Project 1

Darshan Jain
UID: 117454208

Problem 1a.

The task here is to detect the April Tag in any frame of Tag1 video (just one frame). Notice that the background in the image needs to removed so that you can detect the April tag. You are supposed to use Fast Fourier transform (inbuilt SciPy function fft is allowed) to detect the April tag.

Solution:

1. To find the edges in the image first I pre-processed the image using blurring, thresholding and converting to grey scale so that the minute edges of the background are removed and the April tag can be detected.
2. The second step was to convert the image in the frequency domain using Fast Fourier (fft) function so that the edges can be detected. The way fft works is that the low frequency components are placed at the edge of the corners of the two-dimensional spectrum and the high frequency are placed at the centre. So, the spectrum is shifted to place the zero frequency components are at the centre of the circle. The magnitude of the spectrum is shown below in figure 1.
3. As the edges are high frequency components in an image, an appropriate high pass filter must be used to capture the edges. Accordingly, a high pass filter was designed by applying a circular mask of zeros at the centre and multiplied to the fft spectrum as shown in figure 1
4. Finally inverse fft of the masked fft spectrum was obtained which is shown in figure 1 below. In which the edges of the white sheet and the April tag are more prominently visible.
   The program flow diagram is shown below and the figure of various plots is also shown below. Figure 2 shows the flow diagram for problem 3 used in the code.

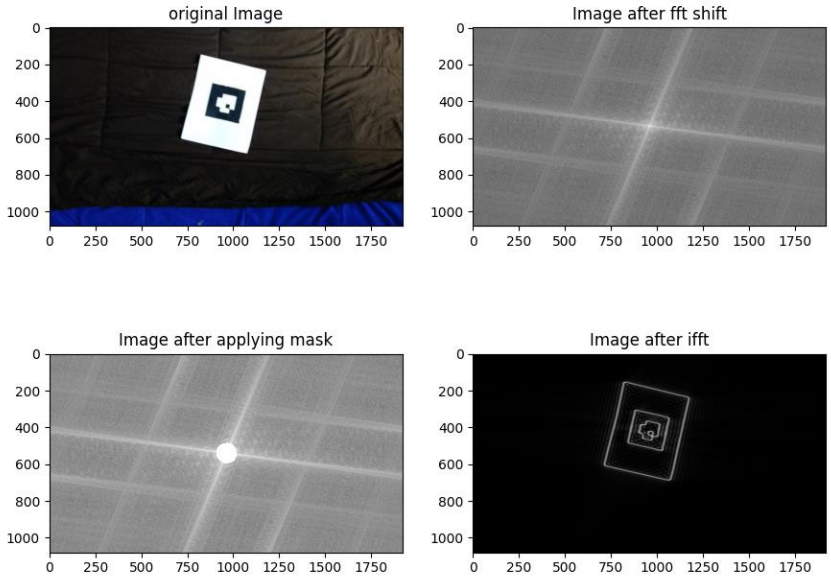

Figure 1: Plot of input image at various checkpoint

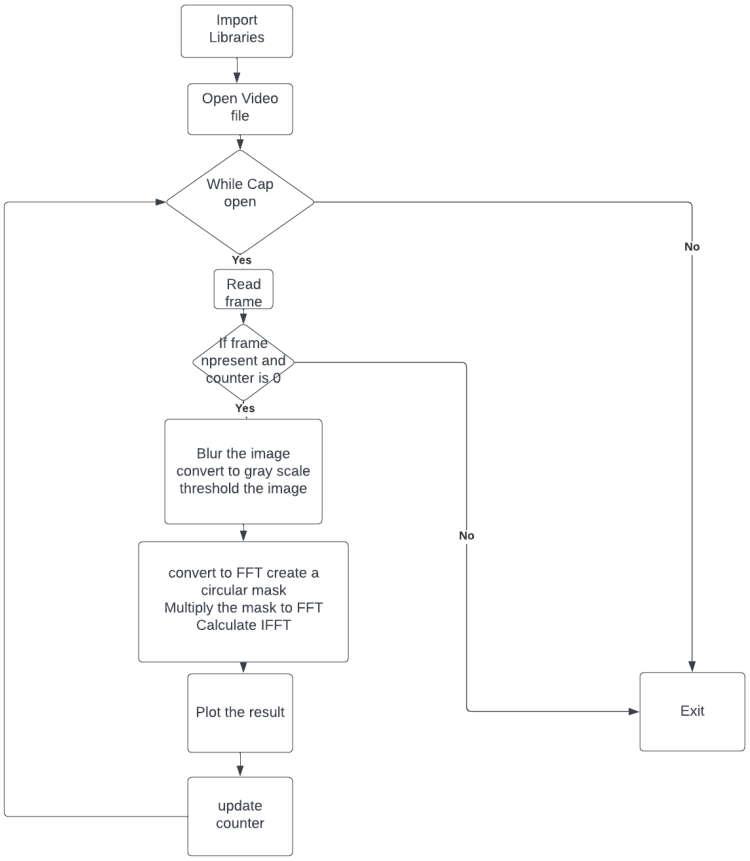

Figure 2: Flow Diagram for Problem 1

Problem 1b.

You are given a custom AR Tag image, as shown in Figure 3, to be used as reference. This tag encodes both the orientation as well as the ID of the tag.

Figure 3: Reference AR tag

Solution:

1. To perform encoding first we have to extract the 8*8 grid and choose the inner 4*4 grid which contains the actual AR tag and are obtained by removing the black padding around the tag. The structure diagram of an AR tag is shown in figure4. The grid diagram is as shown in figure 5 which is created on the reference image.
2. After the inner 4*4 grid is obtained we have to orient the AR tag using the orientation decoding layer as shown in the figure 4. The color pattern of the four corners must be such that the bottom right corner is white and all other corner are black.

3. Finally, after orientation is done, we can decode the AR tag can be decoded by using the inner 2*2 grid block numbered from 1-4, 1 being the most significant bit and 4 being the least significant bit. A white block holds a high level and a black block holds low level. In our case the decoded output is (1111) which in decimal is equivalent to 15 and is shown in the figure 6.

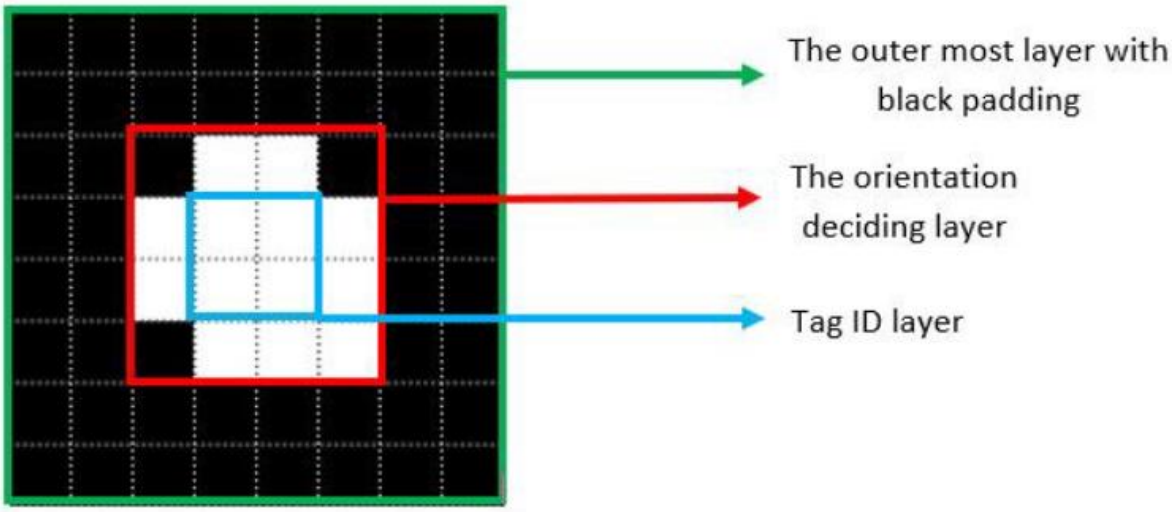

Figure 4: Structure diagram of an AR tag

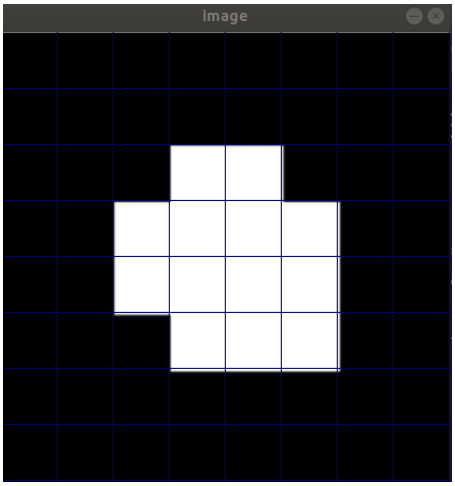

Figure 5: Grid diagram of AR tag

```
darshan@dpj:~/Desktop/ENPM673/Project_1$ /usr/bin/python3 /home/darshan/Desktop
/ENPM673/Project_1/Project_1_P1b.py
tag_id  [1, 1, 1, 1]
number = 15
```

Figure 6: Output for the reference AR tag

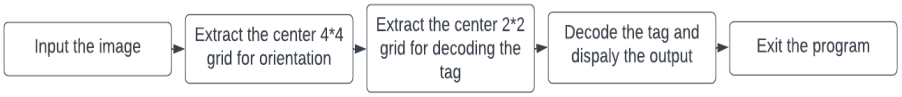

Figure 8: Flow diagram for the program

Problem 2a: Testudo Warping

The flow of the program is as follows:

1. First, we detect the possible corners in the frame after we are done with preprocessing the frame.
2. After that we extract the corner which represent the AR tag and save them in clockwise form with top left been the first point.
3. Then we create a frame using the corner points found in the previous step and perform homography between the tag corners and the frame to obtain a template. The testudo image is also resized to the size of the frame.
4. After that I warped the inverse homography and the template obtained in the previous step to obtain a warped testudo in the same shape as the input image frame.
5. Now we replace the part of the frame with AR tag with the testudo warped image.

The overall program flow is as shown in figure 9. Figure 10 shows one of the frames from the program where we can see testudo attached to the actual frame.

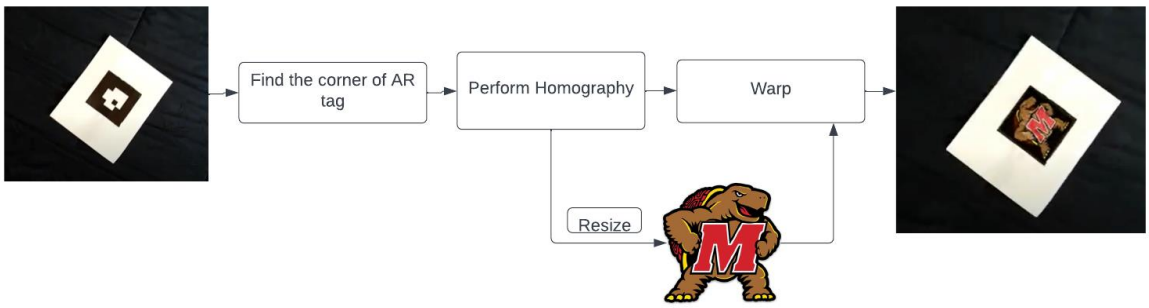

Figure 9: Flow of the program for testudo warping

Figure 10: Actual frame from the program

Problem 2b: Cube projection

Solution:

This problem is some what similar to the previous problem. Let's see the steps followed to solve the problem.

1. First find the corners in the preprocessed frame.
2. Then extract the corners of interest from the corners found i.e., the corners for the AR tag.
3. Now we compute homography between the found corners and a reference frame of arbitrary value.
4. Then we compute the projection matrix with the homography matrix and the camera constant matrix K.
5. The steps below show the way to compute the projection matrix
   - Define h1, h2, h3 as the three columns from the homography matrix H
   - Compute the scale factor using the formula $\lambda = \frac{|K^{-1}.h1| + |K^{-1}.h2|}{2}^{-1}$
   - Compute $b = \lambda . K^{-1} H$.
   - Compute $\check{b} = b(-1)^{|b|<0}$, i.e., $\check{b} = -b$ if determinant of $\check{b}$ is negative.
   - Now compute r1 = b1, r2 = b2, r3 = r1xr2 and t = b3 where b1, b2, b3 are the columns of $\check{b}$.
   - Combine the above vectors to form projection matrix.
6. Now after we have the projection matrix, we find the cube coordinates in the image frame by multiplying the projection matrix with 8 reference coordinates.
7. Finally, we draw line between the coordinates to generate a cube on the input frame.

The program flow is shown in figure 11. Figure 12 shows the actual output from the program in which we can see the cube on the AR tag.

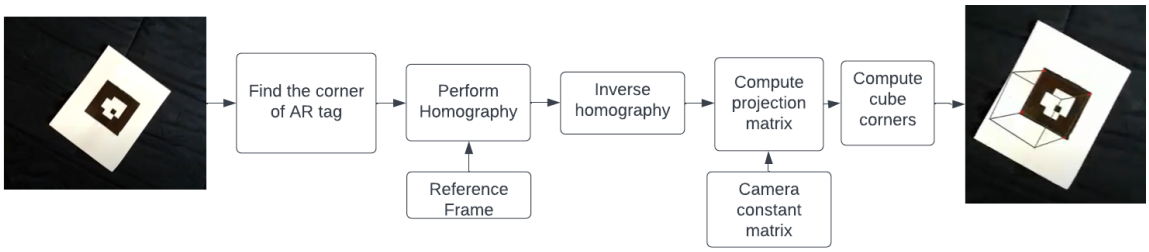

Figure 11: program flow for projecting cube

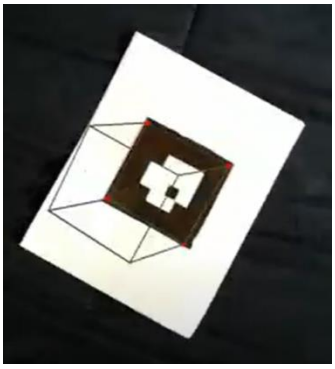

Figure 12: Actual output frame from the program

DexiNed(Bonus question):

The paper is about edge detection using DexiNed neural network based on VGG16 architecture. Paper addresses the issue with other architecture which are trained on datasets created for segmentation purposes. The author as himself created a dataset by the name BIPEDv2 which is better for edge detection purpose.
The DexiNed architecture consist of 6 blocks and each block is further divided into subblocks with a group of convolution layer. Skip connections are used in this architecture. They couple the blocks and sub blocks with each other. The feature map generated at each block is given to USNet which is a conditional stack of two block which consist of a sequence of one convolutional and deconvolution layer to create intermediate edge map. This edge maps then forms a stack of learned features which at the end of network a fused together to form a single edge map.

Figure 13: Actual image

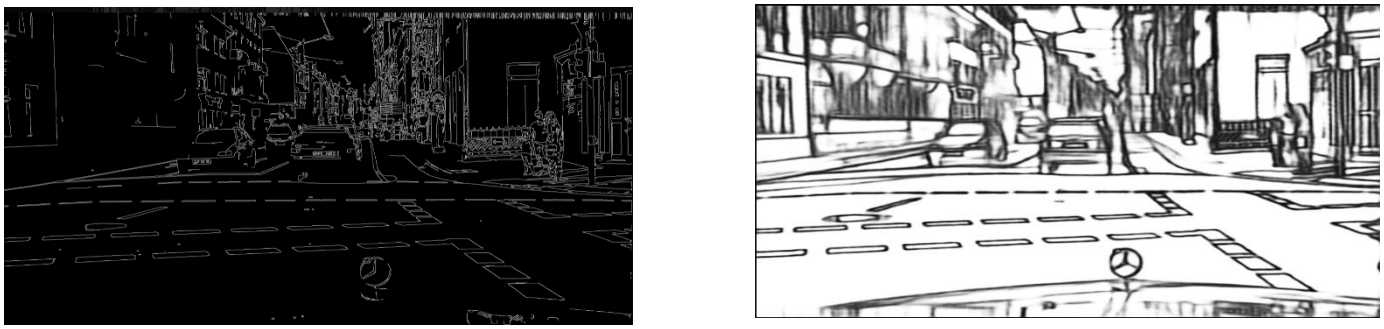

Figure 14: Canny Edge(left); DexiNed(right)

From figures 13 and 14 it can be seen that DexiNed detects edges finer than Canny edge detection. In DexiNed minute edges are also detected around the building on the left and we can also see that it detects the features like humans which are not as clear in canny edge detection. We can also see that the output generated from DexiNed is much like as if it was drawn by human.

Video Output Cube projection: Cube
Video Output Testudo Warping: Testudo