PROJECT I

Perception for Autonomous Robots

Instructors:
DR. SAMER CHARIFA

Student:
RISHABH SINGH;
rsingh24@umd.edu

Semester:
SPRING 2022

Course code:
ENPM673

Contents

List of Figures

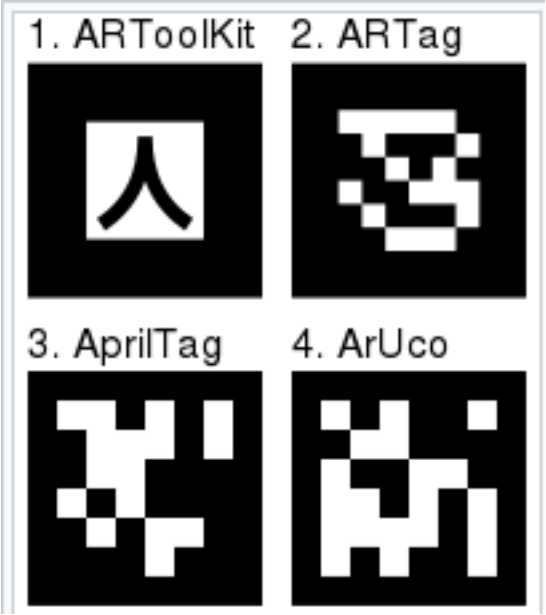


Figure 1: Comparison of some augmented reality fiducial markers for computer vision

1 Introduction

This project will focus on detecting a custom AR Tag (a form of fiducial marker), that is used for obtaining a point of reference in the real world, such as in augmented reality applications. There are two aspects to using an AR Tag, namely detection and tracking, both of which will be implemented in this project. The detection stage will involve finding the AR Tag from a given image sequence while the tracking stage will involve keeping the tag in "view" throughout the sequence and performing image processing operations based on the tag's orientation and position (a.k.a. the pose).

2 Part 1

2.1 Edge Detection using FFT

Problem Description: 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. Notice that you are expected to use inbuilt functions to calculate FFT and inverse FFT.

Pipeline followed: Problem 1(a)

Edges commonly occur on the limit between two unique areas in a picture. Edge detection is every now and again the initial phase in recouping data from pictures. Because of its significance, edge detection keeps on being a dynamic research region. In this project I have illustrated how edge detection works. Though the inbuilt function was allowed but I have written my own Gaussian filter code (Referred Matlab) to blur the images to remove very high frequency noise and some other

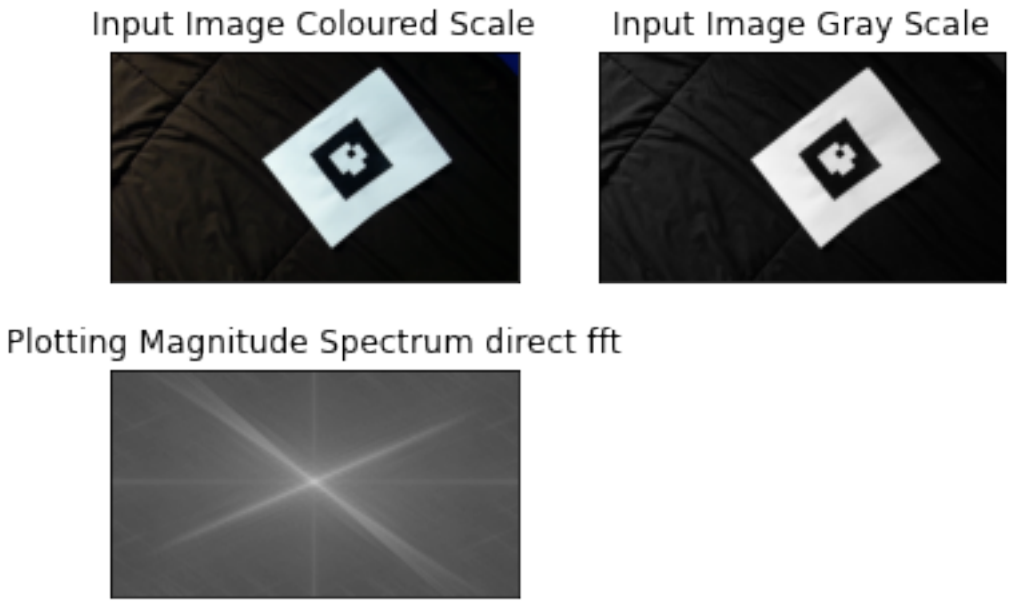


Figure 2: Image in Frequency domain

noises for accurate edge line detection. I have also tried to detect edges without using Gaussian filter and have laid down the comparison.

Steps:

1) Convert the image to gray scale in-order to keep only useful parts in the image and for better detection of edges. This is directly done while reading the image and passing the integer value 0 for the color flag.

2) To perform blurring to the image in order to reduce random noise from the image and smoothen it out for better edge detection.

Filters in image processing are just what the name suggests, Filter. They are typically a mask array of the same size as the original image which when superimposed on the original image, extracts only the attributes that we are interested in. As mentioned earlier, in an FFT transformed image, low frequencies are found in the center and high frequencies are scattered around, we can then create a mask array which has a circle of zeros in the center and rest all ones. Now when this mask is applied to the original image, the resultant image would only have high frequencies. This becomes quite useful as high frequencies correspond to edges in spatial domain.

3) Convert the image into frequency domain for ease in applying mask

$$MagnitudeSpectrum = 20 * log|ImageinFrequencyDomain| \quad (1)$$

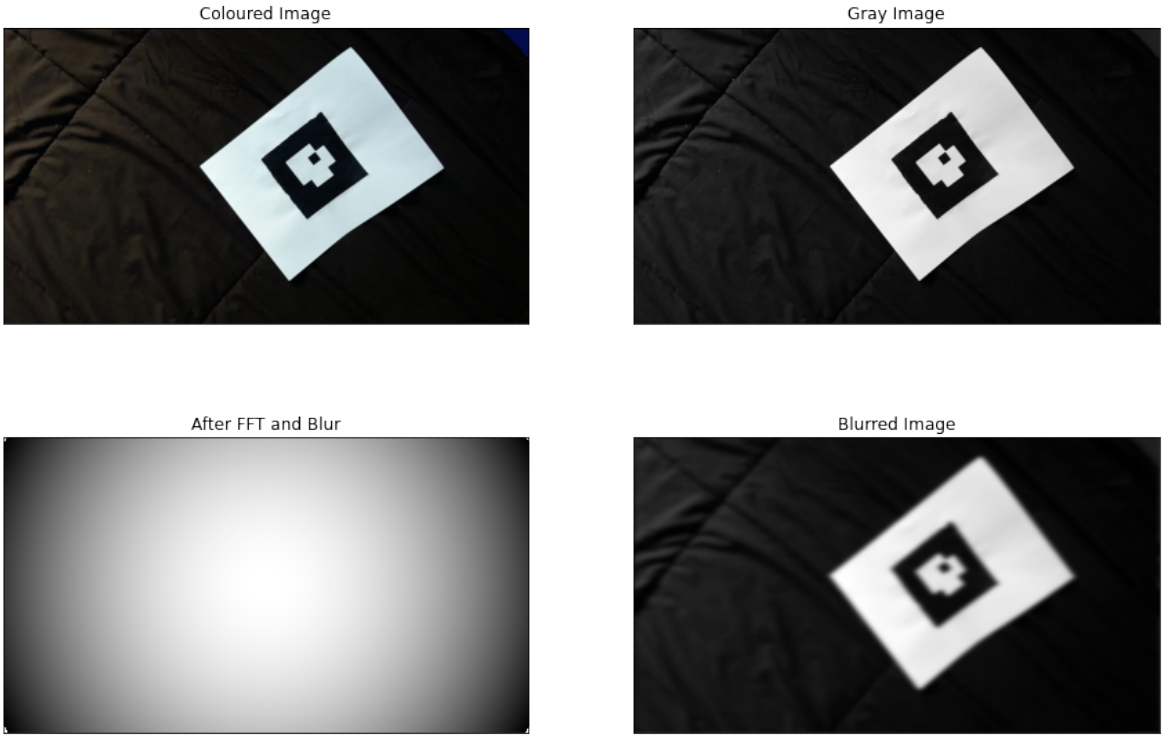


Figure 3: Blurred image after using custom Gaussian function

4) Multiply the converted image with the gaussian blur mask:

The Gaussian blur feature is obtained by blurring (smoothing) an image using a Gaussian function to reduce the noise level. It can be considered as a nonuniform low-pass filter that preserves low spatial frequency and reduces image noise and negligible details in an image. Gaussian blur the image to reduce the amount of noise and remove speckles within the image. It is important to remove the very high frequency components that exceed those associated with the gradient filter used, otherwise, these can cause false edges to be detected.

$$Gaussian(x, y) = (1/(2 * \pi * \sigma^2)) * \exp^{(-(x^2+y^2)/(2*\sigma^2)} \tag{2}$$

4) Now for our purpose we only considered the binary image, by thresholding.

5) Next part is creating a high pass filter for edge detection.

Reasoning: Detecting an edge in an image is of great use in the world of computer vision. Once we can extract edges in a image, we can use that knowledge for feature extraction or pattern detection.

Edges in an image are usually made of High frequencies. So what we need to do after taking a FFT (Fast Fourier Transform) of an image is, we apply a High Frequency Pass Filter to this FFT transformed image. This filter would in turn block all low frequencies and only allow high frequencies to go through. Finally, now if you take a inverse FFT on this filter applied image, you should

see some distinct edge features in the original image.

6) After removing low frequencies, converting the image back to spatial domain.

OUTPUT:

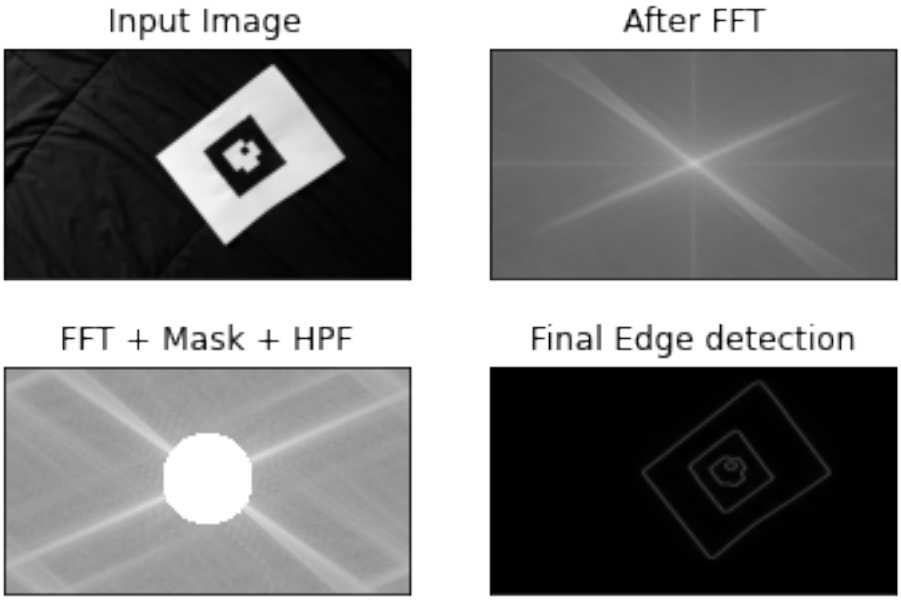


Figure 4: FFT edge detection

Problems faced:

1) It was a task to develop an understanding how the image behaves in frequency domain and what does it mean by very high frequency noise and high frequency noise.

2) Understanding how to Matlab's Gaussian filter is written. Kernel was taking too much of a time.

3) I tried the method of just using the High Pass Filter mask. It solved the problem in the same manner if we are using a binary image. But the good practise is to use gaussian blur to reduce noise and very high frequencies from the image. The reason I found is, once converted to binary the random noise is almost null.

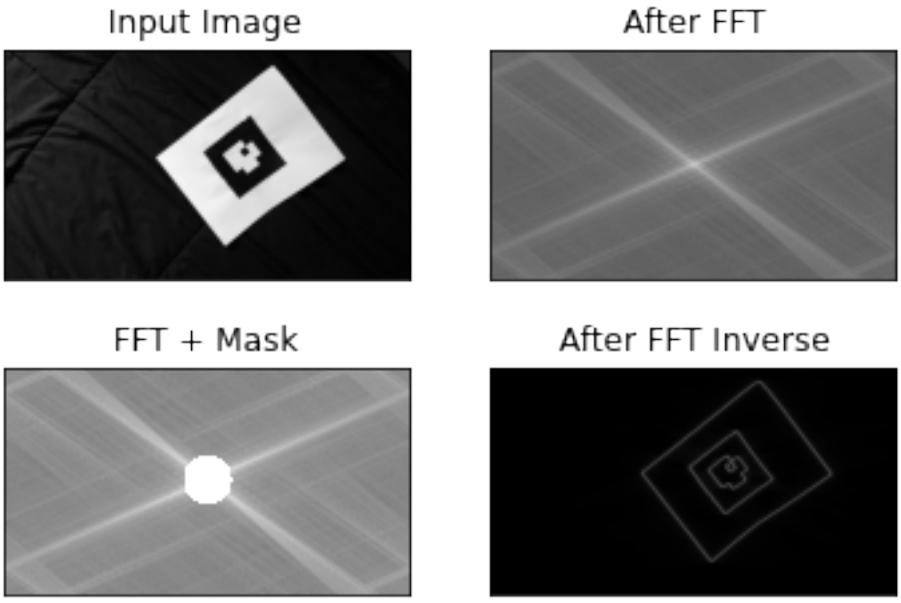


Figure 5: Output by applying only the High Pass Filter

2.2 ID and Orientation Detection

Problem Description: You are given a custom AR Tag image, as shown in 5, to be used as reference. This tag encodes both the orientation as well as the ID of the tag.

Pipeline Followed: Corner detection of the image whose edges are detected above

Theory: Corners are regions in the image with large variation in intensity in all the directions.

$$M = w(x,y) * \begin{bmatrix} Ix * Ix & Ix * Iy \\ Ix * Iy & Iy * Iy \end{bmatrix} \tag{3}$$

Here, Ix and Iy are image derivatives in x and y directions respectively. (These can be easily found using cv.Sobel()).

Then comes the main part. After this they created a score, basically an equation, which determines if a window can contain a corner or not.

$$R = det(M) - k(trace(M))^2 \tag{4}$$

where

det(M)=λ1*λ2

trace(M)=λ1+λ2

and λ1 and λ2 are the eigenvalues of M

So the magnitudes of these eigenvalues decide whether a region is a corner, an edge, or flat.

When mod R is small, which happens when $\lambda 1$ and $\lambda 2$ are small, the region is flat. When R<0, which happens when $\lambda 1$ is greater than $\lambda 2$ or vice versa, the region is edge. When R is large, which happens when $\lambda 1$ and $\lambda 2$ are large and $\lambda 1$ λ, the region is a corner.

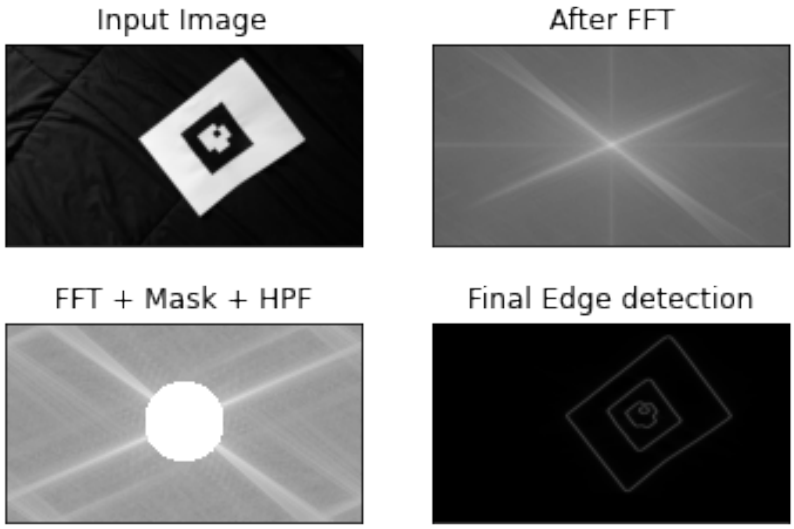


Figure 6: Edge detection theory

I have followed different methods to detect the tag id and orientation. Final method was the pipeline developed to solve problem 2.

Steps:

Step 1: Run the video and store all the frames

Step 2: Find all the corners using Shi-Tomashi or good features to track method. This needs to be calibrated to give optimum number of points which was checked for different frames and definite value was decided. There were various ways to move forward from here. The best way is to use find contour method but since it was not allowed to use, I had to find the next possible method. Then I went on to methods like scaling, corner detection for binary image but then a basic mathematical methodology was formed. Due to noise there were lot of extra corners detected, plus I wanted to build a general pipeline which should work on any video/image input. So, after detecting all the corners using cv2.goodFeaturesToTrack() from the image. All the possible combination of points were checked and the threshold was given for the distance between them as well as the difference between their centres (remember, ideally the centres of the rectangles should pass through the same point). A best possible set was collected out of this method.

Step 3: The best corners were selected and the homography matrix was created using a world frame which was the distance between the adjacent corners, when sorted. Now the homorgraphy is mapped from the camera frame to the world frame, the edge length of the tag being the frame size in the world.

Step 4: Now these corners gave lot of rectangles/polygons. A tag decoder was created, which runs the decoding through every closed loop. A check was kept for the id, the black padding and the orientation which is the right bottom corner of the tag has to be white. A grid checked of 8 X 8 was run in such a way that it detects the code only when the above condition is satisfied. Then by checking the orientation, the tags id was detected using the

$$\begin{bmatrix} {x_1}^{(w)} & {y_1}^{(w)} & 1 & 0 & 0 & 0 & -{x_1}^{(c)}{x_1}^{(w)} & -{x_1}^{(c)}{y_1}^{(w)} & -{x_1}^{(c)} \\ 0 & 0 & 0 & {x_1}^{(w)} & {y_1}^{(w)} & 1 & -{y_1}^{(c)}{x_1}^{(w)} & -{y_1}^{(c)}{y_1}^{(w)} & -{y_1}^{(c)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ {x_4}^{(w)} & {y_4}^{(w)} & 1 & 0 & 0 & 0 & -{x_4}^{(c)}{x_4}^{(w)} & -{x_4}^{(c)}{y_4}^{(w)} & -{x_4}^{(c)} \\ 0 & 0 & 0 & {x_4}^{(w)} & {y_4}^{(w)} & 1 & -{y_4}^{(c)}{x_4}^{(w)} & -{y_4}^{(c)}{y_4}^{(w)} & -{y_4}^{(c)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Figure 7: The SVD and homography (Homogenous solution)

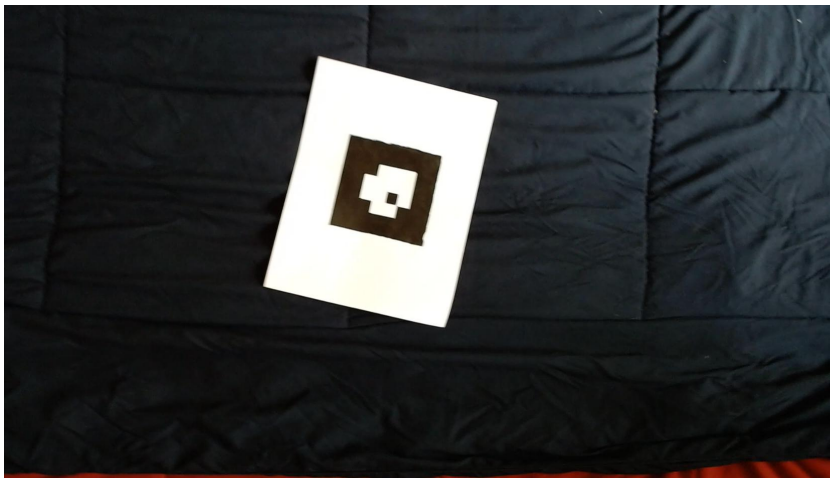


Figure 8: The frame used to decode the tag

Step 5: All the functions are coded, except the inbuilt SVD function to calculate the homography matrix.

Explanation of warping function and homography function: It was required to warp the image to the world frame. Which was basically done by creating a grid of all possible coordinates in the world frame, now by multiplying it to the inverse of the homography function to each coordinate gave us the coordinate in the image/camera frame. Now, it was straightforward to take the pixel values from these coordinates and copy them to the mapped/world coordinates. The output image came as the warped image of the AR tag.

Output:

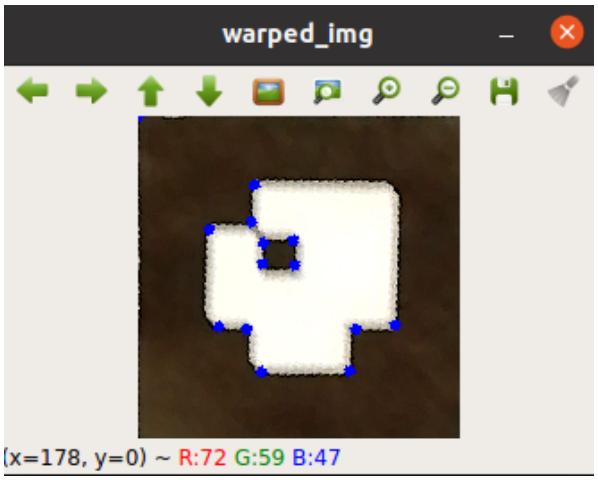


Figure 9: Warped image of the tag (Its the mirror image, which had to be compensated)

Output:

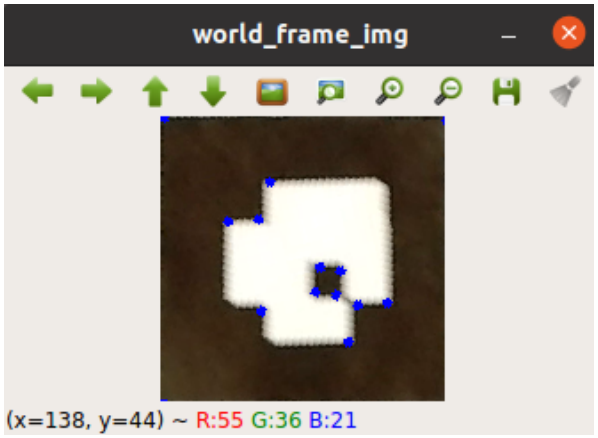


Figure 10: Warped image of the tag (Compensated the mirror image by dual warping)

The tag id found out was: 7 The orientation found out for this orientation was: 90

Problems:

1) Writing the custom functions for warping was a challenge because a deep mathematical understanding was required. I had to go through lot of references to come up with something that makes sense.

2) The warping function was giving out a mirror image for obvious reasons which took me some time to figure out but later I applied the warping to the same frame by getting a new homography matrix and giving the previous warped image as the input. Even the ID and orientation detected was wrong. 9

3 Part 2

3.1 Projecting Testudo on AR tag

After doing the previous part, the other two question were pretty much straightforward with respect to mathematics. The only challenge was to do it for all the frames so that the output should be a video.

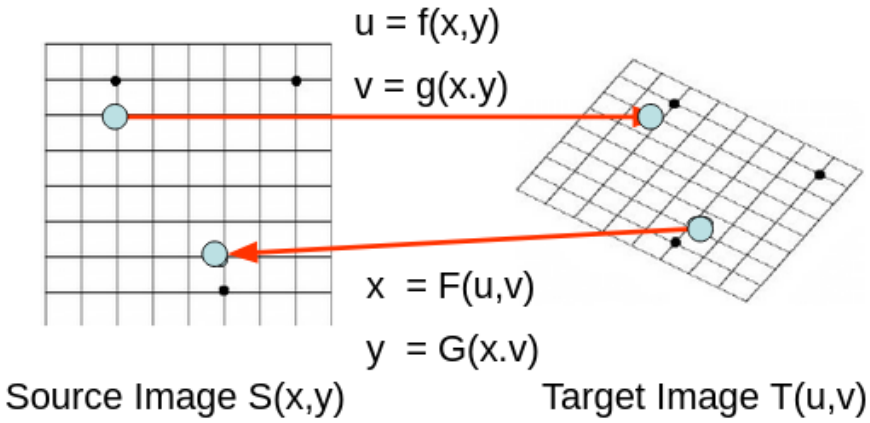


Figure 11: Warping from one frame to other

Pipeline followed: The warping function was recoded to make it modular for both forward and inverse mapping. While projecting the AR tag it needed only one image and the parameters were copied from camera frame to world frame, whereas it was just the opposite while projecting testudo image over the AR tag. The edge length in the world frame was kept same as the detected tag, now multiplying all the coordinates of the world frame (testudo) with the inverse of the Homography calculated for the same frame gave us the coordinates in the camera frame. So, the corresponding pixel values (RGB) was added to the frame or the image at that particular instant.

Output:

This is the link to the video: Link

Problems: The major problem occured when I was trying to do the forward warping, there were lot of gaps in the testudo image after projection. The solution I found was the inverse warping and then I checkd for interpolations and later on applied filter to smoothen it. This was not the

Figure 12: Testudo warped output with gaps

Figure 13: Testudo warped output at one of the frame

ideal solution but it gave a better output.

Figure 14: Testudo warped output at another frame

3.2 Projecting Cube on AR tag

Pipeline followed:

The cube to be projected is a 3D image. So, the homography matrix cannot be used here. Instead we had to get the projection matrix.

Step 1: Break the homography matrix into Extrinsic and Intrinsic parameters. Intrinsic matrix has already been provided.

Step 2: Now projection matrix, $P = K[R|T]$, where K is the camera instrinsic matrix. During 2D homography, extrinsic matrix is a 3X3 matrix, consisting of 2 rotation vector and 2 translational vector (since z is zero). Now, the third rotation matrix vector is calculated as cross-product of the other two rotation vectors. The scale factor λ is found as the average of the norm of first rotation vectors.

$$\lambda = \left(\frac{||\boldsymbol{K}^{-1}\boldsymbol{h}_1|| + ||\boldsymbol{K}^{-1}\boldsymbol{h}_2||}{2} \right)^{-1}$$

Where, h1 and h2 are first two column vectors of homography matrix. The final rotation vectors are calculated by multiplying them with the inverse of the scale factor. Using, all this formula a 3D rotation matrix and new translation vector is calculated to get the projection matrix. Now, we can map any 3D point in the world frame to the image frame. This was all taken care by the coded method for transformation.

Step 4: Now the coordinates of the cubes were defined as the edge size of the tag. Using projection point method, each of these points were projected to the world frame and lines were drawn between these points, which gave us the a blue colour cube as shown in 16.

Outputs:

Link for Video

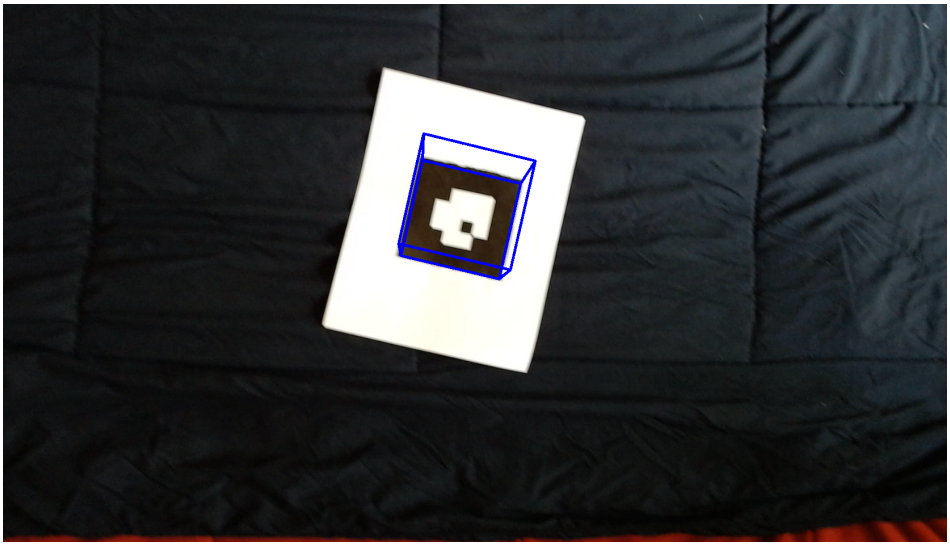


Figure 15: Projected cube in one of the frame

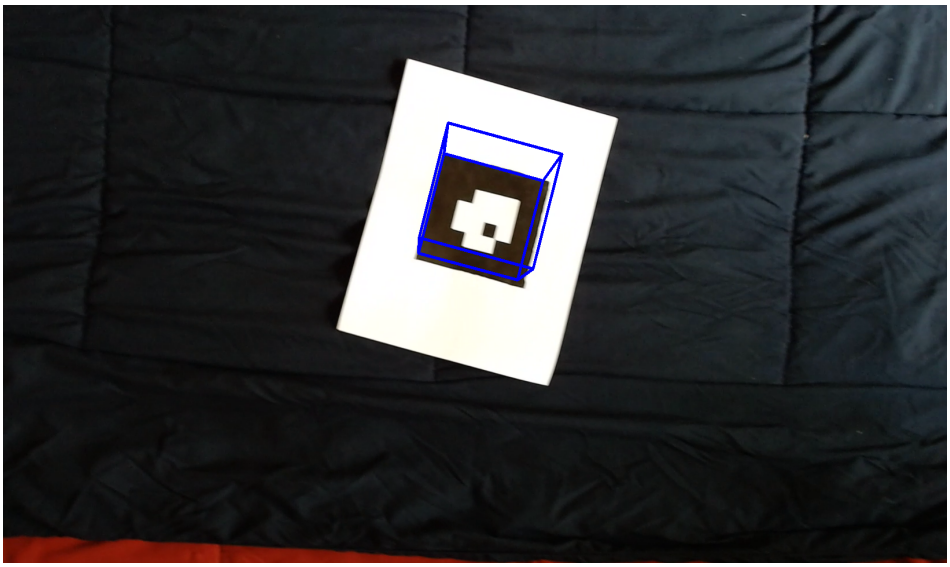


Figure 16: Projected cube in another frame

4 Dexined vs Canny study

4.1 Introduction

Edge detection is the basis of many computer vision applications. State of the art predominantly relies on deep learning with two decisive factors: dataset content and network's architecture. Most of the publicly available datasets are not curated for edge detection tasks. Here, Dexined offers a solution to this constraint. Edges, contours and boundaries, despite their overlaps, are three distinct visual features requiring separate benchmark datasets. Dense Extreme Inception Network for Edge Detection (DexiNed) is a proposed novel architecture, termed , that can be trained from scratch without any pre-trained weights. DexiNed outperforms other algorithms in the presented dataset. It also generalizes well to other datasets without any fine-tuning. The higher quality of DexiNed is also perceptually evident thanks to the sharper and finer edges it outputs. Dense Extreme Inception Network for Edge Detection (DexiNed)[1] is a supervised deep learning model that performs edge detection. Unlike other state-of-the-art convolution neural network based edge detectors, DexiNed has only a single training stage. Also, DexiNed can be trained from scratch with tuning parameters. In this report, the focus will be on performance of DexiNed against Canny [2] Edge Filter, and not about training the network. A trained DexiNed network will be used to find edges in an image from Cityscape [3] dataset and the result will be compared with Canny Edge filter.

4.2 DexiNed Architecture

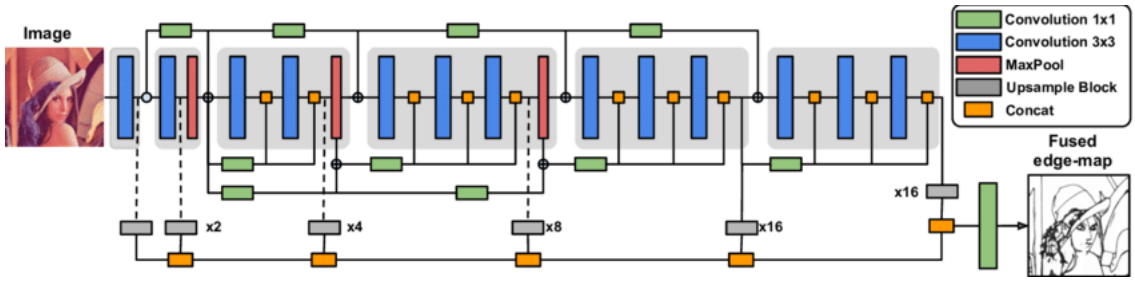


Figure 17: Architecture for Dense Extreme Inception Network [1]

DexiNed consists of 6 main blocks connected sequentially as shown in figure 17. The main blocks are connected to each other using 1x1 convolution blocks. Each main block consists of sub blocks made up of 3x3 convolution blocks and MaxPool which is a down-sampling strategy used in convolution neural networks. These sub blocks are interconnected by the output of the previous main block. The output of each main block is fed to an up-sampling block to get intermediate edge maps as shown in figure 18 and later combined using concatenation to give a fused edge-map. Up-sampling block, as name suggests increases the resolution of the image, it is converse of down-sampling techniques. The up-sampling block receives learned features as input and generates intermediate edge map by using stacks of convolutional and transposed convolutional filters, see 18.

4.3 Comparison with Canny Filter

On an image from the Cityscape[3] dataset, DexiNed and Canny filter are used to obtain edge maps as shown in figure 19 below. These results will be compared to the original gray-scale image

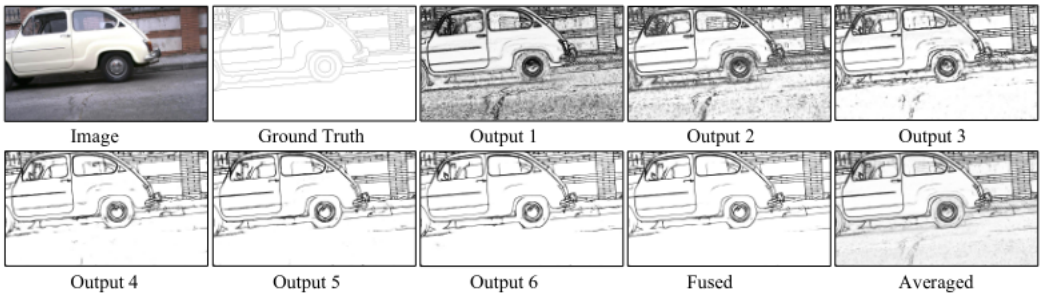


Figure 18: Edge maps from DexiNed [1]. The outputs of each upsampling block are labeled as output 1 to 6. Fused is the concatenation and fusion of the 6 outputs and average is the average of the 6 outputs.

using two parameters, namely Mean Square Error (MSE) and Structural Similarity Index (SSIM).

MSE is the most common estimator of image quality measurement metric. It is a full reference metric and the values closer to zero are the better. It is the second moment of the error. It has the same units of measurement as the square of the quantity being calculated like as variance[4]. Mean Squared Error (MSE) between two images such as g(x,y) and g^(x,y) is defined as shown in equation 5.

$$\text{MSE} = \frac{1}{MN}\sum_{n=0}^{M}\sum_{m=1}^{N}[\hat{g}(n,m) - g(n,m)]^2 \tag{5}$$

Structural Similarity Index Method (SSIM) is a perception based model. In this method, image degradation is considered as the change of perception in structural information. It also collaborates some other important perception based fact such as luminance masking, contrast masking, etc. The term structural information emphasizes about the strongly inter-dependant pixels or spatially closed pixels. These strongly inter-dependant pixels refer some more important information about the visual objects in image domain. Luminace masking is a term where the distortion part of an image is less visible in the edges of an image. On the other hand contrast masking is a term where distortions are also less visible in the texture of an image. SSIM estimates the perceived quality of images and videos. It measures the similarity between two images: the original and the recovered.[4]

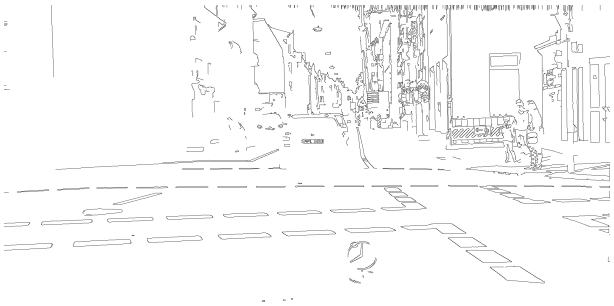
MSE and SSIM are calculated as shown in figure 20 between the edge maps and the original gray-scale image. It can be seen that Canny Edge filter performs worse in case of Mean Square error among the three while Dexined Average provides the least Mean Square error. Although, when comparing Structural Similarity index, the results are opposite, Canny Edge filter performs better than both DexiNed Average and DexiNed Fused. On visual observation of the images it was noted that the Canny Edge filter missed a lot of important edges, for example the building on the left and the outer edges of the cars ahead. These edges are captured by DexiNed and can be seen in both DexiNed Average and DexiNed Fused. On the other hand, Canny filter has captured minute edges in some places like the number plate of the car ahead and the bicycle road sign on the right side of the road. These fine edges are missing in both the DexiNed results. Hence, the higher SSIM in Canny can be explained. Overall, while Canny captures minute edges in areas of good contrast, it misses important outer edges which are necessary for most perception applications. DexiNed performs well in both regions of low and high contrast. Even though DexiNed does not provide detailed edges, it reliably provides the major edges in the image, hence making it the superior edge detection method in

(a) Original Image

(b) Gray-Scale Image

(c) DexiNed Fused Edge Map

(d) DexiNed Average Edge Map



(e) Canny Edge Map

Figure 19: Edge Detection using DexiNed and Canny filter on an image from Cityscape Dataset

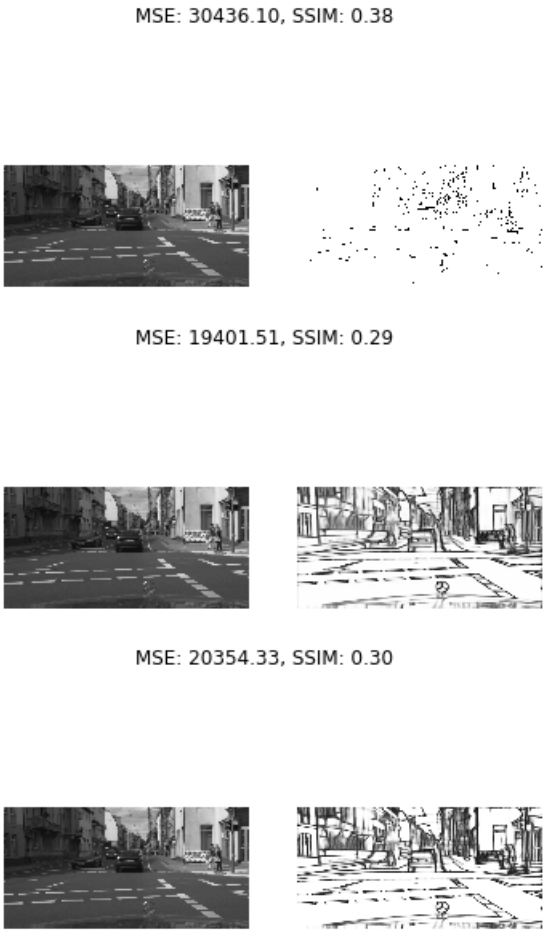


Figure 20: The result of comparing Canny filter, DexiNed Average and DexiNed Fused edge maps to the original gray-scale image using Mean Sqaure Error(MSE) and Structural Similarity Index(SSIM)

most perception applications. Canny can perform better than DexiNed in some applications, namely when the subject has good contrast and the application needs detailed edges.

5 Bibliography

[1] Xavier Soria, Edgar Riba, and Angel Sappa. Dense extreme inception network: Towards a robust cnn model for edge detection. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1912–1921, 2020.

[2] Canny edge detection.

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[4] Umme Sara, Morium Akter, and Mohammad Shorif Uddin. Image quality assessment through fsim, ssim, mse and psnr—a comparative study. *Journal of Computer and Communications*, 07(03):8–18, 2019.