,2,3. We can then apply either no rotation or a multiple of 90à to the

Testudo image using the function cv2.rotate()
(a) Upright Image (b) Right turned image

3). Use homography to generate a new image

Use the same process as above to generate a warped version of our Testudo image. In this case however instead of generating a square image of arbitrary size, instead create an image that is the same size as the original frame. When we generate our new homography matrix we need to use the corners of the frame in addition to the corners of the tag. We can then follow the same process as in equation: to generate a new image that is the same size as our frame but has a warped image of the dog and is black everywhere outside of the frame (Figure 8a).

4). Blank the original region of the tag

In order to perform the next step we need to blank the region of the image where the tag is. We can achieve this by drawing a filled black contour in the region defined by the corners of the tag.

5). Add the new image to the existing frame

With the image generated from homography and the original frame with the blank region we can add them together after that. Since the first image is black (0) everywhere except the tag and the second image is black (0) only in the tag region, the output is the Testudo superposed onto the tag

Placing a virtual cube onto tag:

As the cube is a three dimensional entity, we need a 3 × 4 projection matrix to project 3D points into the image plane. The basic assumption while calculating the homography matrix is that the points are planar, i.e. z = 0 for all the points. Now, we have z values as well. Thus we need to modify this H matrix to suit our requirements.

The coordinates of the corners of our cube, in the world frame, are:

$$\text{corners}_{c} = \begin{bmatrix} x_1, & y_1, & 0 \\ x_2, & y_2, & 0 \\ x_3, & y_3, & 0 \\ x_4, & y_4, & 0 \\ x_1, & y_1, & -d \\ x_2, & y_2, & -d \\ x_3, & y_3, & -d \\ x_4, & y_4, & -d \end{bmatrix}$$

where d is the height of the cube. The base of the cube is flush with the floor plane of the world frame, so the first four points have zero $\hat{Z}$ component. The last four points bound the top face of the cube, and exist d away from the floor plane. Our task is to find these points in the image frame so that they can be displayed in the image.

To steps to obtain a projection matrix(P) from a homography matrix(H) and camera calibration matrix(K) are as follows:

- Lets write the homography matrix as $H = K\tilde{B}$.
- Then, $\tilde{B} = K^{-1} H$
- Obtain matrix B such that $B = \lambda\tilde{B}$
- Compute mod of $\tilde{B}$. if it is negative, then $\tilde{B} = -1 * \tilde{B}$, where b1 and b2
- We will calculate λ as $\lambda = kb1k+kb2k$
- Therefore, $B = [\lambda b1, \lambda b2, \lambda b3]$
- B can be decomposed and written as [r1, r2, t], where r1 and r2 are rotational components and t is translational component.
- We know that the rotation matrix is orthonormal; the last column must be perpendicular to the first two[2]. Hence, $r3 = r1 \times r2$
- The final projection matrix can be written as $P = K[r1, r2, r3, t]$.

Once we obtain the projection matrix, we can multiply the homogeneous 3D co-ordinates of the cube and obtain its equivalent image co-ordinates.