ENPM 673 - Project 1

Nishad Kulkarni
A. James Clark School of Engineering
University of Maryland, College Park
UID - 117555431, Email: nkulkar2@umd.edu

I. Introduction

This Project required us, the students to detect an April Tag, and perform various operations on that. There were 2 main questions divided into 2 subparts each. The parts are named 1a, 1b, 2a, and 2b. They will be referenced by the same convention here onwards throughout this report. The code that accompanies this report uses Python (3.8.10) and the libraries numpy (1.22.2), matplotlib(3.5.1), and opencv (cv2, 4.5.5).

II. Question 1

The task of 1a was to get the Fast Fourier Transform(fft) of the image for effective edge detection. For the fft of the image was found using the fft package found in the numpy library. The fft which was calculated was then passed through a high pass filter. Essentially, the center frequencies are tapered off. This tapering was done by simply forming a square block that acheived the desired output.

For part 2b, the tag of ID was to be detected by orienting the tag such that the lower right corner of the patch is white and the remaining three are black.

A. Part 1a

In this subsection of the question, as previously stated, fft of the image wes found, center frequencies were tapered, and inverse of the tapered fft was found to detect the edges of the thresholded image where the page on which the fiducial marker was drawn was is isolated. This isolation was done using thresholding of the image to get a binary image.

Further, the binary image was passed to a custom function that performed fft, removed noise, and returned the tapered fft along with it's inverse. The tapering was done using a square of side 500 (Readers may notice from the code that 250 iterations are performed in the x and y axes in the positive and negative directions, thus the side being double the number of the iterations). The size length was found by trying multiple values and was a Hit or Miss, with the final decision being to use a side of 500. The figures for this subsection can be found ahead in this report(figures 1 to 4).

B. Part 1b

Although seemingly easy, this was inarguably the toughest part of the whole assignment. The task was to detect the tag id of given fiducial marker in a frame. The report uses frame 11 for example purposes. This involved numerous steps, which are as follows:

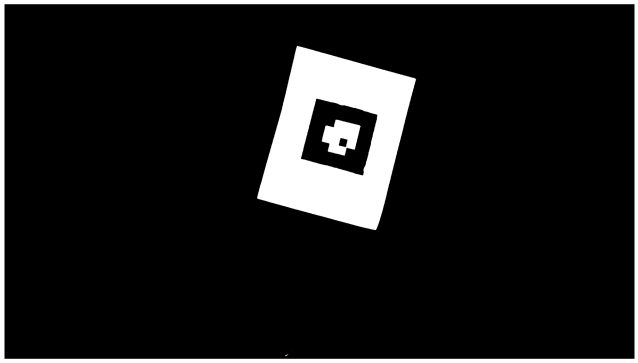

Fig. 1. Thresholded image

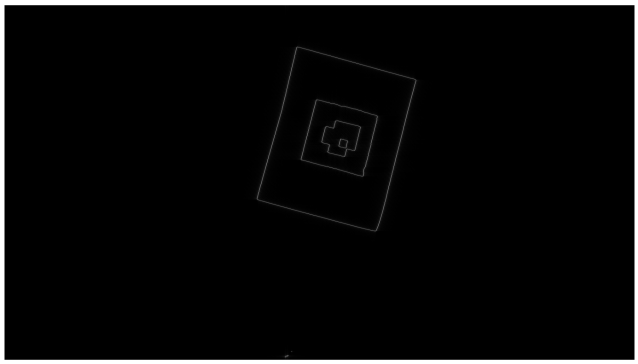

Fig. 2. Inverse of FFT of thresholded image

1) Reading the frame and converting it to grayscale for image processing operations.
2) Thresholding the grayscale image to isolate the sheet of paper containing the fiducial marker.
3) Isolating the fiducial marker.
4) Performing morpholigical operations on the marker to fill the marker
5) Bitwise and of isolated patch and morphological patch to preserve original shape of patch.
6) Detecting 4 extreme vetices of the marker patch.
7) Deciding the order of the corners detected.

The first two steps in the above list are self-explanatory. We will directly understand the third step. Step 3, involved threshlding exactly like Fig. 1. After this everything outside the marker patch is made to be white using flood fill function

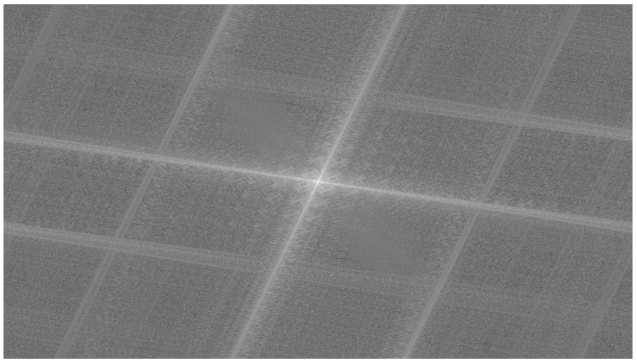

Fig. 3. FFT of thresholded image

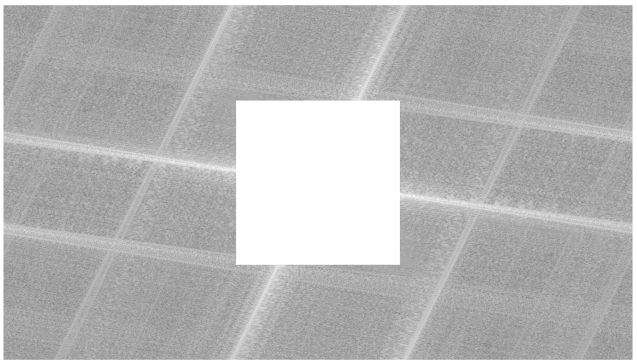

Fig. 4. High Pass Filtering on FFT of thresholded image

in cv2.

Then in Step 4, we perform morphological operations on the the isolated patch to fill the white hole. This is done using four cycles of open followed by closing, followed by 10 cycles of bitwise and of original isolated patch and Gaussian Blurring for shape preservation and noise removal, respectively. This step was thought of independently and is the key to the performance of the code accompanying this report.

Step 5 again used bitwise and for preserving original shape. The only drawback of this step is that it reintroduces noises due to low lighting in the frames that are found towards the end of the video. This noisy data occurs near the top left edge of the patch.

We now detect the vertices of the patch formed using the cv2.goodFeaturesToTrack() function. The parameters for the same were agian decided by trial and error.

Now, we arrive to the unique solution implemented in this code which is the ordering of corners. This also consumed the most time in the implementaion of the project and yet involved a solution from a concept that is taught in 11th grade. For this, the corner with the lowest distance from the origin is selected to be the first corner. Then, the Euclidean distances of the remaining corners are found from this first corner; the corner with the maximum distance is selected to be the third corner. A straight line is drawn between the images using two point form of a line and the qeuation is used to decide which corner

Fig. 5. Patch shape post morphological operationse

lies of left or right depending on the sign of the value found from the equation. In the accompanying code, line is drawn from point 3 to point 1 and point 2 is set to be the one with negative value , while point 4 is set to be the one with the positive value. The following figure shows the ordered corners.

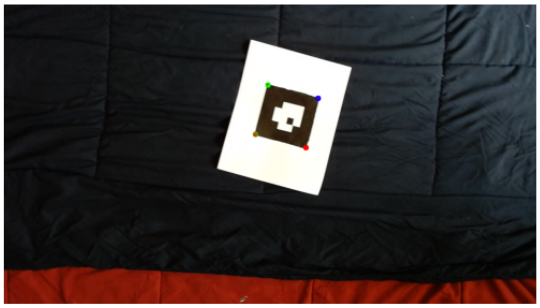

Fig. 6. Ordered corner detection

Thus, the corners are detected which are used to find the Homography matrix of the current patch. Now, we extract the tag in the image, detect it's orientation, and id. This is done by rotating the tag till the lower right corner is white and others are black. The tag id number is decided by starting in the upper left corner of the center of the code and going clockwise for finding the LSB to MSB for a 4 digit binary number. The detected Tag id in this video was 7.

III. Question 2

A. Part 2a

We now arrive at the AR part of the assignment. Part 2a, was imposition of the given Testudo mascot image onto the fiducial

Fig. 7. Detected and Oriented Tag

marker with proper orientation. This is acheived by getting the orientation of the fiducial marker found in the previous step and rotating Testudo image in the opposite direction. The Image was resized to a square and then imposed onto the tag accordingly, as seen in the figure below.

Fig. 8. Testudo image projected onto the tag

This is done for every frame in the video. It must be noted that the impostion itself is not programmatically efficient. It was found in peer discussions that this performance may be improved using vectorization, which, due to time and knowledge constraints was not implemented in the accompanying code.

B. Part 2b

Fig. 9. AR Cube

The final part of the assignment involved the generation of an AR cube onto the fiducial marker. To do this, the length and breadth of the marker is used and their mean is passed as the height of the cube, which in the code may be noticed to be negative (code 'cube.py', line 173), which is due to the reason that we want it to project towards the camera. If it is kept positive, it would project away from the viewing point and into the projective plane. The mathematical process provided in the supplementary document was used to perform this task. It may be noted from the video that the cube seem to jiggle. This is due to the fact that the Matrix 'K' is asssumed to be a constant and thus does not necessarily get calibrated. The sample for the same is provided below.

IV. Conclusion

We may derive two conclusions by judging the performances of part2a and 2b:

1) It may be noted from the image imposition that effective illumination of the scene is important for efficient feature detection.
2) The camera's intrinsic parameter matrix need to be dynamically calibrated for efficienct AR projections.

Present technologies like low light photography and methods like histogram equalization may be able to solve these problems. As for K matrix calibration, that is altogether an independent project of it's own.

V. Appendix - video links

Please note that the videos are unlisted and can be only accessed using these links -

1) Testudo image on fiducial marker.Testudo AR
2) AR Cube on fiducial marker.Cube AR

Extras:

1) Morphological outcome.Image Processing Outcome
2) Ordered Corner detection.Ordered corners
3) Blooper Reel.Blooper