Abhijit Vinod Mahalle 117472288
ENPM673 Project 1 Report

Problem 1a

1. The task here was to detect the edges of an April Tag in any one frame of the given video using Fast Fourier Transform. Fast Fourier Transform(FFT) converts an image from spatial domain to frequency domain.
2. Edges are associated with high frequency. To detect edges, low frequency should be removed from the frequency channel. Low frequency is concentrated at the center of the channel.
3. A circular mask of suitable radius was being applied at the center of the frequency channel so that low frequency values are eliminated, and only high frequency value remain. Inverse of the FFT converts the image back to spatial domain from frequency domain.
4. The resulting image only comprises of edges of the April Tag.

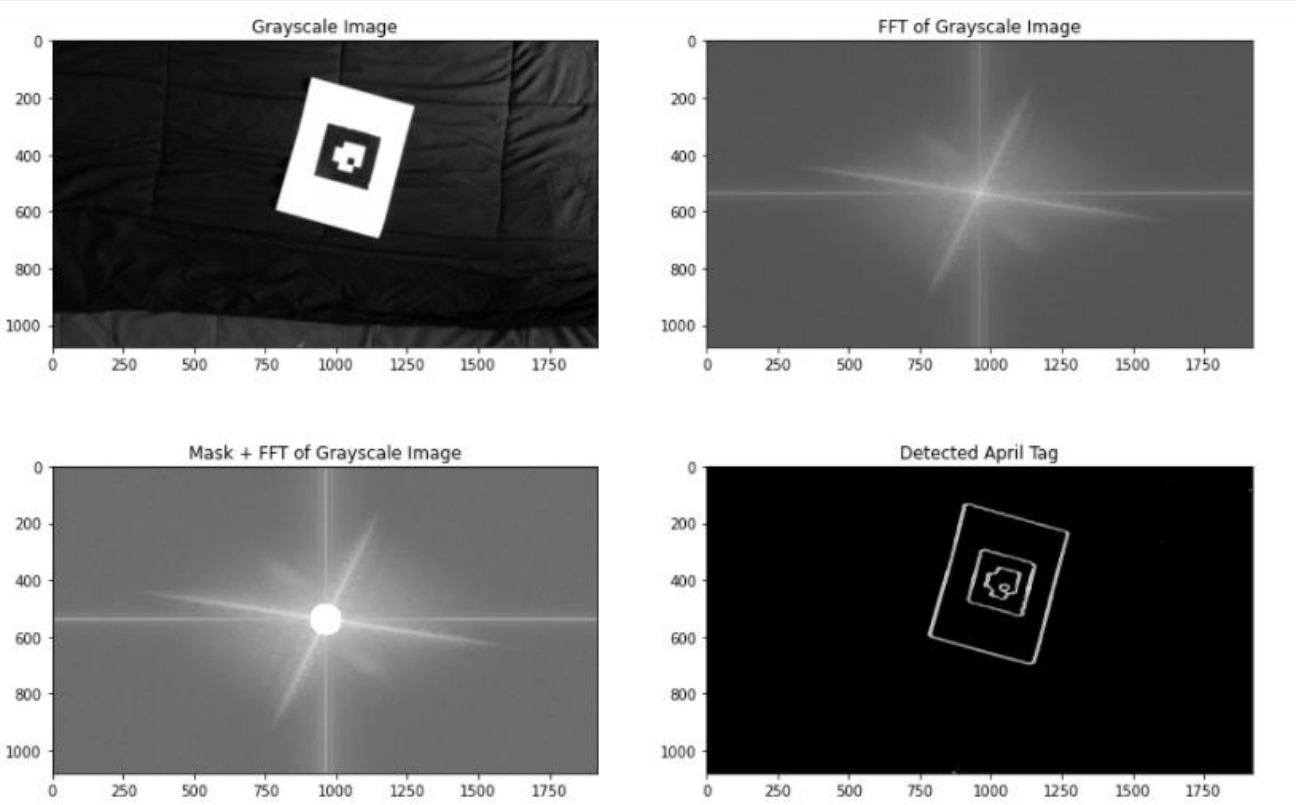

Fig. 1: Results of using FFT to detect April Tag

Challenges:

To compute the FFT of an image, “fft2” function of the SciPy library should be used. Initially, “fft” function was being utilized to find the FFT and was yielding unexpected results like an image without any edges after computing inverse FFT. This function is suitable for one-dimensional data, unlike an image. To compute FFT on an image, a two-dimensional data, “fft2” function should be utilized. The low frequency channel needs to be shifted to the center to make the masking easier. The frequency channel obtained earlier while not shifting the center did not produce a symmetric frequency channel.

Pseudo Code:

1. Extract random frame from video.
2. Apply Gaussian blur to the image to reduce the noise.
3. Convert the processed image to grayscale.
4. Perform FFT on the grayscale image.
5. Shift the low frequency to the center of the FFT.
6. Apply circular mask of suitable radius to the FFT spectrum.
7. Shift the low frequency value back to its original location.
8. Compute inverse of FFT.
9. Display the resulting image.

Problem 1b

1. The task was to decode the given April Tag. The image of the April Tag was sliced into 8x8 blocks of NumPy array and stored in a list.
2. The median value of each block was computed and all the pixels within a given block were assigned the corresponding median value.
3. The median of pixel intensities of blocks at indices [3,3], [3,4], [4,4], and [4,3] (in the same specific order) were calculated.
4. If the median value of a given block was above a threshold value of 220, value “1” was stored in the list corresponding to that block. If the median value was less than the threshold, value “0” stored in the list for that blocking.
5. The generated list contains the id of the identity of the April Tag in the binary format. The binary number was then converted to decimal number to get April Tag’s identity in the decimal format.

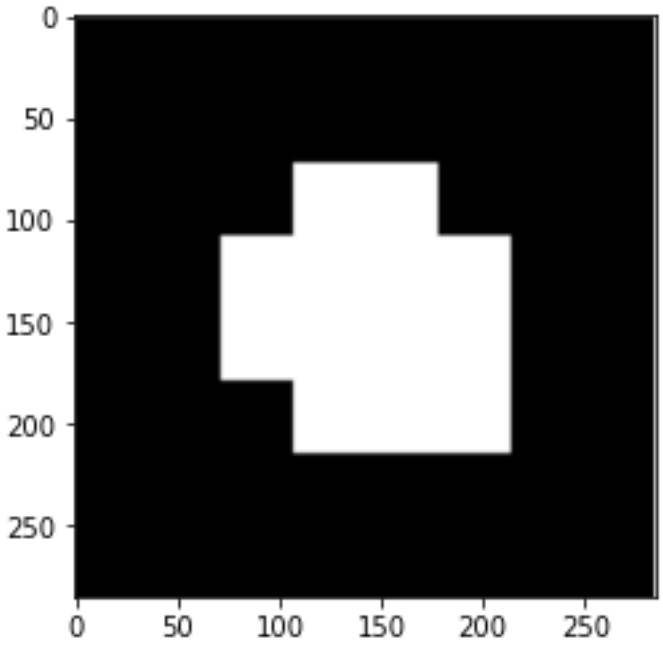

Fig. 2: April Tag used to decode

Result:

The ID of the April Tag is **15**.

Pseudo Code:

1. Convert the April tag into grayscale image.
2. Threshold the grayscale image to appropriate value.
3. Divide the image into 8x8 blocks.
4. Compute the median value of pixel intensity of block (6, 6)
    if value != 255:
        rotate the image by 90 degrees.
        Repeat step 4
    else:
        Jump to step 5
5. Determine the median value of pixel intensity for blocks (4, 4), (4,5), (5,5), and (5,4) in the given order.
6. if value != 255:
    Assign value 1 corresponding to the block
   else:
    Assign value 0 corresponding to the block
7. Convert the binary value corresponding to the blocks in the above order to decimal value.

Problem 2a

1. The task was to superimpose the Testudo image on the April Tag in the video and save the resulting video output.
2. The corners of the April Tag in one frame were detected first using goodFeaturesToTrack function from OpenCV.
3. The corners of the white paper were eliminated by eliminating the corners corresponding to minimum and maximum values of x and y co-ordinate values.
4. The corners corresponding to the current minimum and maximum values of x and y co-ordinates are the corners of the April Tag.
5. Homographic transformation was then performed to straighten the April Tag by mapping the April Tag onto a black image and storing its pixel value.
6. Clockwise 90 degrees rotations were performed on the transformed April Tag image to orient it to its base position and the number of rotations were stored. The Testudo image was then rotated in the counterclockwise direction by the same number of times.
7. Homographic transformation was then performed between the Testudo image and the April Tag and the pixel value at a given co-ordinate of the Testudo image was assigned to the pixel at the transformed coordinates after performing Homography.
8. The above process was then repeated for all the frames and the processed frames were stored in a video object to generate the output.

Fig. 3: Superimposed Testudo image on April Tag

Challenges:

1. Corner detection was difficult as some stray corners were getting detected.
2. Image pre-processing operations like Gaussian blurring, thresholding and morphological operations like dilation and erosion were performed on each frame of the video to minimize the detection of stray corners.
3. Sometimes false corners were detected on the edges of the white paper. To eliminate them, lines were formed using the true corners of the white paper and the points that were within some threshold values were deleted. In this manner, the false corners on the edge of white paper were eliminated.
4. Finding the orientation of the April Tag was also difficult since after performing homographic transformation, the image of the April Tag was not mapping correctly.
5. To overcome this, the x and y co-ordinates of the pixels after computing homographic transformation were divided by the third co-ordinate of the homographic co-ordinates.

Pseudo Code:

1. Pre-process the frame.
2. Detect corners using goodFeaturesToTrack
3. Find corners of Tag by eliminating the corners associated with the white paper.
4. Perform homographic transformation on the April Tag to straighten it.
5. Determine the number of clockwise rotations required for the April Tag to reach its home orientation.
6. Perform equal number of counterclockwise rotations on the Testudo image.
7. Compute Homography matrix from the Testudo image to the April Tag.
8. Map every pixel co-ordinate from the Testudo image to the April Tag and assign the mapped pixel co-ordinate the pixel value of the corresponding pixel in the Testudo image.
9. Repeat the process above steps for every frame of the video and store the processed image in the video object.

Problem 2b

1. The task was to draw a virtual cube over the April Tag.
2. The origin of the world co-ordinate was placed at the bottom face of the topmost left corner of the cube. The cube was assigned the unit length.
3. Homography matrix was computed between the co-ordinates of the bottom face of the cube to the April Tag co-ordinates in the image plane for each frame.
4. Projection matrix was computed using the Homography matrix and the provided camera matrix.
5. All the three-dimensional corners were mapped to two-dimensional image co-ordinates by multiplying them with the Projection matrix.
6. The detected corners of the cube in the two-dimensional image plane were then joined using lines to represent a three-dimensional cube on a two-dimensional image plane.
7. The above process was repeated for every frame and the processed image was saved in a video object.

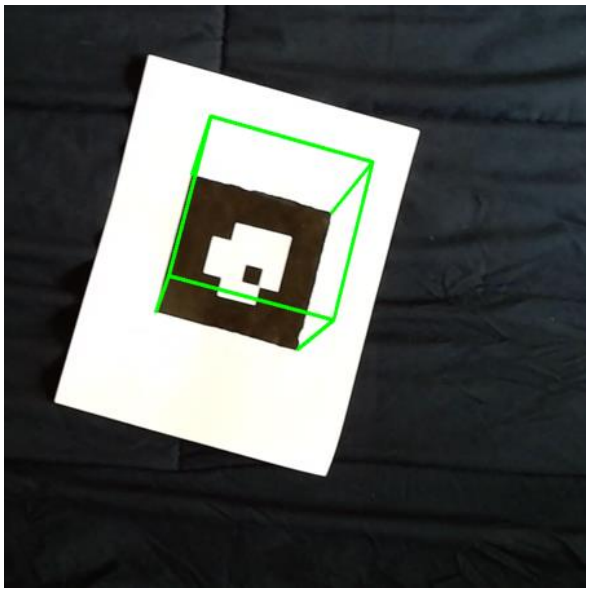

Fig. 4. Virtual Cube

Pseudo Code:

1. Pre-process the frame.
2 Detect corners using goodFeaturesToTrack.
3 Find corners of Tag by eliminating the corners associated with the white paper.
4 Compute homography matrix from cube's world co-ordinates to April Tag corners.
5 Compute Projection matrix using Homography matrix and Camera Matrix.
6. Multiply three-dimensional co-ordinates of cube with Projection matrix to find two-dimensional co-ordinates in image plane.
7. Draw line between the corners in the image plane.

Extra credit

Network Architecture of DexiNed

1. DexiNed is a neural network based on VGG16 architecture. The paper on DexiNed addresses the issue with other state of the art architectures like RCF, BDCN, and CATS which are trained on BSDS dataset which was created for the image segmentation purpose.
2. The paper presents the based on VGG16 CNN architecture along with a dataset BIPEDv2 that better suit the edge detection problem.
3. The DexiNed architecture had six blocks acting like an encoder. Each block is a collection of smaller sub-blocks with a group of convolution layers.
4. The skip connections which are generally used in deep neural networks are used in this architecture. Skip-connections couple the blocks as well as their sub-block with each other.
5. Each block generates feature-maps that are fed to a separate USNet to create intermediate edge-maps. These intermediate edge-maps are concatenated to form a stack of learned filters.
6. These features are fused to generate a single edge-map at the very end of the network.
7. Unsampling network USNet is a conditional stack of two blocks. Each one of then consists of a sequence of one convolutional and deconvolutional layer that up-sample features on each pass.
8. A deconvolution increases the dimensions of the image and reduce the number of filters as opposed to the convolution operation.
9. Each sub-block in the USNet constitutes two convolutional layers.
10. Each convolutional layer is followed by batch normalization and a rectified linear unit (ReLU). After the max-pooling operation, these SSC average the output of connected sub- blocks prior to summation with the first skip-connection.
11. In parallel to this, the output of max-pooling layers is directly fed to subsequent sub- blocks.

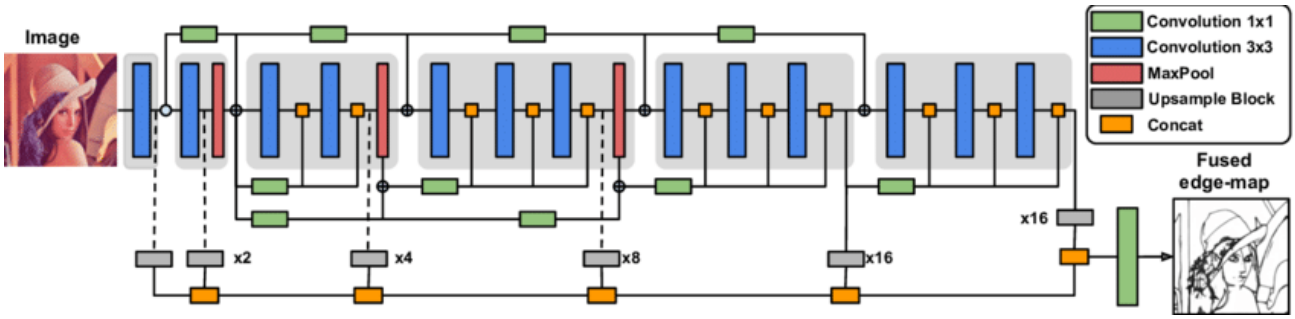

Fig. 4. Network architecture of DexiNed

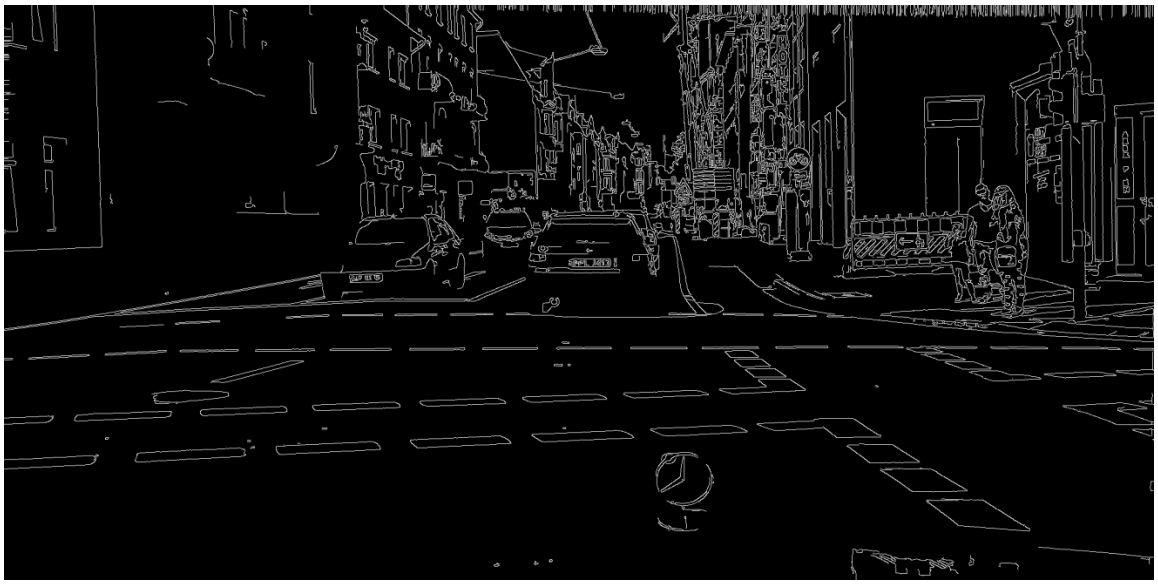

Fig. 5: Edge detection using Canny Edge Detector

Fig. 6: Edge detection using DexiNed

Comparison of results of Canny Edge detection and DexiNed:

1. In canny edge, pixels in the detected edges have value 255, while in the DexiNed, the pixels in the edges have value 0.
2. In DexiNed, the detected edges are easier to observe as opposed to the Canny Edge.
3. Since DexiNed is trained on a huge training data, it can detect more varieties of edges like edges on the curve.
4. DexiNed is faster compared to Canny Edge if the number of edges to be detected is high.
5. The edges detected by DexiNed are more continuous and do not abruptly break as opposed to Canny Edge.

Video Output Links:

https://drive.google.com/drive/folders/1Ersh5jQ5gG6O-w12Oadr2RzMQosyWMKX?usp=sharing