ENPM 673 - Project 1

Sahruday Patti

University of Maryland, College Park

Summary 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 location and identity of AR Tag from a given image sequence while the tracking stage will involve keeping the tag in "view" which involved superimposing an image and placing a 3D cube over the tag throughout the sequence and performing image processing operations based on the tag's orientation and position (a.k.a. the pose).

Keywords · AR Tag · Image Processing · Detection · Tracking · HomographyPerspective Transform

✉ Sahruday: sahruday@umd.edu

AR Code Detection In an Image, an edge is the place where sharp contrast occurs. This sharp contrast forms the boundary of the objects in the image, thus paving a way to identify and locate the objects in the image. To perform Edge detection, we need to take the first derivative (Discrete Derivatives as we are dealing with pixels) along both the axis to detect the places of high contrast in an image along both the axis. This involves using Convolution over the entire image by various type of operators like Sobel,Prewitt. Canny Edge detection is also an edge detection algorithm which involves non maximal suppression and thinning. All these algorithms involve high computations as we need to convolve masks like sobel over an entire image. As this convolution as is all image processing operations are linear so we can reduce a certain number of computations by using the property of linearity. Another way to detect edges by reducing the computations is by use of Fourier Transforms. Any Periodic signal can be expressed in terms of Sines and Cosines which is the essence of Fourier Transforms. So, by assuming our whole image as repeating periodic sequence we can express our image in terms of a Sine-Cosine wave form. This property allows us to reduce the computations to perform the edge detection, As multiplication in frequency domain is equivalent to Convolution in Spatial Domain. So by use of the Scipy functions i converted the image to the Frequency Domain. Later, i shifted all the frequencies by placing the center at zero. Now,There are far less edges in the image as compared to objects thus all the edges form a high frequency components of a signal. But Noise in the image will also be part of this high frequency signals so in order to reduce the noise, i multiplied a Gaussian kernel to blur the image thus reducing the noise. An important property of Fourier Signal is that we can reconstruct the image if we know the amplitude,frequency and phase of the signal.So,Later converted the signal by using Inverse FFT to give a Blurry image. Now, Threshold-ed the image to give a Binary image and converted the image to a Fourier Signal. Now, the high frequency components of this image form the edges of the image as there are far too less edges than objects in the image and less noise. So by multiplication with a circular mask and by using the high pass signals we get the Fourier Signal of an Edge Image. How good an edge is determined by the radius of the circular mask but there will be trade off between the number of edges detected vs number of edges in the image. Later, converted this signal back to the spatial domain.

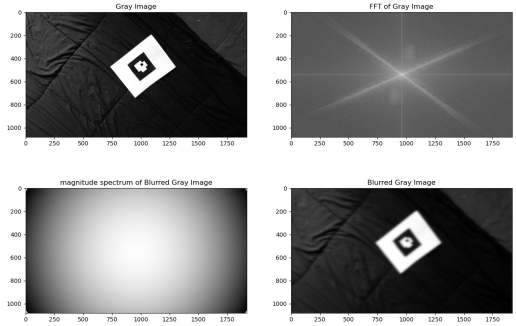

Figure 1: Image Blur using FFT

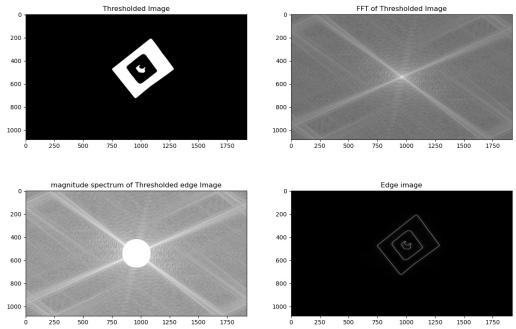

Figure 2: Image Edges using FFT

Decode custom AR tag In order to Decode the Custom AR tag, i resized the tag into a 160 x 160 grid. Now, as mentioned i have converted the 160 x 160 grid to an 8 x 8 grid so that each grid contains 20 pixels. Then i calculated mean of each grid and if the mean of the grid > 255/2, then i set the value of the grid to 255 otherwise it is set to 0. now, i extracted the inner 4 x 4 grid where the Rotational and ID value is encoded. Now, if in a video frame if the tag corners are found rightly we would have only one high value at the corners of the 4x4 matrix. Now, the 4 x 4 grid is rotated until the bottom right corner is high i.e 255 which represents the upright position of the AR tag. Then, the 2 x 2 grid is Extracted in which the LSB to MSB are ordered clockwise from the left corner of 2 x 2 Grid. The ID of the reference tag is found to be 15.

Corner Detection Corners are the points in the image where there is a change in both x and y directions. This points can be used to detect the objects where there is change in scale,rotations etc by various methods such as Harris corners and Shi-Thomasi version of GoodFeatures to Track. I have used both of these to detect corners and settled on Harris corners algorithm given the video.

Shi-Thomasi version of corner detectors considers the minimum eigen value of the auto correlation matrix where the direction of change is rapid and computes maxima in the direction of smaller eigen value. As it considers only the one eigen value its shape may change with respect to rotation. But, the features provided will be robust for in the case of tracking where we can use Lucas Kanade tracking algorithm which involves Optical flow which assumes all the pixels in the image undergo same motion in the image so that we can track the detected features. I tried to implement the algorithm but i couldn't get good results with the algorithm implemented, if i had some more time i might have got good results. The Shi-Thomasi corners involved 3 parameters the clarity of the corner, the euclidean distance between the corners and number of corners. Tried to tune these values and found the best to 10 corners of 0.1 clarity with minimum of 100 euclidean distance between the coordinates.

The Harris Corner Detector algorithm on the other hand considers both the eigen values of the auto correlation matrix which makes it insensitive to change in rotation. As our video has predominantly rotational motion i choose to continue with the Harris Corner Detectors. But, the initial few frames couldn't detect the corners properly and some of the frames through out the video does not detect the corners properly. The Harris corners involved the tuning parameter alpha, sobel operator kernel size and neighbourhood window size. Tried to tune these values in order to detect good corners. I performed the morphological operations to the gray image in order to remove noise and detect the rectangles properly but i found to miss some initial frames and where there is a change in rotation speed predominantly. found the Harris corners are also robust to illumination changes in the image. All in all, with more fine tuning the result can be improved over time.

Decoding AR tag from the Video In the Video, for every frame the corners are found using the Harris Corner Detector and once the corners are found, we use the rectangle properties and we detect the corners of the AR tag by popping out the Xmin,Xmax,Ymin,Ymax of the white sheet. Once we found the corners, sort the corners in anti clockwise starting from top left of rectangle.Then we compute the Homography between the image corner coordinates and the desired 160 x 160 coordinates. Once we Compute the coordinates, we have two ways to retrieve the information for the upright image to be shown. The Homography computed is a 3 x 3 matrix so we are doing a Perspective Transform which has 8 degrees of freedom. so a quadrilateral can be transformed to a different plane allowing it to distort the quadrilateral. After computing the coordinates, we have two ways to get the intensity values of our upright image. One method is to find for every pixel intensity in the image plane coordinates where does these pixel values maps to the desired image. This is called Forward warping. This causes holes in the desired upright image because the pixels in image plane may be mapped to in between pixels in the desired coordinates. there is another method, which is called inverse warping. Here, we say, for every pixel coordinate in the desired image figure out where it came from in the image plane. In order to do this we multiply with the inverse of the Homography matrix. There are methods like Bilinear interpolation and cubic interpolation which can plug in the holes. Bilinear Interpolation is used to superimpose Testudo.A simple inverse warping is used for AR tag Detection.This method produced good results as most of the holes are plugged and we get a complete binary image while decoding the tag in the video when we detect the corners accurately. Later, we flip the AR tag by 90 degrees until the right most corner is found to be 255. If the corners are not detected properly after four rotations the rotation is stopped and we get an incorrect value and orientation of the AR tag. If we found the Upright image, the AR tag is Decoded in the similar manner where we use 4 x 4 grid to detect orientation and 2 x 2 matrix to decode the value. The ID value of AR tag in the video is found to be 13.

Figure 3: AR tag decoded in one of the frames

Superimposing Testudo onto the AR tag

Superimposing the Testudo also follows the same pipe line as decoding the Ar tag from video. We detect the corners of the Ar tag, then we compute the orientation of the tag and we then sort our corners according to the orientation and then perform Homography between the corners of Testudo image and the AR tag. After performing the homography we get the image plane coordinates for the corresponding testudo image to be superimposed. Now i performed an inverse warping from the image plane to the testudo and did an bi-linear interpolation between pixels and calculated the intensity value at the corresponding image plane coordinates.

In some frames the corners detected where out of bounds of the image plane, some frames the square is distorted to many different shapes, due to the lack of proper corner detection. I have tried to adjust this to the previous orientation and corner values without good results.

Figure 4: Testudo superimposed on AR tag with orientation

Superimposing a virtual Cube onto tag

To project a 3D object onto the camera plane, we need 3d coordinates. So in order to obtain the coordinates we need to compute Projection Matrix P. where $P = K[R|t]$. where K is the intrinsic camera matrix. Consider the rows of the homography matrix as h1,h2,h3. Now, $\lambda H = K\tilde{B}$ where $B = [r_1, r_2, t]$.

Now the scale λ is given by $\lambda = \left(\frac{\|K^{-1}h_1\|+\|K^{-1}h_1\|}{2}\right)^{-1}$

Next step is to compute, $\tilde{B} = \lambda K^{-1}H$.

later, $B = \tilde{B}(-1)^{|\tilde{B}|<0}$. if the det is less than 0 we multiply with -1. this is because the equation gives two solutions one for behind the camera and the other is in front of the camera. we need the solution in front of the camera.Here,$r_1 = b_1, r_2 = b_2, r_3 = r_1 x r_2, t = b_3$ The above is the code written for the projection matrix as it is.

Now, to draw the cube define the coordinates of cube in reference frame and then multiply with the projection matrix to get the coordinates of the cube in the image plane. Now, we draw lines to connect these coordinates to get the desired result of cube on the AR tag.

The output Video can be found in the Below link : `https://drive.google.com/file/d/1CPy15xIwnzFrXd7T_zlj4bBuVcweZHAB/view?usp=sharing`

Figure 5: Testudo superimposed on AR tag with orientation

ExtraCredit The DexiNED is a Deep Learning Based edge detector. It is an inspiration from the Convolutional Neural Networks. As the name indicates the idea of CNN is to spatially convolve the kernel on a given input image. Just like in edge detection where we convolve the kernel over the entire image and place the value in the center pixel, the Kernels in CNN is also just a matrix of values, called weights, which are trained to detect specific features. In conventional edge detection, we use a custom kernel to detect different types of edges. In CNN'S we want the network architecture to be able to learn those kernels. In order to make the network learn, we input the value of the kernel multiplication with the image to a non linear activation function like ReLU,Sigmoid.

To speed up the training process and reduce the amount of memory consumed by the network, the convolutional layer is often followed by a pooling layer to remove redundancy present in the input feature. for eg: max pooling moves a window over the input and simply outputs the maximum value in that

window effectively reducing to the important pixels in an image.

The above same essential properties are found in the DexiNed Architecture. In Essence,The DexiNed contains Six main blocks connected by the 1x1 Convolution Blocks. Each of these Six blocks contain single/multiple convolutional layers of 3x3 blocks. The output of each of these blocks are given to an Up sampling function and each of the sub blocks gives out an intermediate edge map. Later, these intermediate sub blocks are fused together in Up sampling function to get an output edge image. The output of the DexiNed is characterized by its Loss function as is the case for any architecture or ML model as this is the function we are trying to minimize. The DexiNed has a Cross Entropy Loss function which states that its output is in terms of probability. If most of the times the probability that the edge is detected as an edge then we would have achieved the optimal Minima of the loss function.

The architectural details of what is a Convolution network and an up sampling network all the kernel details are mentioned in the paper. In my understanding, what each of these blocks are doing is computing intermediate edges at various levels and figuring out different Kernels as in some blocks only one Convolution is done and the output is fed to other block via a 1x1 convolution where in that block they use different kernel size,stride and input image shape and figure a different kernel (weights) later all these are summed up the Upsampling function to produce a Final edge image. what essentially each of these blocks are doing is computing the intermediate edge images. For example if we convolve a Sobel filter with the image we get the edges of that image. Similarly there are various types of kernels to detect various type of edges or patterns in an image. In a convolution network also we try to detect various types of edges so as to achieve the final goal which in the case of DexiNed is to detect very THIN edges of an image. we achieve this by finding the global minima in the Cross entropy loss function.The Canny edge Detector also involves detecting the edges and doing non maximal suppression so that we find very thin edges.

Figure 6: Given Input image to compare

Figure 7: Edge image of a DexiNed

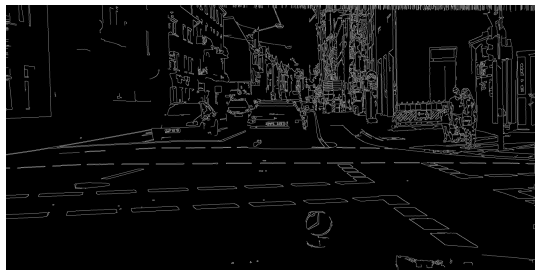

Figure 8: Edge image of Canny

The Difference between the DexiNed and the Canny edge images is the ability of the DexiNed architecture to only have essential edges. In other words, the DexiNed has the ability to make texture inside an edge to suppress. As we can see in the image, the buildings have edges which have an edge inside of these edges which are visible in Canny but not in the DexiNed image.

In Canny, we can make these little edges disappear by increasing the threshold value (canny has two thresholds min and max) but the clarity of the edges that are important are also effected.