ENPM 673: Perception for Autonomous Robots - Project 1

Aswath Muthuselvam
aswath@umd.edu

26th Feb 2022

1 Problem 1

1.1 A) AR Code detection

A robot need to make sense of it's surroundings and able to localize itself with respect to it's surrounding environment. Problem 1a involves detection of AR Code. An AR Code is a marker that is used as an easy method of perceiving the environment by using simple algorithms rather than having to understand the entirety of the scene.

First, the sample video that is provided is relatively easier to process, with not many objects present in the scene. The sheet of paper that has AR Tag printed on it, can be located with performing image filtering techniques. The original image is taken and FFT(Fast Fourier Transform) is performed. Then, the zero of the frequency map is shifted to the center of the image. A circular Mask filled with value of zero is created, the values outside the circle is given as 1. The shifter frequency map is multiplied with the Mask to remove the low frequency components. The remaining high frequency values are converted back into an image by using Inverse-FFT Transform. Now the output of this operation provides us with only the edges of the paper and the AR Tag.

Finally, the AR Tag is warped to a square of size 128x128, the AR Tag is brought form the image space to a custom Top down - birds eye view.

The detection of this tag involves multiple steps:

1. Pass the image through High Pass filter
    1. Take Fourier Transform
    2. Frequency shift
    3. Prepare a mask
    4. Apply the mask to filter the low frequency components.
2. Apply inverse Fourier transform.
3. Apply FFTShift
4. Convert the image to binary and dilate the image.
5. Apply canny edge detection to make the the ARcode slightly round
6. Use Blob detection to detect the near round arcode
7. Take a template of the ARcode in top down view position
8. Use SIFT feature matching on the blob and the template ARCode image to find the homography.
9. Detect blob

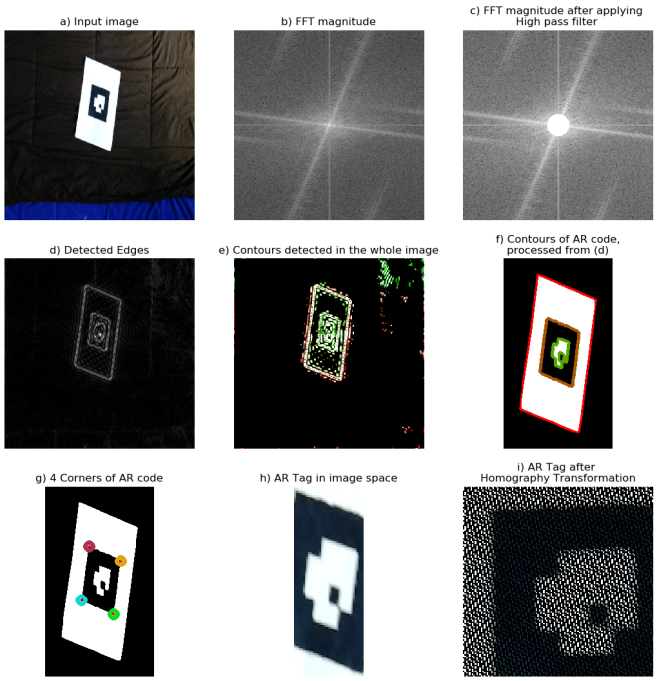

Figure 1: AR Tag localization with FFT and Homography

The homography transformation matrix is given by:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

1.2 B) AR Code Decoding

An AR Code has two orthogonal protrusions on one corner. Using this feature, the orientation of the AR Code can be determined. The AR Code decoded by splitting the image into a 8x8 Grid and summing the pixel intensities in each grid.

- The square image is converted to greyscale image and applied binary thresholding.
- The tag is split into 8x8 region, but our region of interest is only the middle 4x4 matrix which contains the id and the orientation of tag.
- Black is considered as 0 and white is considered as 1.
- In case of the white block being present at the bottom left corner, the orientation of the tag is bottom left.

- The innermost 2x2 matrix can be located and used for decoding based on least significant to the most significant bit.

- The orientation of tag is used in superimposing the testudo image on the tag.

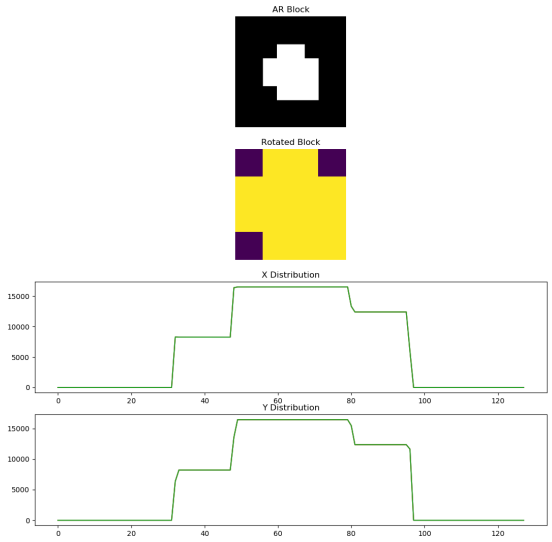

Figure 2: Decoding the AR Code

2 Problem 2

Figure 3 shows the projection of Testudo image on top of the AR Tag.

- Firstly, the image is resized to the dimension of 200 x 200 to match the world coordinates.

- A "changeOrient" is defined to reorient the tag. The image, according to the orientation of the tag is rotated by using this function.

- After transformations and inverse homography, the tag in the image is warped to form a black background

- The function cv2.bitwise_or is used to superimpose the testudo image on the AR tag.

2.1 A) AR Testudo Projection

Figure 3: Projecting the Testudo image on the ARTag

2.2 B) AR Cube Projection

Figure 4 shows the detection and projection of virtual cube with blue edges on top of the virtual tag. The cube is projected along the normal to the Tag.

The steps to compute projection matrix is given as follows:

- Define h1,h2,h3 as the 3 columns in the homography matix H
- Compute scale factor $\lambda = \left[\frac{(K^{-1}h1 + K^{-1}h2)}{2}\right]^{-1}$
- $\tilde{B} = \lambda K^{-1} H$

- $B = \tilde{B}(-1)^{|\tilde{B}|<0}$ i.e. $B = -\tilde{B}$ if B is negative, else $B = \tilde{B}$. b1,b2,b3 are defined as column vectors of B.

YouTube link

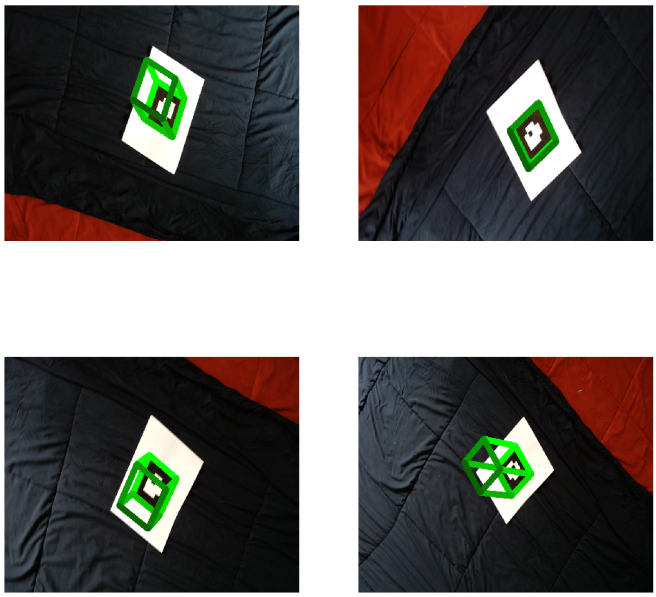

Figure 4: Projecting the AR Cube

3 Extra Credit

Two types of Edge detection are performed on the Cityscape image show in Figure: 5. The first method is by using Canny edge detection and the second is by using DexiNed. DexiNed(Dense EXtreme Inception Network for Edge Detection) is a CNN based Edge detector.

Contrasting with many CNN type architectures, all of them has a feature extraction backbone that fed their multi-layer feature outputs to corresponding upsampling layers, while DexiNed used a more robust Dense Extreme Inception architecture with residual style skip connections to extract and propagate far richer edge feature information to deeper layers in the network.

• They all use an upsampling layer to reproduce scaled intermediate edge maps at different layers within the network. The multi-level intermediate maps are then fused together to create a final output.

• DexiNed uses depthwise separable convolutions and extracts more edge feature information which is learned in deeper layers of the network.

• The accuracy of the results of each model was also mainly impacted by how robust it's feature extraction backbone was. DexiNed which used a dense extreme inception network performed better than neural networks with backbone architectures such as ResNet-50 and VGG-16. From this observation, it is concluded that using a robust feature extractor is crucial in producing accurate edge maps.

• In summary, the above-listed edge detection architecture share similar traits. They have a rich feature extractors and finally upscaled near the output layer.

Figure 5: Canny Edge detection

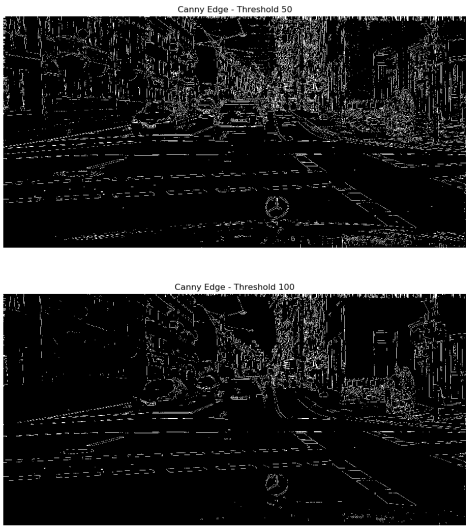

Figure 6: Canny Edge detection

Figure 7: Edge Detection with DexiNed