ENPM673 - Perception for Autonomous Robots Project 1

Saurabh Palande (UID: 118133959)
Masters in Robotics Engineering
University of Maryland College Park
Email: spalande@umd.edu

***Abstract*—This project focuses 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 are implemented in this project. The detection stage involves finding the AR Tag from a given image sequence while the tracking stage involve keeping the tag in "view" throughout the sequence and performing image processing operations based on the tag's orientation and position.**

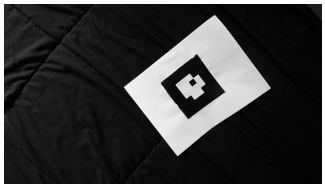
I. Problem 1 - Detection

A. Problem 1A

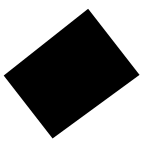
In this part, the AR tag is detected using the Fast Fourier Transform(FFT). When we apply FFT to an image, it converts the image from spatial domain to frequency domain.Low frequencies are situated towards the center of the image and high frequencies scattered around.
Steps to detect AR tag using FFT:-

1) Convert to grayscale and blur the image: First step is to convert the image to gray scale and then blur it by using a low-pass filter to reduce the amount of noise and detail in an image. By blurring an image prior to applying techniques such as edge detection reduces the amount of high-frequency content, such as noise and edges.

2) Generating mask: The next step is to generate a mask to remove the background and the white paper surrounding the AR tag. To generate the mask, first the corners of the white paper are found. Then a mask is created using cv2.fillPolly with these 4 corner points where the region inside these 4 points is black and the background is white. The black region is dilated so that it entirely covers the region inside the white paper. The input image and the mask generated is shown in Figure 1.



(a) Input Image



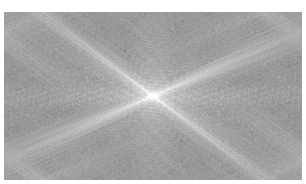
(b) Mask

Fig. 1: Input image and its corresponding mask

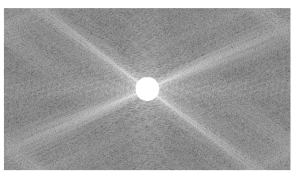
3) Adding the mask to the thresholded and blurred image: The mask is then added to the thresholded and the blurred image. After adding, the black background in the original image becomes white and the region inside the white paper remains the same. So now we have an image where only the AR tag is visible on a white background. The masked image is shown in Figure 2.

Fig. 2: Masked image

4) Applying FFT and inverse FFT: FFT is applied to the masked image to convert it from spatial domain to frequency domain to find the edges. After taking FFT of the masked image, a High Pass Filter (a mask array which has a circle of zeros in the center and rest all ones) is applied to this FFT transformed image. This filter would in turn block all low frequencies and only allow high frequencies to go through. Finally, inverse FFT is applied on this filter applied image to see the distinct edges of the AR tag. The FFT of the image and the FFT after HPF is shown in Figure 3. The final output is shown in Figure 4.



(a) FFT image



(b) FFT after HPF

Fig. 3: FFT and FFT after high pass filtering

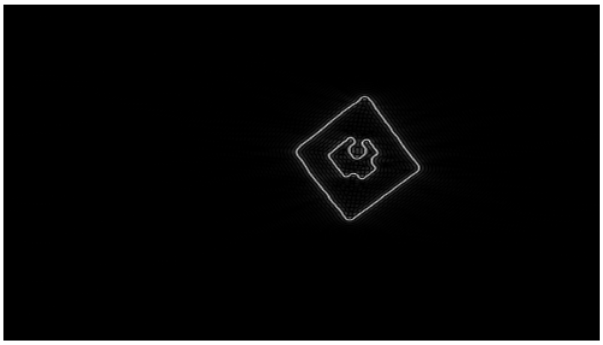


Fig. 4: Detected AR tag using FFT

B. Problem 1B

The following are the steps to decode the orientation and the ID of the reference AR tag:

1) The tag can be decomposed into an 8 x 8 grid of squares, which includes a adding of 2 squares width (outer black cells in the image) along the borders. This allows easy detection of the tag when placed on any contrasting background.
2) After the borders are cropped, take the median of the values in the 4 corner grids and check for a high value. Only one corner of the AR tag should have a high value. If more than one corner has a high value it is discarded
3) The tag is rotated till the bottom right corner value become high and the angle by which it is rotated becomes the orientation of the AR tag.
4) Lastly, the inner-most 2 x 2 grid (i.e. after removing the padding and the orientation grids) encodes the binary representation of the tag's ID, which is ordered in the clockwise direction from least significant bit to most significant. So, the top-left square is the least significant bit, and the bottom-left square is the most significant bit.

The orientation of the reference AR tag is 0 and the ID is 15.

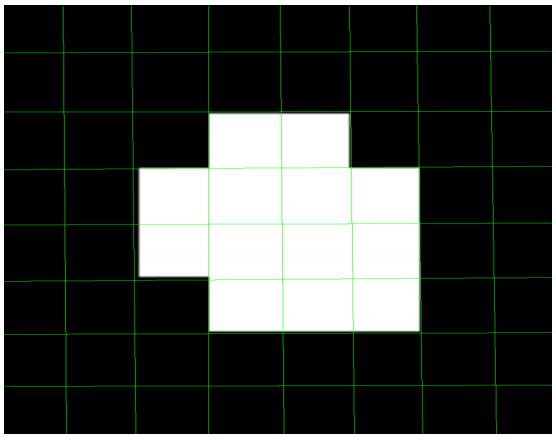


Fig. 5: Reference AR tag with grids

To decode the tag in the video frames the following steps are followed:

1) Detect the tag: The first step is to detect the tag in the video frames. To detect the tag the steps mentioned in Setion A.1, section A.2 and section A.3 are followed. Once we have the masked image, the 4 corner points are of the AR tag are extracted using the minimum and maximum condition on the x and y indices of the black pixel.

2) Warp the tag: Next step is to warp the tag so that it can be decoded. Once the 4 corners of the tag are found, they are stored in the order $[top-left, top-right, bottom-right, bottom-left]$. After that, homography between these 4 points and a planar set of reference points is computed using the below mentioned steps. Each pair of corresponding points gives us two linearly-independent equations; eight linearly-independent equations are formed by using four pairs of corresponding points. A homography has eight degrees of freedom, and is therefore fully determined by at least four correspondences. The homography matrix can be solved by writing these eight equations in matrix form. The A matrix is

$$\begin{bmatrix} {x_1}^w & {y_1}^w & 1 & 0 & 0 & 0 & -{x_1}^c{x_1}^w & -{x_1}^c{y_1}^w & -{x_1}^c \\ 0 & 0 & 0 & {x_1}^w & {y_1}^w & 1 & -{y_1}^c{x_1}^w & -{y_1}^c{y_1}^w & -{y_1}^c \\ : & : & : & : & : & : & : & : & : \\ {x_4}^w & {y_4}^w & 1 & 0 & 0 & 0 & -{x_4}^c{x_4}^w & -{x_4}^c{y_4}^w & -{x_4}^c \\ 0 & 0 & 0 & {x_4}^w & {y_4}^w & 1 & -{y_4}^c{x_4}^w & -{y_4}^c{y_4}^w & -{y_4}^c \end{bmatrix}$$

The h matrix is

$$\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}$$

We can calculate the homography using the equation 1. The solution to this equation can be found out using SVD.Decompose matrix A via SVD: [U, S, V].The elements of homography, h, are obtained from the last column of V. Since h_{33} should be 1, V should be normalized using v_{99} :

$$Ah = 0_{8*1} \tag{1}$$

After computing the homography matrix inverse warping is done to obtain the AR tag for decoding. The warped AR tag is shown in Figure 6

Fig. 6: Warped AR tag

3) Decode the tag: The same steps as mentioned in section B are used to decode the warped tag. The four corners of the decoded AR tag and the orientation and the Tag ID for one of the frames is shown in Figure 7. The entire video of the detected and decoded AR tag can be found here.

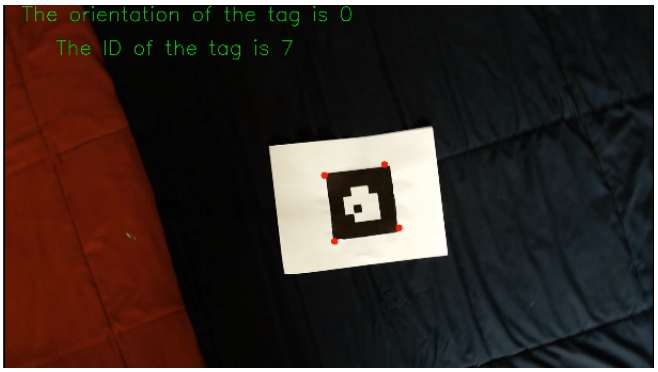


Fig. 7: Detected and decoded AR tag

II. Problem 2

1) Problem 2A: In this section a template image (Testudo) is warped on the AR tag using the Homography matrix. Once the 4 corners of the AR tag are detected using the steps mentioned in the above section, we compute the homography between the AR tag corners and the testudo image corners (the image is resized to (320,320)). Once the homography matrix is computed, the testudo is warped on the AR tag by using inverse warping. Based on the indices or pixel locations of the template image we calculate the pixels to be replaced in the original image by multiplying the template image indices by the homography matrix. Then the template image pixels are copied into the corresponding pixels of the original image as shown in Figure 8.

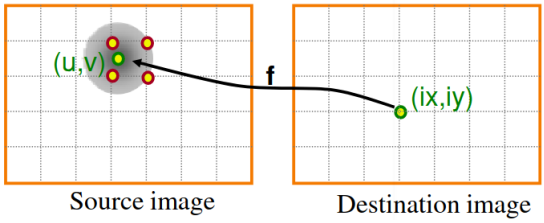


Fig. 8: Inverse Warping

After copying the pixels of the template image into the original image we get the testudo in place of the AR tag on the image frame. One such image with superimposed Testudo is shown in Figure 9.

The entire video of the Testudo superimposed on the AR tag can be found here.

Fig. 9: Superimposed Testudo

2) Problem 2B: To place a virtual cube on the AR tag we first need to calculate the projection matrix from the camera intrinsic matix and the homography matrix. The point $x(w) = [x(w), y(w), z(w), 1]^T$, expressed in world coordinates, can be converted to a point in the camera's image plane $x(c) = [x_(c), y_(c), 1]^T$. This relation is given by the projection matrix P The steps to calculate the projection matrix are as follows:

1) Detect four corners of the marker in the input image captured by camera.
2) Determine the world coordinates of the corners and compute the homography between the detected corner points in the image (pixel coordinates) and the corresponding points in the reference (world) coordinate system.
3) Find the R and t matrix using the below equations: Let us assume a new matrix $B = [r1, r2, t]$

$$\lambda H = K\tilde{B} \tag{2}$$

$$\tilde{B} = \lambda K^{-1} H \tag{3}$$

After computing $\tilde{B}$ if its determinant is less than zero, the matrix $\tilde{B}$ is multiplied with -1 because it is assumed that the object is infront of the camera. This gives us the matrix B. Now, to determine the scaling factor we use the following equation

$$\lambda = \left(\frac{||K^{-1}h_1|| + ||K^{-1}h_2||}{2}\right)^{-1} \tag{4}$$

All elements of the rotation and translation matrix are computed as following: $r_1 = \lambda b_1, r_2 = \lambda b_2, r_3 = r_1 \times r_2, t = \lambda b_3$.

4) Constuct projection matrix,

$$P = K[R|t] \tag{5}$$

Now that we have the projection matrix we can convert any point in homogenous world coordinate to a coordinate in

camera image plane. We have the bottom 4 corners of the cube. The top four corners of the cube in the world frame are $[[1, 0, -1, 1], [1, 1, -1, 1], [0, 1, -1, 1], [0, 0, -1, 1]]$. Using the projection matrix we convert these coordinates to the camera image plane and then draw a cube based on the 8 points that we have. The cube projected on the AR tag can be seen in Figure 10. The entire video can be found here.

Fig. 10: Cube on AR tag

III. DexiNed

Dense Extreme Inception Network for Edge Detection (DexiNed)is designed to allow an end-to-end training without the need to weight initialization from pre-trained object detection models, like in most of DL based edge detectors. DexiNed can be interpreted as a collection of two sub-networks, the dense extreme inception network (Dexi) and the upsampling network (USNet). Dexi receives an RGB image as input processing it in different blocks, whose feature-maps are fed to USNet.

A. Model Architecture

1) Dexi: The Dexi architecture contains six blocks acting similar to an encoder. Each block is a collection of smaller sub-blocks with a group of convolutional layers.Each sub-block constitutes two convolutional layers. All kernels are of size 3 × 3. In the very first block, convolutions are with a stride of 2. Each convolutional layer is followed by batch normalization and a rectified linear unit (ReLU). The feature-maps generated at each of the blocks are fed to a separate USNet to create intermediate edge-maps. These intermediate edge-maps are concatenated to form a stack of learned filters. At the very end of the network, these features are fused to generate a single edge-map (Fused edge map).

2) USNet: The upsampling network (USNet) is a conditional stack of two blocks. Each one of them consists of a sequence of one convolutional and deconvolutional layer that up-sample features on each pass. The block-2 gets activated only to scale the input feature-maps from the Dexi network. This block is iterated until the feature-map reaches a scale twice the size of the GT. Once this condition is met, the feature-map is fed to the block-1. The block-1 processes the input with a kernel of size $1 * 1$, followed by a ReLU activation function. Next, it performs a transpose convolution (deconvolution) with a kernel of size $s * s$, where is the input feature-map scale level. With the last deconvolution of block-1, the feature-map reaches the same size as the GT. The last convolutional layer does not have an activation function. It considers three strategies for upsampling blocks: bi-linear interpolation, sub-pixel convolution and transpose convolution. This is an influential factor in generating thin edges that enhances the visualization of edge-maps. The entire architecture is shown in Figure 11.

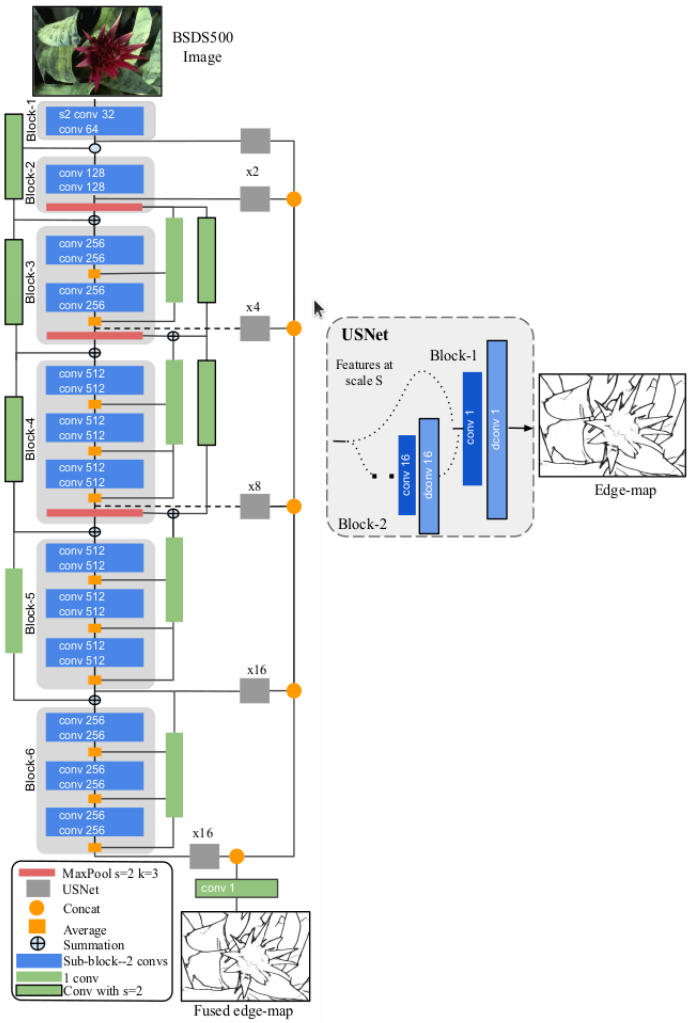


Fig. 11: DexiNed

B. Testing DexiNed

The pretrained tensorflow model is tested for the image in the Cityscape dataset. The original image, fused edge map and the final edge map is shown in Figure 12, 13 and 14 respectively.

Fig. 12: Original Image

Fig. 13: Fused Edge map

Fig. 14: Edge map

C. Comparison with Canny Edge Detector

The Canny edge detection algorithm is applied to the same input image and its comparison with DexiNed is shown in Figure 15. The entire code on Google colab can be found here.

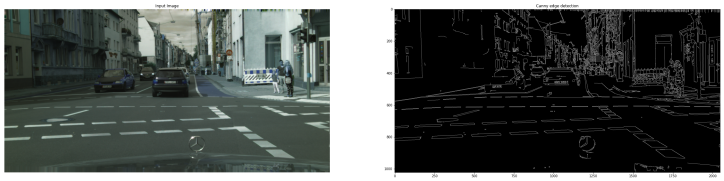


Fig. 15: Canny detection output

REFERENCES

[1] https://github.com/xavysp/DexiNed/tree/master/legacy
[2] https://stackoverflow.com/questions/44457064/displaying-stitched-images-together-without-cutoff-using-warpaffine/4445986944459869
[3] https://akshaysin.github.io/fourier$_{t}ransform.html.Yiai9xtOl8v$