ENPM 673
Project 1 Report
Pradipkumar Kathiriya (117678345)

Question 1-A:

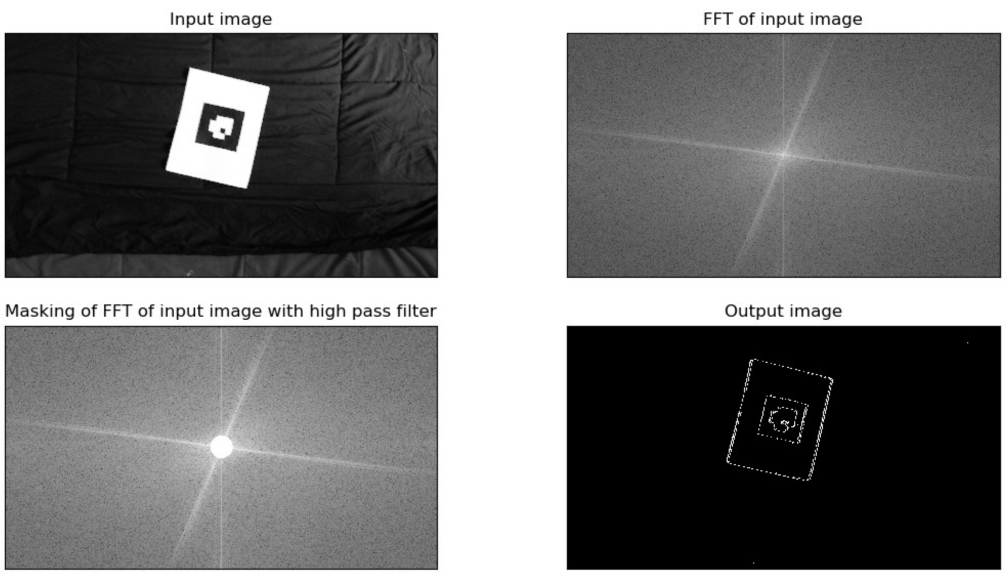
Method/Pseudo code:

def rescaleFrame(image, scale):
    return image with reduced scale

def maskArea(image, radius):
    return circular high pass filter mask

1. read single frame from video
2. re-scale image and convert to gray scale
3. apply Fourier transform
4. shift image origin to the center of the image
5. apply mask
6. shift image origin back to the top left corner
7. apply inverse Fourier transform

Result:



Question 1-B:

Method/Pseudo code:

def rescaleFrame(image, scale):
    return image with reduced scale

def computeHMatrix(corresponding points in video frame and blank image):
    construct A matrix
    apply SVD
    find the eigenvector corresponding to minimum eigen value
    reshape eigen vector into (3,3) matrix(H)
    divide each element by H(3,3)
    return H matrix

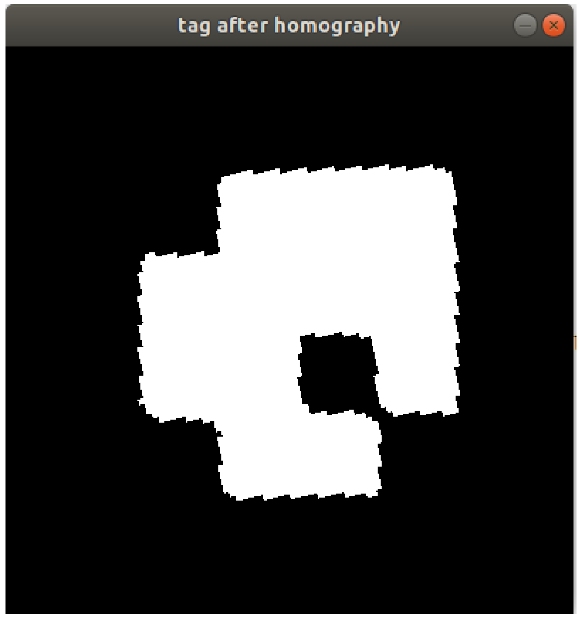
def computeHomography(H, video_frame, blank image):
    multiply each pixel of blank image with H matrix to compute corresponding points in video frame
    transport pixel from video frame into the blank image
    return homoprahy image (blank image)

def decodetag(tag):
    slice tag into 64X64 blocks
    rotate the tag till (5,5) block has intensity 255
    check the pixel intensity of (3,3),(3,4),(4,4)(4,3) in sequence.
    If pixel intensity is 255, label it as 1 other wise label it as 0. Stored it as a each element of string in sequence
    find the decimal number corresponding to binary number of above string
    return decimal number

1. Read single frame from video
2. Re-scale image and convert to gray scale
3. Process image to remove noise
4. Detect corner using shi-Tomashi corner detector
5. Find the white paper corner and remove it from the corner list
6. In the remaining corner list, corner corresponding to x_max, y_max, x_min and y_min is the corner of the tag
6. Create blank image and find its corner
7. Compute homography between blank image and video frame to impose tag into blank image.
8. Pass the tag imposed blank image to decodetag function. It will return tag ID

Result:

ID of the tag is 14



Question 2-A:

Method/Pseudo code:

```
def rescaleFrame(image, scale):
        return  image with reduced scale

def computeHMatrix(Blank image, corner list):
        find four corner of blank image
        find the white paper corner and remove it from the corner list
        In the remaining list, corner corresponding to x_max, y_max, x_min and y_min is the corner of
        the tag
        construct A matrix
        apply SVD
        find the eigenvector corresponding to minimum eigen value
        reshape eigen vector into (3,3)  and stored it as s matrix H
        divide each element by H(3,3)
        return H matrix

def computeHomography(H, video_frame, blank image ):
        multiply each pixel of blank image with H matrix to compute corresponding points in vide
        frame
        transport pixel from video frame into the blank image
        return homoprahy image (blank image imposed with tag)

def computeTestudoHomography(H, video_frame, Testudo) :
```

multiply each pixel of testudo image with H matrix to compute corresponding points in video frame
transport pixel from testudo into the video frame
return homoprahy image (video frame with testudo image imposed on it)

def tagOrientation(corner, video frame, blank image):
Impose tag from the video frame and imposed it on to blank image
slice tag into 64X64 blocks
initialized variable to store number of rotation
rotate the tag till (5,5) block has intensity 255
for each rotation increment rotation variable by 1.
return rotation(variable storing number of rotation)

1. Read each frame from video
2. Re-scale image and convert to gray scale
3. Process image to remove noise
4. Detect corner using shi-Tomashi corner detector
5. Find the white paper corner and remove it from the corner list
6. In the remaining corner list, corner corresponding to x_max, y_max, x_min and y_min is the corner of the tag
6. Create blank image and find its corner
7. Compute homography between blank image and video frame to impose tag into blank image
8. Check the rotation of tag using tagOrientation function and image of tag obtained from step above step 7
9. Rotate the testudo to align it with tag orientation
10. Impose the testudo on each video frame using computeTestudoHomography function

Result:

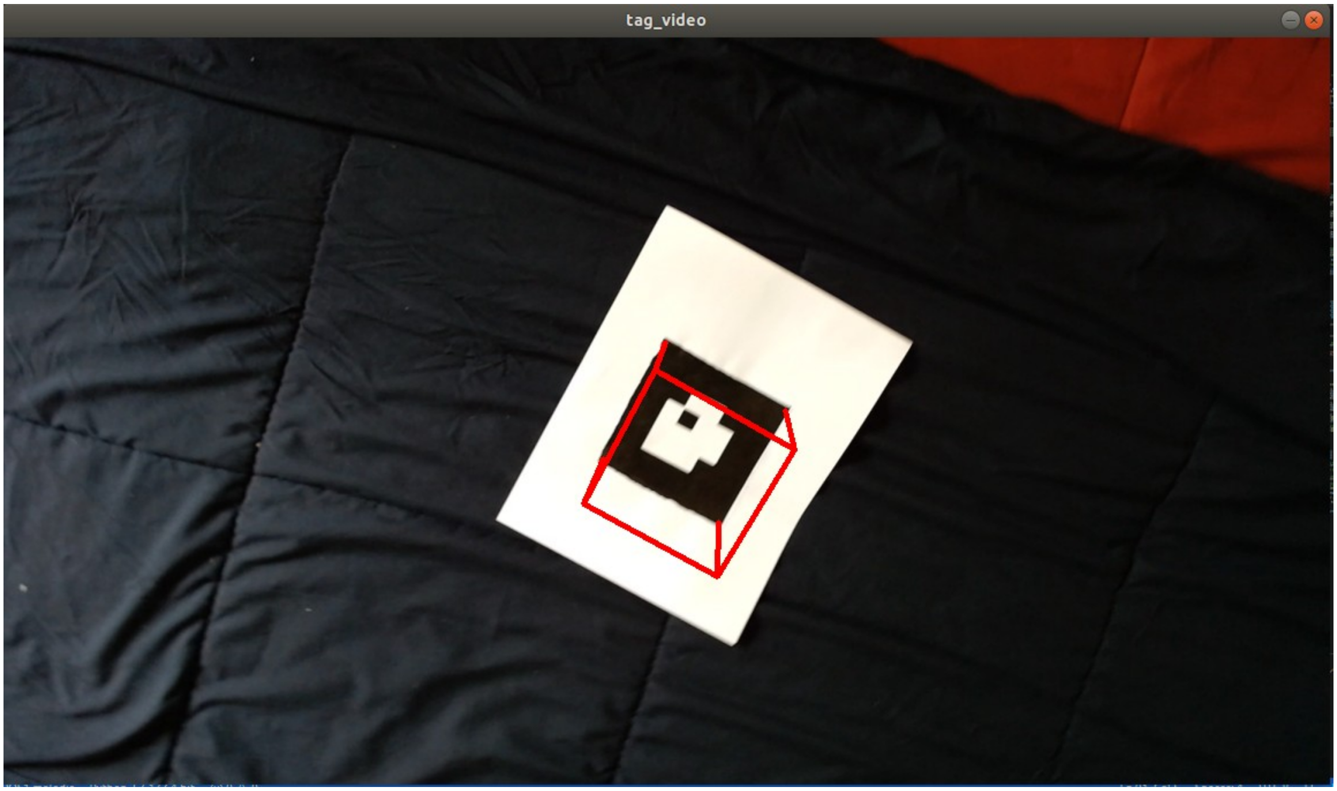
Question 2-B:

Method/Pseudo code:

```
def projMatrix(intrinsic_camera_matrix, Homography matrix):
        return  projection camera matrix

def findCorner(projection camera matrix):
        return corner of cube to be imposed on video frame

def drawCube(video frame, tag corner, cube corner):
        return video frame with cube drawn on it

def rescaleFrame(image, scale):
        return  image with reduced scale

def computeHMatrix(corner list):
        find four corner of blank image
        find the white paper corner and remove it from the corner list
        in the remaining corner list, corner corresponding to x_max, y_max, x_min and y_min is the
        corner of the tag
        take (0,0),(1,0),(0,1) and (1,1) as cube points
        construct A matrix to compute homography between tag corner and cube points.
        apply SVD
        find the eigenvector corresponding to minimum eigen value
        reshape eigen vector into (3,3)  and stored it in a matrix H
        divide each element by H(3,3)
        return H matrix
```

1. Normalize intrinsic camera matrix as per image scale factor
2. Read video frame by frame
3. Re-scale image and convert to gray scale
4. Process image to remove noise
5. Detect corner using shi-Tomashi corner detector
6. Find the white paper corner and remove it from the corner list
7. In the remaining corner list, Corner corresponding to x_max, y_max, x_min and y_min is the corner of the tag
8. Compute H matrix using computeHMatrix function
9. Compute projection matrix using projMatrix function
10. Find pixel location of corner using projection matrix and findCorner function
11. Draw the frame on each video frame using draw cube function

Result:



Extra Credit:

Network Architecture of DexiNed

1. DexiNed is a supervised deep neural network based on VGG16 architecture that perform edge detection.
2. The DexiNed architecture has six blocks acting like an encoder. Each block is a collection of smaller sub-blocks with a group of convolution layers. Skip-connection are used in this architecture which couple the blacks as well as their sub-block with each other.
3. 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.
4. These features are fused to generate a single edge-map at the very end of the network.
5. Unsampling network USNet is a conditional stack of two blocks. Each one of then consists of 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 USNe 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.

Dense Extreme Inception Network for Edge Detection

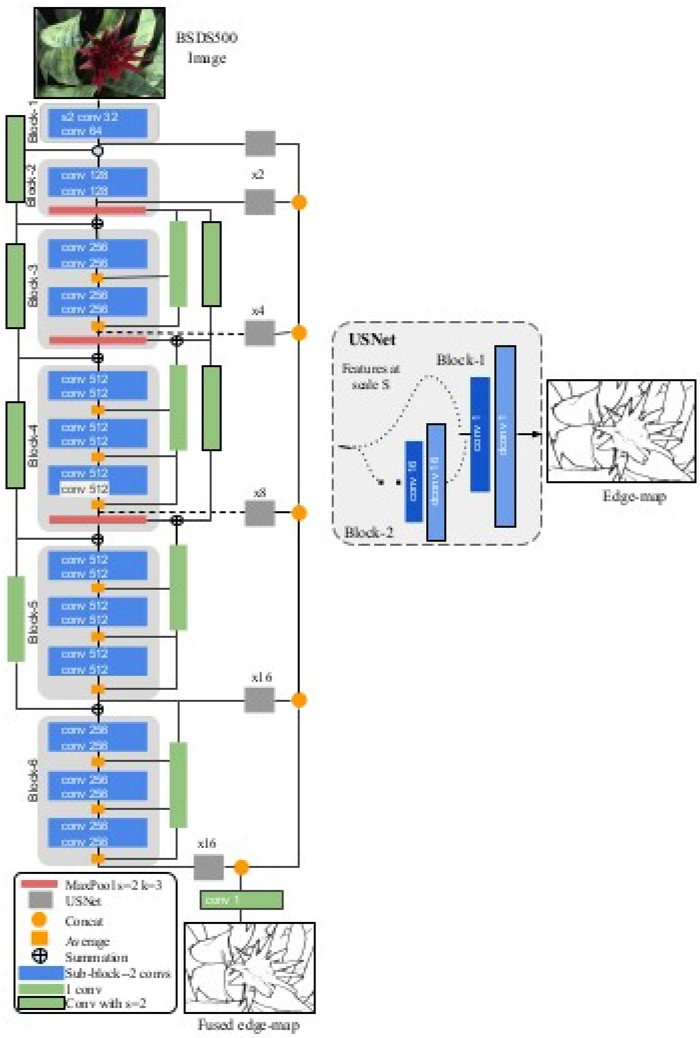
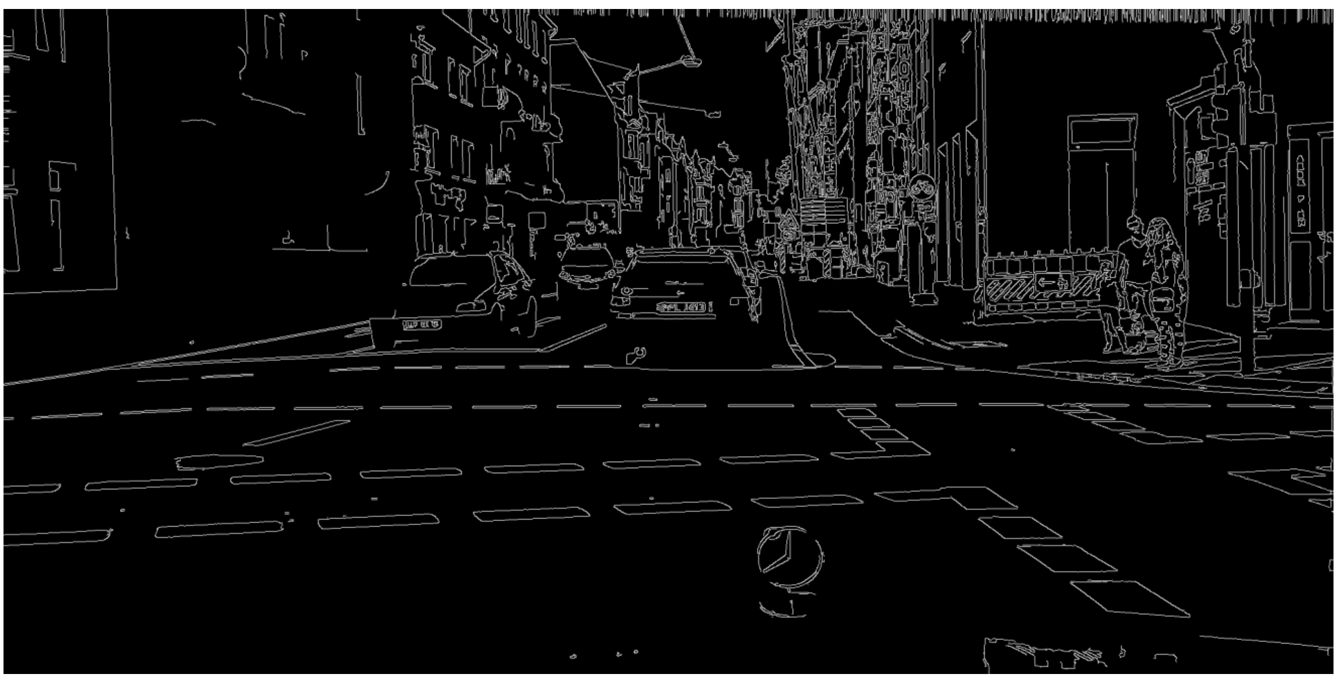


Figure 3: Flowchart of proposed architecture (DexiNed). It consists of two building blocks a *Dense Extreme Inception Network* and an upsampling Net (*USNet*).

Source: Dense Extreme Inception Network for Edge Detection by Xavier Soria , Angel Sappa,Patricio Humanante, Arash Arbarinia

Original image



Edge detection using canny edge detector

Edge detection using DexiNed

Comparison of Canny Edge Detector and DexiNed :

1. Edge visualization is easy in canny edge detection compare to DexiNed.
2. Edge pixel intensity is 255 in canny edge detection while in DexiNed it is 0.
3. Since DexiNed is a deep neural network, it can be trained to detect important edges and omit less important edges.
4. The edges detected by DexiNed are more continuous and do not abruptly break as opposed to Canny Edge.

Video link

Testudo image on April tag : https://youtu.be/LMh10FB7Dt4
Cube on April tag : https://youtu.be/8Ex9Ba9KA_k