ENPM673 – Perception for Autonomous Robots

Project 1

Pratik Acharya

117513615

March 07, 2022

Problem 1 - Detection

Problem 1.a) AR Code detection:

Ans:

Fourier Transform is a mathematical transform that decomposes a function that is present is time domain, to a function in frequency domain.

This transform can be used in Computer Vision for image processing routines such as blurring, edge detection, thresholding, and others. In this question, we have used the tranform to detect edges in the image.

OpenCV has inbuilt functions that can be used to perform fourier transform in an image. 2D discrete fourier(DFT) transform is used to find the frequency domain of the image, after which fast fourier transform(FFT) is used for calculations on DFT. After the fourier transform is calculated, the resulting image is shifted such that the lower frequencies are mapped at the middle of the frame and the high frequencies are mapped to the outer side. After shifting, a mask is applied, where a circular shape is removed from the center of the fft shifted image, which correspond to the lower frequencies. This acts as a high pass filter and will retain all the edges in the image, while discarding other details. The masked image is then passes through an inverse fourier transform function which restores the image.

The resulting image from fft edge detection is :

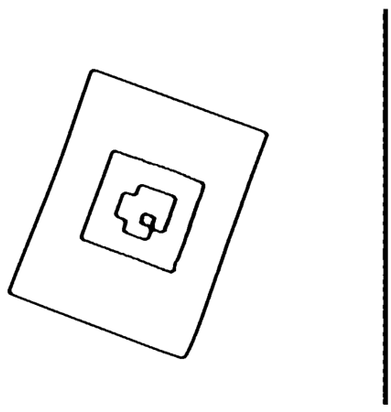

Figure 1: FFT Result

Problem 1.b) Decode custom AR tag:

Ans:

To find the corner points of the AR Tag, the image was processed multiple times. First the image was changed from BGR to Grayscale, and was later passes through a gaussian blur filter to smoothen the image. Erosion and Dilation was applied to the image to further clarify the image after which CLAHE filter was applied to improve the contrast in the image.

After this initial processing, fourier transform was applied to the image as explained in 1.a), which enhanced the edges of the image and removed unnecessary details. This image was further passed through a gaussian blur filter to remove edges which were not connected.

This image was used to detect corners. Shi-Tomasi corner detection algorithm was used for detecting corners. As this detection also included corners of the paper, and false detections, the paper corners were found and only the points included in the polugon enclosed by this region were retained. From these retained points, 4 extreme corners were calculated, which would be the four edges of the tag.

Using this four corners, homography matrix was calculated which transformed the tag into a flat image. This image was then divided into 8x8 grid. The outer padding consisting, of 2 rows of black grids, were removed and from the inner 4x4 grid, the four corners were detected to find the rotation of the tag. The rotation provided the order of the decoded message in the center 2x2 grid of the image.

The following image was used to detect the tag:

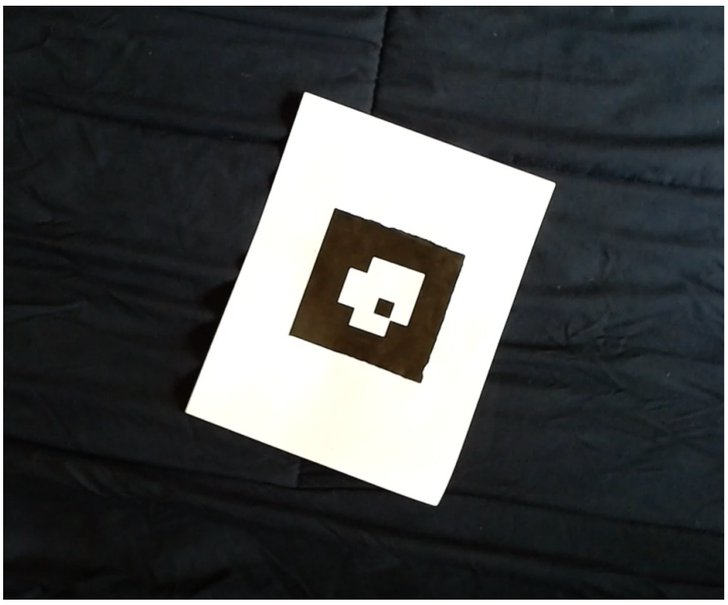

Figure 2: Original Image

And the extracted tag was,

Figure 3: Tag

The grid looked as such:

The decoded message was ”0111”, which is binary for 7.

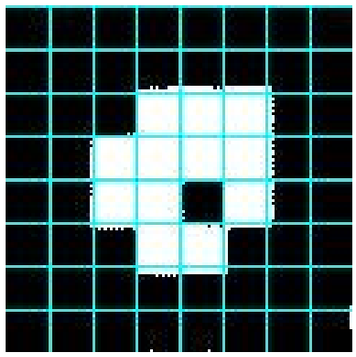

Figure 4: Grid

Problem 2 – Tracking

Problem 2.a) Superimposing image onto Tag:

The major challenge of this part was the inclusion of video as the input. As the video was blurry, the corner detection pipeline would sometimes fail to detect all the edges of the tag. To overcome this, whenever a missing corner was detected, the corners detected in the previous frame were used for homography.

After detecting the corners, as explained in 1.b), the rotation of the tag was figured out. This rotation was applied to the destination corners(Tag Corners), to change the order of the corners. This ensured that the testudo image was rotated according to the tag. After calculating the homography from the testudo corners, to the tag corners, the code iterated through all the pixels from the testudo image and found it's position on the original tag video.

Each pixel from the original, corresponding to the pixel from the testudo image, was replaced, which placed the testudo image on the given image.

Figure 5: Testudo 1 projected on Tag

Problem 2.b) Placing a virtual cube onto Tag:

To start, the cube was defined as an array consisting 8 elements, which were the 8 vertices of the cube. According the frame of reference of the camera, the z axis was negative, directing outward from the plane of the paper.

Figure 6: Testudo 2 projected on Tag

To calculate the projection matrix, first the homography matrix is calculated. Then each of the columns of the homography matrix is isolated, to get three vectors, $h1, h2, h3$.

Then the scaling factor is calculated as such,

$$\lambda = \left(\frac{(||K^{-1}.h1|| + ||K^{-1}.h2||)}{2} \right)^{-1}$$

For computing the $\tilde{B}$, we use :

$$\tilde{B} = \lambda K^{-1} H$$

Later, according to the sign of the determinant of $\tilde{B}$ matrix, the B matrix is calculated as such:

$$B = \tilde{B}(-1)^{|\tilde{B}|<0}$$

Each row from the B matrix is isolated as $b1, b2, b3$, and is used as follows to generated the transformation matrix:

$$\begin{aligned} r1 &= b1 \\ r2 &= b2 \\ r3 &= b1 \times b2 \\ t &= b3 \end{aligned}$$

Using the above transformation matrix, we calculate the projection matrix P as follows:

$$P = K[R|t]$$

This projection matrix, of dimension 3×4 is then multiplied by each vertice of the cube, by adding a padding of 1 to make it a 4×1 column, matrix, which then gives a 3×1, column matrix, containing the x, y and z coordinates of the destination point. To make it a

projection for 2D surface, the first two elements are divided by the last element, and the x and y coordinates are found.

These coordinates represent the position of the cube in the world frame, which can be represented in the original video.

Figure 7: Cube Projected on Tag

Dexined

Dexined is an edge detection framework based on a supervised deep learning. It consists of two networks which are 'Dexi' and 'USNet'.

Dexi(e dense extreme inception network) consists of 6 blocks, which are further divided into sub-blocks. Each of the sub-block contains two convolutional laters, with a kernal size of 3×3. Each convulational layer is followed by batch normalization and ReLU. Dexi gets it's input as an RGB image and outputs feature maps. As convolutional layers decrease the size of the image, the some of the features can be lost, which can be avoided by introducing parallel skip-connections.

The USNet (upsampling network) is a conditional stack of 2 blocks, which each consisting of a convolutional and a deconvolutional layer. The block 2 is activated only when scaling is required for the feature map inherited from dexi. The USNet, after upscaling the image, provides with an output which is the edge map of the image.

For the loss function, DexiNed is formalized as a regression mapping function.

To train the algorith, BIPED(Barcelona Images for Perceptual Edge Detection) dataset was used, which is a labelled edge detection dataset. Testing was completed by using various benchmarl datasets which are common in edge, contour, and boundary detection, such as MDBD, BSDS500, BSDS300, NYUD and others.

For comparison, the canny edge detector was used on an image, and DexiNed was also used to find the edges.

As we can see from the image, even though Canny provides sharper and thinner edges, they are discontinuous and alot of edges are missed. Whereas DexiNed provides better, continuous edges which can be made thinner using erosion.

Figure 8: Original Image

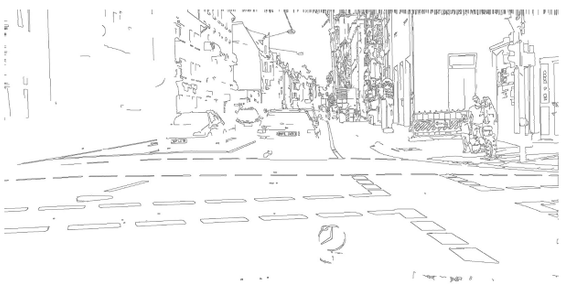

Figure 9: Canny Edge Detector

Result Video

The result videos can be found in the following Google Drive

ENPM673 Proj 1 Videos.

Code

The code is submitted in the ELMS portal and is also available in the following GitHub repository

ARTag-Detection-Projection.

References

Dense Extreme Inception Network for Edge Detection

Figure 10: Dexined