ENPM673# Project 1

Problem 1
1(a). AR tag detection using FFT.

The goal of this question is to detect the AR tag in any frame of the video. I consider the first frame of the video and the following are the steps performed.

Approach

Taking the FFT of the image converts the input from spatial to frequency domain which means the low frequency components would be concentrated in the center of the image and the high frequency components would be scattered away from the centre as can be seen in the second image in above plot. I use a high pass filter with a circular mask of zeros to remove all the low frequency components from the image and obtain an image with just the higher frequencies which in this case would be the edges of the tag. To reconstruct the image IFFT is performed to get the edges of the image in spatial domain.

Program Logic

1. Convert the BGR image to grayscale.
2. Use threshold function to obtain the binary image.
3. Find the contours of the image.
4. Create a mask and draw a contour across it in the same position as in the original image.
5. Perform a bitwise operation with the image and the mask to remove the background of the frame.
6. Convert the input image into a np.float image to perform DFT to find the frequency domain.
7. FFT.fftshift is used to shift the low frequency components to the center of the image and the high frequency components are scattered around.
8. Implemented a high pass filter by way of a circular mask which has a circle of zeros in the center.
9. IFFT is performed to undo the shift from the centre and reconstruct the image.
10. The plots are displayed in MATPLOTLIB.

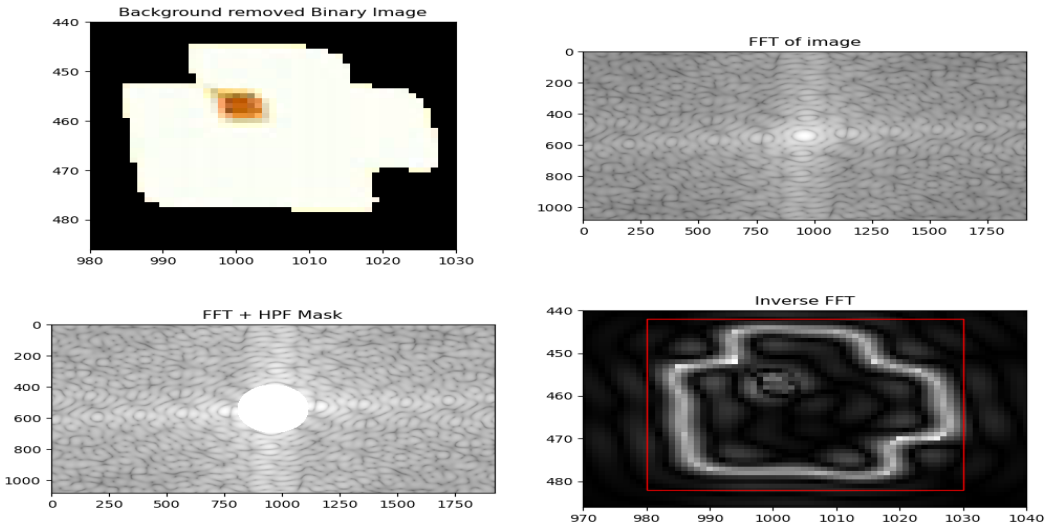

Figure 1: FFT and IFFT process to detect the AR Tag from a frame of the video

Observations
From the plots obtained the following observations can be made
1. The image on the top left is is the image of the AR tag with the background components removed.
2. The second image on the top right is after performing the FFT and shifting the low frequency components to the center of the image.
3. The image on the bottom left is the implementation of the circular mask which creates a circle of zeros at the center thus eliminating the low frequency components and getting the edges.
4. The final image on the bottom right is formed after performing IFFT to reconstruct the image and obtain the AR tag's edges. The tag is enclosed within a bounding box

Problems Encountered
Running the FFT without removing the background created a contour around the white area surrounding the AR tag as well. Even using an HPF did not eliminate the surrounding area of the tag. Therefore I removed the background altogether by creating a mask and performing the cv2.bitwise() operation.

1 (b) Decode custom AR tag

Approach
The first step was to form contours on the image based on which the corners can be detected. I perform a Gaussian blur to the grayscale image to smoothen it. Based on which I obtain the contours and hierarchy of each contour. I then select the contour which has a parent. I then use the contour approximation function cv2.approxPolyDP() to reduce the number of corner points. I try to reduce it to a shape with 4 corner points. The contour corresponding to inner AR tag would be the only contour with a parent. After selecting this contour the contours are drawn on the original frame.

Homography
The next step is to perform homography between the corner points obtained and a dummy image corresponding to the region of the AR tag. I consider this dummy image to be a 200x200 image as it would be easier to create an 8x8 grid of squares with each cell being represented as a white or black pixel.

Warping
Once the homography matrix is obtained warping needs to be performed to multiply warp the image containing the AR tag using the homography matrix. The inverse of the homography matrix needs to be taken to multiply with the pixel values of the image to obtain an image consisting of just the AR Tag alone.

Tag Decoding
To detect the tag the newly obtained image needs to be divided into an 8x8 matrix with each cell representing a value of 1 or 0. The pixel value for a cell is assigned by counting the number of black and white cells in each cell and assigning the corresponding value if 1 is more than the other. The orientation is obtain by checking the (2,2),(2,5),(5,2) and (5,5) cells and identifying the cell consisting of the white pixel and based on that the angles 0,90,180,-90 is assigned. Based on the angle the inner grid comprising of (3,3),(3,4),(4,3),(4,4) values are taken from the most significant bit to the least

significant bit to decode the binary value which is displayed on the terminal and the equivalent decimal value is displayed in the video outputs

The following images show the decoded AR tag on a frame of the different videos. Clicking on the images will open the YouTube link of the tag detection on the different videos provided for the project.

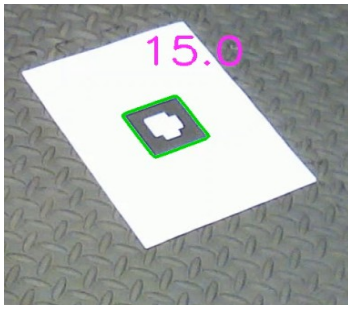

Figure 2: AR Tag detection and decoding in video Tag0.mp4

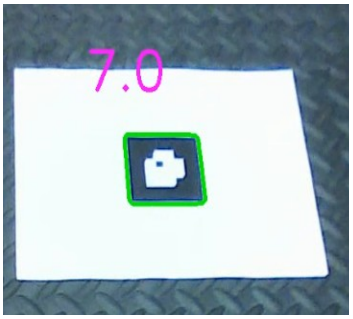

Figure 3: AR Tag detection and decoding in video Tag1.mp4

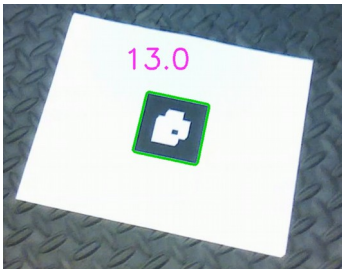

Figure 4: AR Tag detection and decoding in video Tag2.mp4

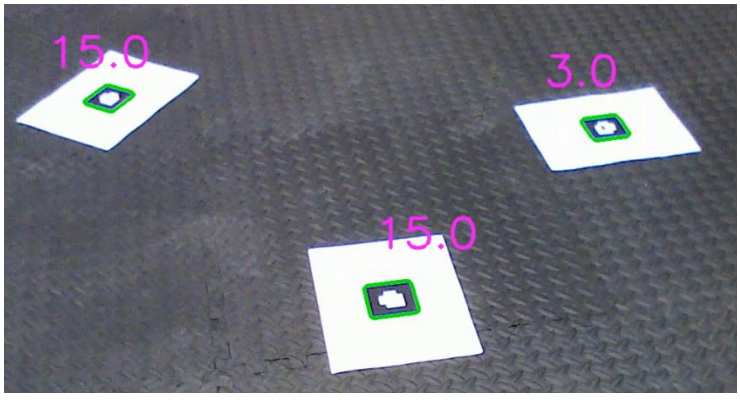

Figure 5: AR Tag detection and decoding in video multipleTags.mp4

Problems Encountered
Since I am using angles of 0,90,180,-90 whenever the frames in the video shifts to a different angle there is a very slight delay to detect the orientation and the tag ID. This could be a problem when working on a realtime application. Another issue was when I run the program on the video containing multiple tags. All the tags are not being continuously detected. The tag in the far side of the video does not get detected all the time. For different frames there are different tags which end up on the far side of the video which causes the tags to be detected only occasionally.

2(a) Superimposing image onto tag:

Approach

The following are the steps to superimpose the image testudo.png after detecting the AR tag in each frame.
1. Based on the orientation of the AR tag the corner points of the image to be superimposed needs to be oriented in the same way.
2. The homography is calculated between the corner points of the template image testudo.png and the corner points of the 8x8 frame based on the orientations of the AR tag.
3. The warp perspective is implemented again using this homography matrix obtained in the previous step to superimpose the pixels of the image on the frames of each video.

The following images show the superimposed image on a frame of the different videos. Clicking on the images will open the YouTube link of the image superposition on the different videos.

Figure 6: Superimposed image in Tag0.mp4

Figure 7: Superimposed image in Tag1.mp4

Figure 8: Superimposed image in Tag2.mp4

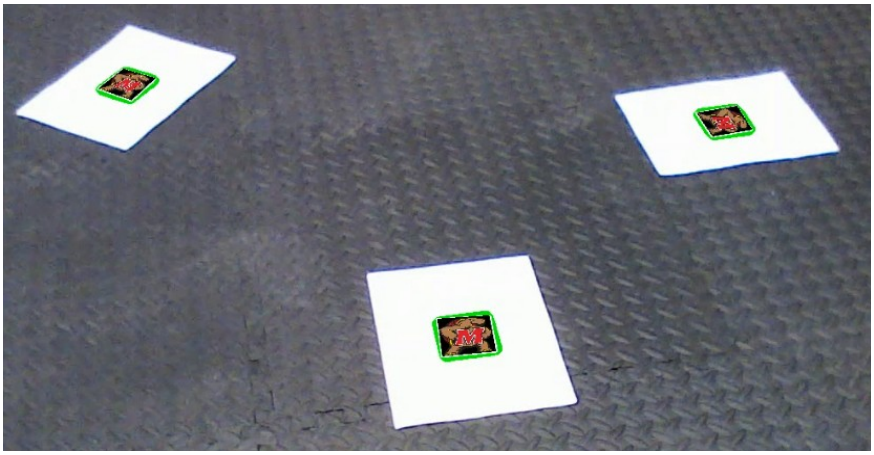

Figure 9: Superimposition of image in multipleTags.mp4

Problems encountered

Not assigning the proper corner points based on the AR tag orientation can cause a lot of jitters or the image could be oriented wrongly. Appropriate corner points of the template should be mapped with the corner points of the AR tag. I get an image like this when corner points are not properly assigned. Also the image should not change its orientation and match that of the tag in all the frames. All these problems were eliminated by assigning proper corner points.

Another problem was when trying to superimpose the image on the video containing multiple AR tags. Similar to the last case the tag is not constantly present on the tag as the image goes to the far side of the video.

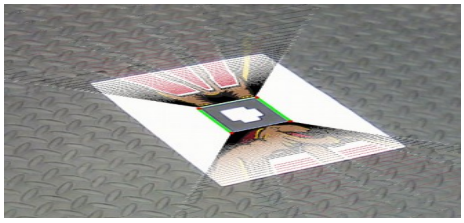

Figure 10: Jitters while superimposing tag due to incorrect assignment of corner points

2(b) Placing a virtual cube on the Tag

Approach

In this case once the corners are obtained based on the AR tag the homograhy matrix needs to be computed between the world coordinates (positions of the virtual cube) and the corner points.
The projection matrix needs to be computed using the camera intrinsic parameters (K matrix whose transpose needs to be taken) and the homography matrix. The first 2 rows of the homography matrix is used to compute lambda and Btilda based on which the B matrix is obtained. The final projection matrix would be a 3x4 matrix consisting of the rotation and the translation elements

The following images show the projection of the virtual cube on a frame of the different videos. Clicking on the images will open the YouTube link of the image superposition on the different videos.

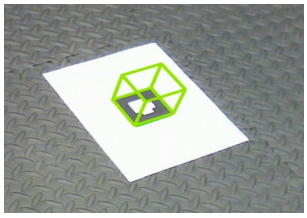

Figure 11: Cube Projection in Tag0.mp4

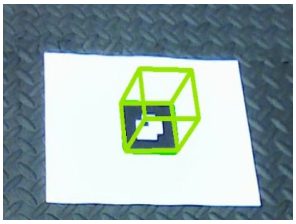

Figure 12: Cube projection in Tag1.mp4

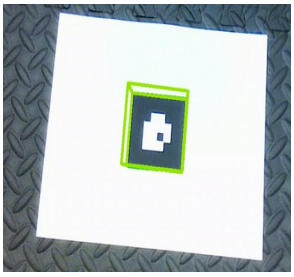

Figure 13: Cube projection in Tag2.mp4

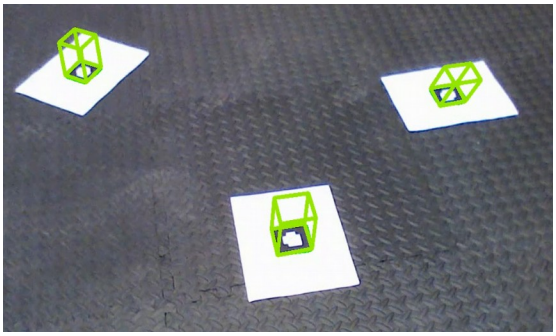

Figure 14: Cube projection in multipleTags.mp4

Problems Encountered

Similar to the previous case imcorrect assignment of corner points can cause jitters and the lines connecting the virtual cube to the image might not be correct. The cube might be properly formed as the frames might go on but it would not be consistent. Similarly when running the video with multiple tags the formation of the cubes is consistent on the tags on the far side of the video because the detection of the contour is not consistent as well. In general there are a lot of jitters when superimposing an image or projecting the virtual cube on the AR tag.

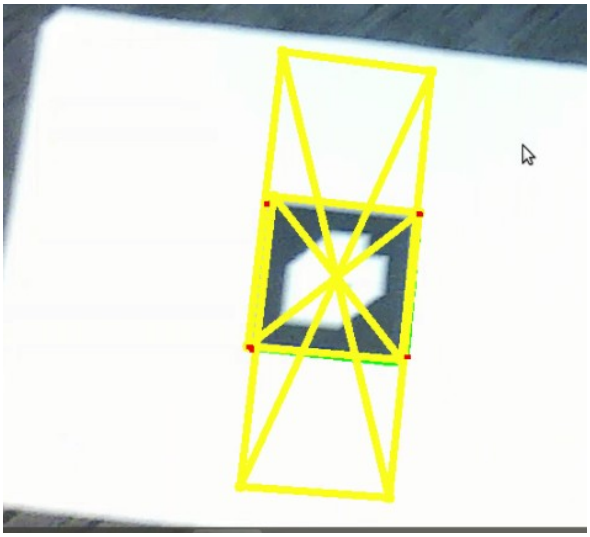

Figure 15: Incorrect projection from the cube onto the AR tag.