Project Report

-Rishabh Mukund

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). You are provided multiple video sequences on which you test your code.

Detecting the AR Tag:

The AR tag is printed on a white paper and is blacked on a background, to detect the AR tag we have to ignore the paper and the background. This can be done through morphological operations such as opening and closing.

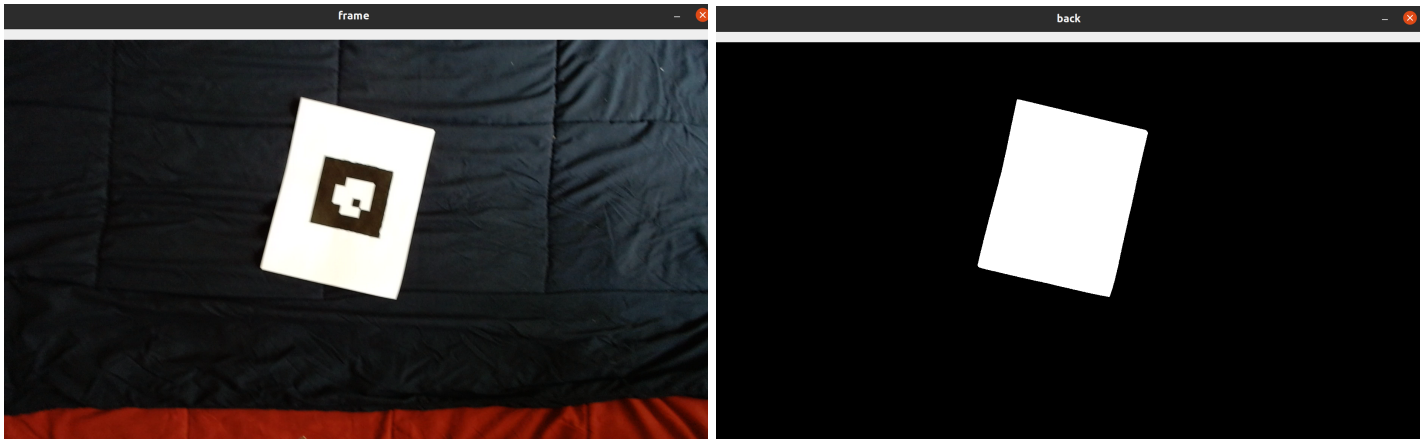

Fig: Left side is the original frame, Right side is the extraction of paper after morphological operations

After we eliminate the background, we perform FFT on the thresholded image. In the FFT image, the high frequency components are away from the center and the low frequency components are closer to the center. As edges are high frequency, to detect edges we mask the center of the image and calculate the inverse FFT of the image.

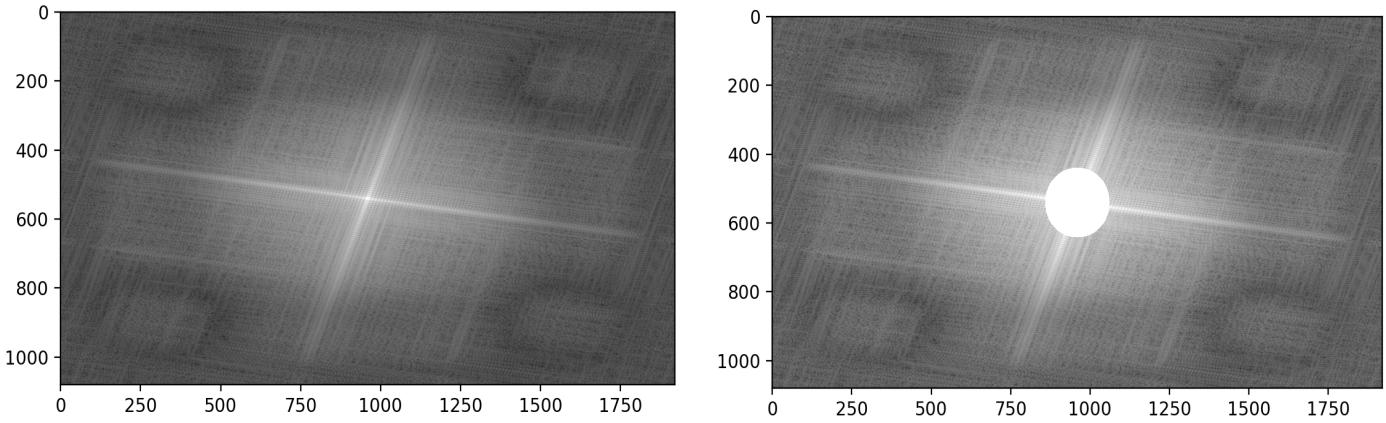

Fig: Left side is the fourier transform of the frame, Right side fourier transform with mask

After we get the edges, we remove the noise in the edges by performing an opening morphology operation and then gaussian blurring. To detect the corners, we use the Shi-Tomasi algorithm.

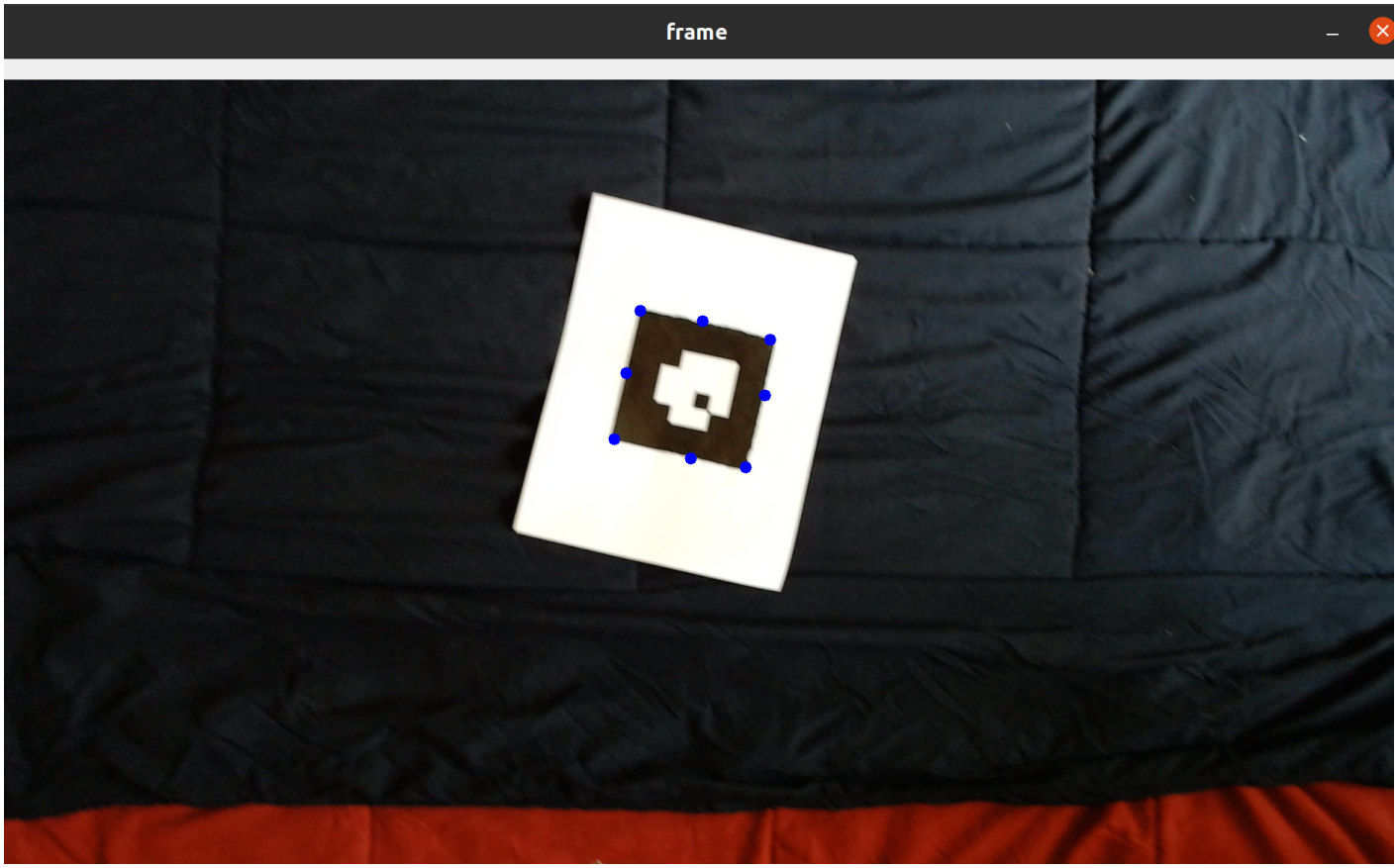

Fig: Detected edges after Shi-Tomasi algorithm

After the shi-tomasi corner detection, we will receive more than 4 corners based on the noise of the image. To get the corners of the tag, we have to perform some geometrical operations.

As we know that for a square, all sides are equal and the diagonal is (2)^0.5 * side.

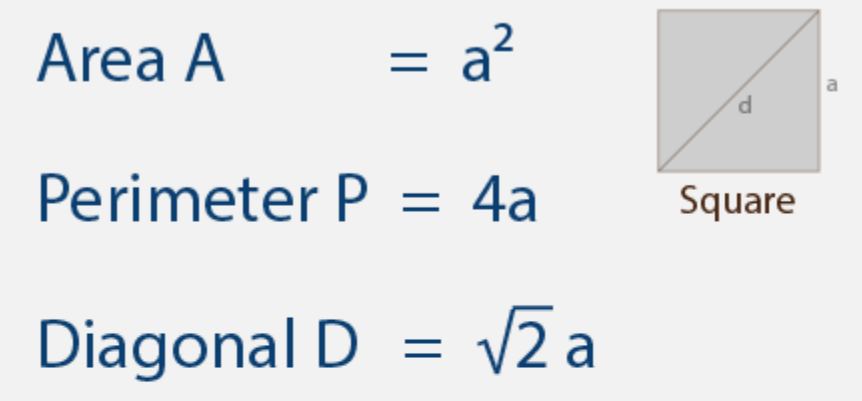

Fig: Equations of a square

Using the square equations, we iterate through all the detected corners to find the closest square. After we perform that we get the exact desired points.
We have to make sure the corner points are cyclic.

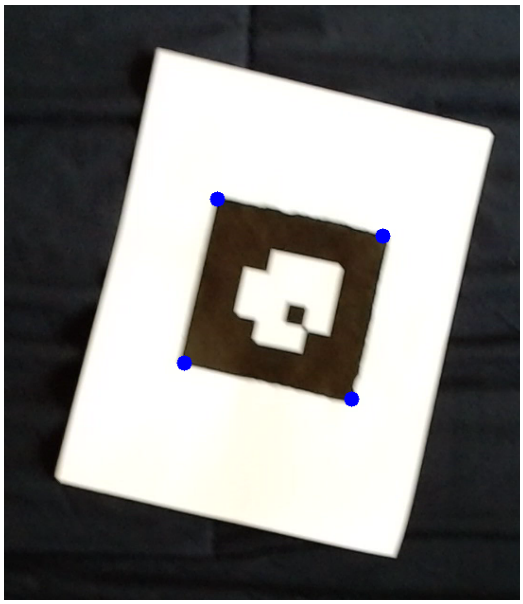

Fig: Exact corners for TAG

Homography and Warping:

After finding the four corners of the tag and their co-ordinates we find the homography between the corners of the template and the corners of the tags. The co-ordinates in the world frame are chosen to be as [(0, 0), (80, 0), (80, 80), (0, 80)]. To find Homography we first define the A matrix :

$$A = \begin{bmatrix} x0 & y0 & 1 & 0 & 0 & 0 & -xc0 * x0 & -xc0 * y0 & -xc0 \\ 0 & 0 & 0 & x0 & y0 & 1 & -yc0 * x0 & -yc0 * y0 & -yc0 \\ x1 & y1 & 1 & 0 & 0 & 0 & -xc1 * x1 & -xc1 * y1 & -xc1 \\ 0 & 0 & 0 & x1 & y1 & 1 & -yc1 * x1 & -yc1 * y1 & -yc1 \\ x2 & y2 & 1 & 0 & 0 & 0 & -xc2 * x2 & -xc2 * y2 & -xc2 \\ 0 & 0 & 0 & x2 & y2 & 1 & -yc2 * x2 & -yc2 * y2 & -yc2 \\ x3 & y3 & 1 & 0 & 0 & 0 & -xc3 * x3 & -xc3 * y3 & -xc3 \\ 0 & 0 & 0 & x3 & y3 & 1 & -yc3 * x3 & -yc3 * y3 & -yc3 \end{bmatrix}$$

Now, using Singular Value Decomposition we find U, S and V Matrices. SVD is expressed as:

$$A = U * S * VT$$

U is $m \times m$ orthogonal matrix, where each column forms the EigenVectors of the matrix *(A*AT)*.
VT is an $n \times n$ orthogonal matrix, where each column forms the transpose of Eigen vectors of the matrix *(AT*A)*.

Now to find the homography matrix we take the last column of the V matrix and then reshape it to form a 3x3 matrix. Once the homography matrix is found we then warp the contours from the image plane to the world frame. This can be seen in the figure below. - 3

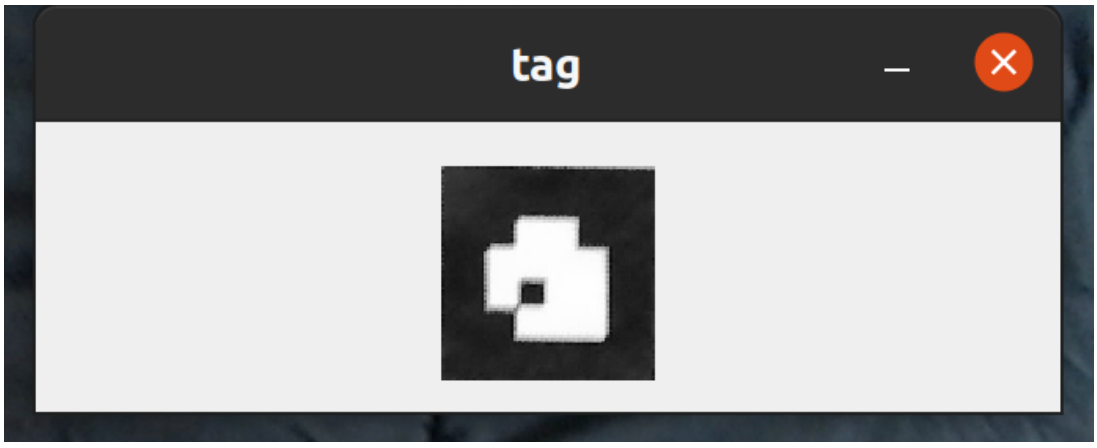

Fig: Warped image of TAG

Decoding the Tag:

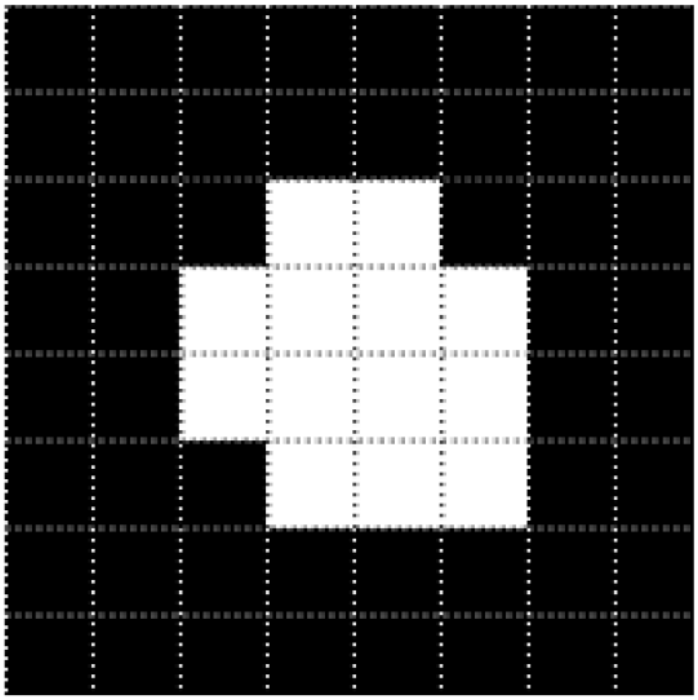

Fig: AR Tag

To properly use the tag, it is necessary to understand how the data is encoded in the tag. Consider the marker shown in Fig.
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.

The inner 4 x 4 grid (i.e. after removing the padding) has the orientation depicted by a white square in the lower-right corner. This represents the upright position of the tag.

Lastly, the innermost 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.

Replacing Tag with Template:

Fig: Template

Now, using the detected tags found above we superimpose Testudo's logo onto the AR tag with the correct orientation. We once again find the homography between the corner points of the template and the corners of the tags produced. Template is then projected on the world frame with the order of its points reversed. This then gives the image of testudo in the world frame.(seen in figure)

Fig: Template superimposed on the AR Tag

Augmented Reality:

In this part of phase 2, we have to project a 3D cube on the tracked AR Tag from the video. We use the homography matrix derived from the previous part and the given camera calibration matrix to calculate the rotational and translational components of the projection matrix. The transposed calibration matrix was defined as:

$$k = \begin{bmatrix} 1346.1 & 0 & 932.16 \\ 0 & 1355.93 & 654.9 \\ 0 & 0 & 1 \end{bmatrix}$$

The values of these rotational and translational components along with a 3D axis are then used to detect the coordinates of the tag in world frame by using the Projection Matrix. The cube is now to be placed at the generated coordinates. Since we are projecting a 3D shape on a 2D image, we assumed the z axis to be negative in the upwards direction and calculated the other four corner coordinates of the cube using a projection matrix. The cube is finally made by drawing simple lines using the functions line().

Points for the cube
[(0, 0, 0, 1), (0, 80, 0, 1), (80,80,0, 1), (80, 0, 0, 1), (0, 0, -80, 1), (0, 80, -80, 1), (80,80, -80, 1), (80, 0, -80, 1)]

Problems encountered

The main problem with the code was faced with the corner detection part. Due to blurry images, detection of the tags could not be done continuously. As the frames in the video came to better focus, the tags were detected easily. Problems were also faced while superimposing and projecting the cube over the detected tag. The cubes were not being positioned properly on the tag.

Links

Video:
https://drive.google.com/drive/folders/1tmlK2vKYNR2--Q1U5XqxsrlBvOemE5kJ?usp=sharing

Extra Credit Question (DexiNed)

DexiNed (Dense Extreme Inception Network for Edge Detection) with convolutional layers can eliminate most of the false edges and also improve the accuracy of edge detection. DexiNed can be trained from scratch without any pre-trained weights. It also generalizes well to other datasets without any fine-tuning.

DexiNed can be understood as a combination of two sub-networks i.e Dense extreme inception network (Dexi) and upsampling network (USNet). An RGB image is given as an input to Dexi, it processes it in different blocks and feature maps are generated which are further fed to the USNet. Using the feature map Usnet generates intermediate edge-maps. These maps are further concatenated to form a stack of learned filters. At the end of every network, a single edge-map is generated.

Dexi

The architecture of Dexi is constitutes of 6 blocks. The architecture is similar to a encoder's architecture. The blocks are made up of a group of sub-blocks with a group of convolution layers. The sub-blocks and the blocks are coupled using skip-connections.

Each Sub-block contains two convolutional layers. The size of the kernels are of size 3x3. Each layer is followed bu a batch normalization and a rectified linear unit (ReLU). It also houses max-polling operators with a 3x3 kernel size and stride of 2.

Due to many convolutional layers over the process, important edge features are not visualized. To solve this issue parallel skip-connections are implemented. From the third block, the output is averaged with another skip-connection names as second skip-connections. After the max-pooling operation, these second skip connections average the output of connected subblocks prior to summation with the first skip-connection—FSC. Parallely, the output of max-pooling layers is directly fed to subblocks.

USNet

The upsampling network (USNet) is a conditional stack of two blocks. The blocks consists of a sequence of a convolutional and a deconvolutional layer that up-sample features while processing. The block-2 is triggered to scale the input feature-maps from the Dexi network. The process is repeated in the block 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 kernel of size 1 × 1, followed by a ReLU. it performs a deconvolution. The feature map reaches the same size as the GT.

Result Images

Two images were generated using DexiNed. Ie, an average image and a fused image.

Average Image *Fused Image*

The same image was run through canny edge detector and the result is the image below by adjusting threshold values.

The canny detector has less holistic and semantic understanding of the content in images which tends to limit the accuracy in the predictions made. With the use of deep learning and convolutional neural networks DexiNed can holistically and reliably detect edges better in the images.