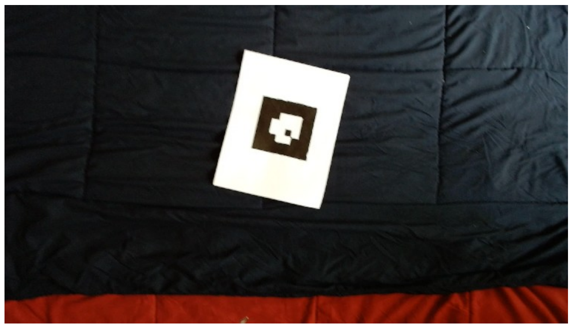
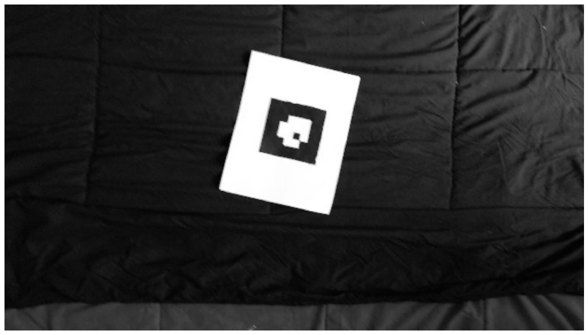
Project 1

ENPM 673

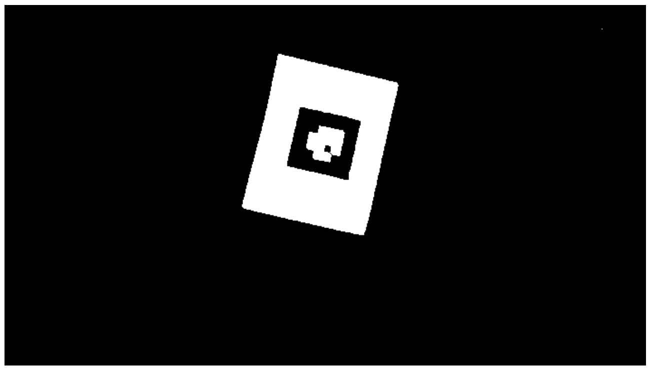
Nitesh Jha (0101)

1. The task of April tag detection consists of the following steps:
(I) Reading the image and converting it to grayscale. This is done so as to use the single channel image for processing such as thresholding and applying fast fourier transform.

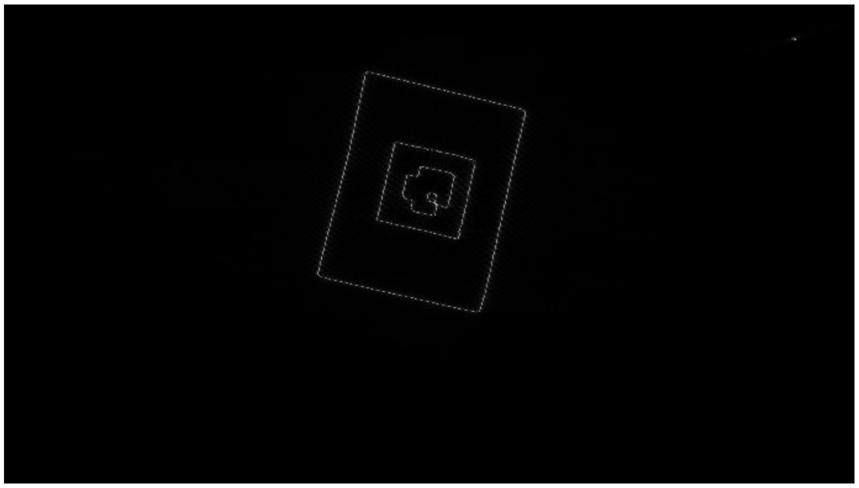




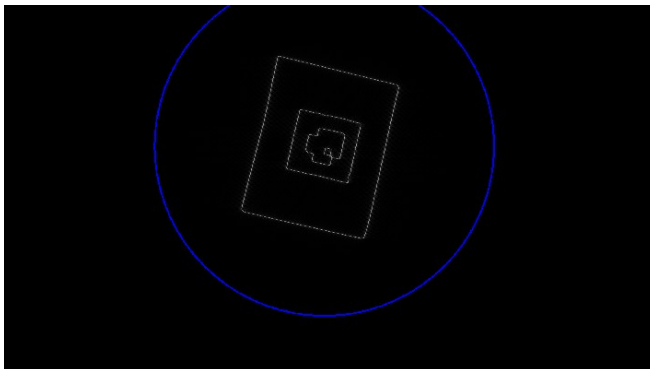
(ii) Threshold the image: The frames of the video contain light reflections on the mat, red areas, noisy pixels which would hinder the detection of the tag using methods based on pixel locations that are white. Thus, applying thresholding is useful with an appropriate threshold, which was determined by trial and error over the video, and was set to 150 (binary thresholding). This resulted in very few noisy pixels in frame.

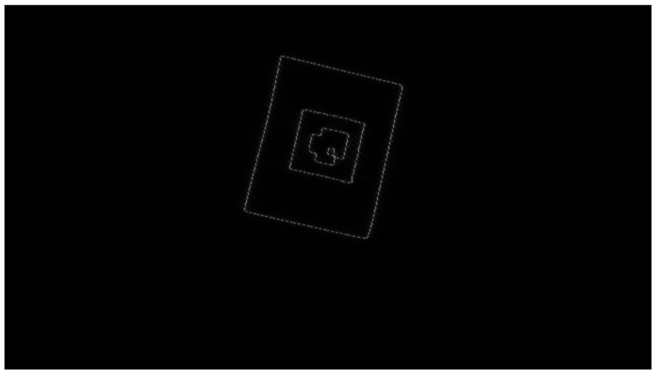


(iii) Fast Fourier Transform: Next, the edges of the frame were extracted using the fast fourier transform. The frame was transformed with the discrete fourier transform. This results in an image in the frequency domain. Then, a shift of the transform was performed which centered the transform. As we require only the prominent, strong edges, a simple high pass filter can be used. This is done by creating a circular mask and placing the mask at the center of the FFT image. As the frequency increases outwards from the center, the radius of the mask would determine the extent of 'thresholded' frequencies. This again was decided by trial and error as 500. The resulting image was inverse shifted back, and then inverse FFT was performed to obtain an image which consists of prominent edges. This is shown in the image below. As the result of this was representative of the required edges of the sheet and tag, this was used throughout the length of the video to detect the edges.

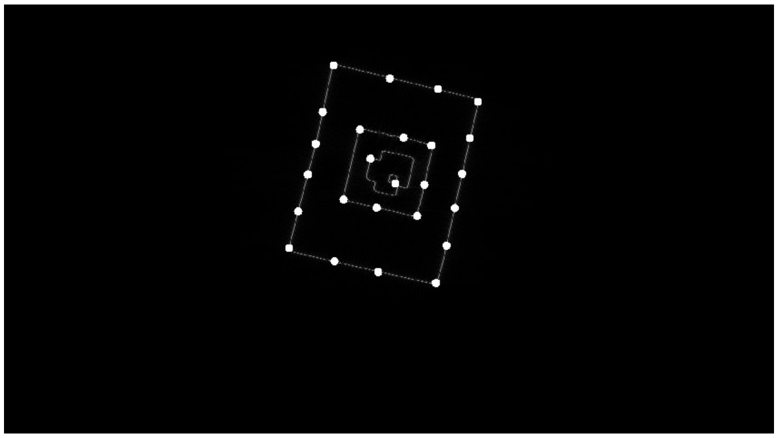


(iv) Masking:The resulting image consists of noise around the sheet which required to be eliminated. Masking some regions of the image would be useful for this. This could be done with a fixed mask which covers only certain areas at all times. However, to make the pipeline robust in all positions of the sheet, a movable mask was created. The idea behind this was to have a mask move along with the sheet and eliminate all noise which are beyond a certain distance away from the sheet. The shape of this mask was chosen as circular as the symmetry would help eliminate orientation issues of the mask, as a misoriented mask would mask the corners resulting in poor results. The center of the mask was calculated per frame, as the mean of the white pixels in the frame. Although this resulted in considerable instability on the thresholded image, using FFT with this movable mask gave the mask a gradual motion. The radius of the mask could be calculated per frame as well, by using the average deviation of the white pixel coordinates. However, setting the value of 450 resulted in the mask that would not cut the paper and mask corners, while masking as much area as possible. This condition was checked by running the mask over the entire video.
Using such a mask, all pixels outside this were set to zero, while those inside were retained.



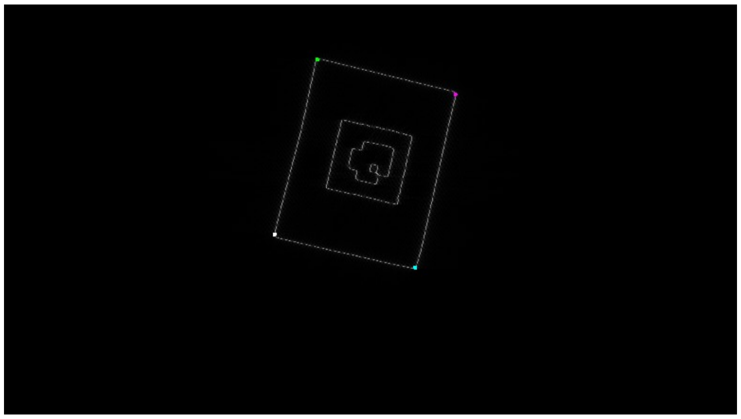


(v) Corner detection: The next step was to detect corners using Shi-Tomasi corner detection. For detecting the outer edges of the sheet, the number of points passed were set as 25 with a quality factor of 0.01, having a minimum distance of 70 pixels between any two corners. The number of corners were set to a high value as we would like as sufficient corners so that a reasonably accurate assumption of extreme corners can be taken. The distance between them was set as 70 as this resulted in well spaced corners which would capture the extreme corners accurately.



However, there were some cases this would fail. This would happen when the edges were not strong enough for the algorithm to detect corners at all the actual corners.

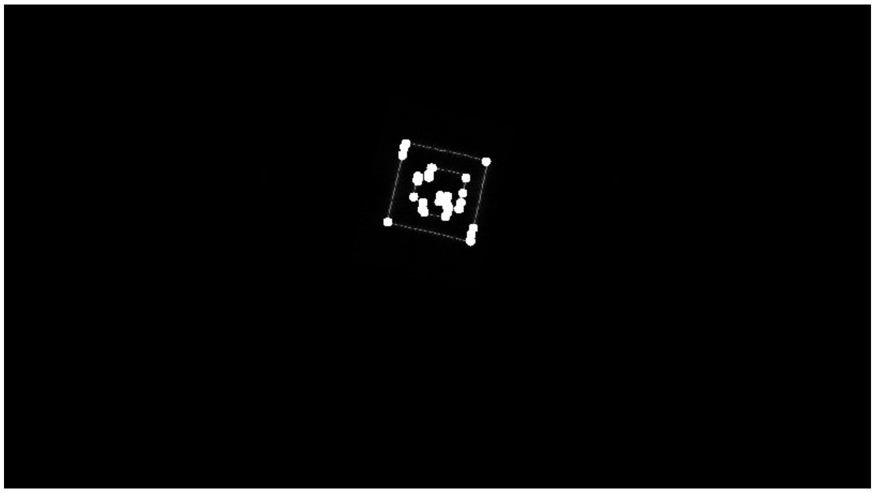
(vi) Extreme corners: After we have all possible corners in an image, we require corners which are at the extremes on X and Y directions. This is obtained by using finding the pair of corners with minimum and maximum X and Y values, resulting in 4 values: Xmin, Ymin, X,max, Ymax. The other coordinate(X or Y) of these values would be used to get the pairs which are the four pairs at the lowest and highest X and Y values.This was an accurate representation of the actual sheet corners.



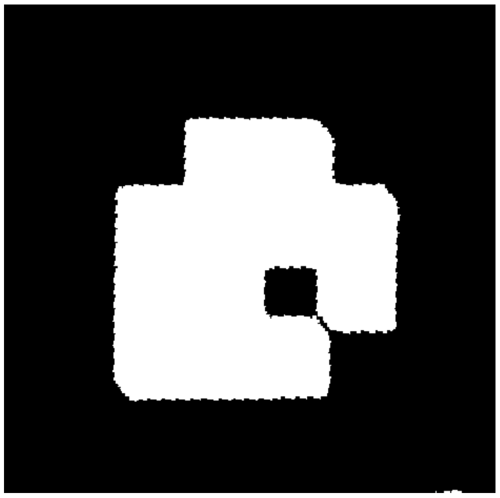
However, the major point of failure of the pipeline stems from this step. When the sheet is horizontal or vertical, finding minimum or maximum values give many locations which are very close, if not at the same value. As was observed, a bulge in the sheet or even slightly weaker edges resulted in the sheet extremes not being detected correctly.

A number of approaches can be adopted to counter this:
1. Moving average of corners points, which would make the corner shift gradual. This resulted in the output lagging consistently. However, this could be explored by adding a weight to elements in the moving average list, in decreasing order of recency.
2. Another way to counter this would be a masking that would leverage the temporal nature of frames. As there is no sudden shift in the edges, we do not require to search for the entire image for corners after the first corner as it would have shifted only by a limited distance. Thus, a search could be performed only in this limited distance for the corner in the next frame.
3. By leveraging the fact that this situtation occurs only when the edges are horizontal. Thus, we can determine a confidence bool (with a threshold) based on the last corner position, and if the consequent corners are far apart indicating horizontal/vertical orientation, the image can be rotated about the center by an angle (45 deg), and the corner points can be found on this rotated image. Then the points can be rotated back (-45 deg) using trignometry, and this resulting corner points can be used.

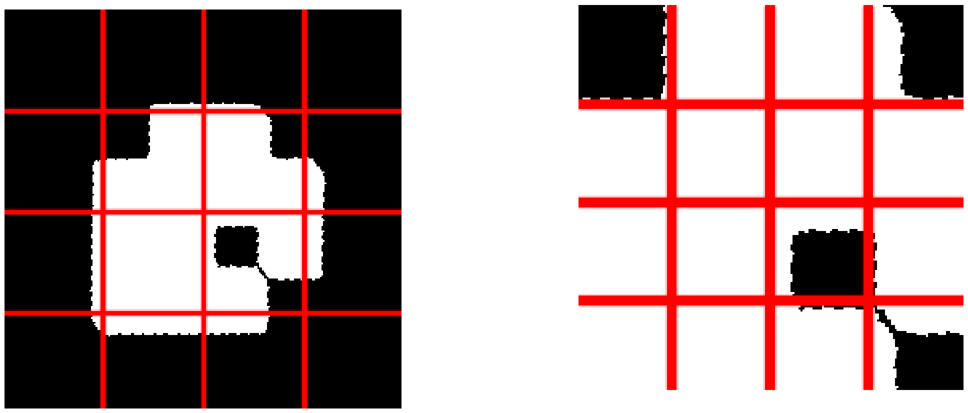
(vii) Inner corner extraction: Once we have the outer corners, this pixels can be masked using cv2.polylines() with a thickness. This is done in both directions of the outer corners to eliminate the entire edge (as cv2.polylines only pads on the inner side of the polygon). This gives only the tag region of the sheet. Then, the same procedure of applying Shi-Tomasi algorithm, followed by extreme corners extraction, can be applied to get the tag corners.



(viii) Homography: Using the tag corners, we can find estimate the homography of these corners and 4 corners chosen which form a square (of side 400 here). The resulting homography can then be used to warp the tag to make it normal to the image plane, for the ease of tag id extraction.

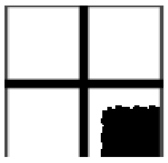


(ix) Decoding AR tag: As the warping puts the tag in a square shape, we can divide the image into equal parts to remove the padding. Although a minimum-maximum masking approach would be more robust to warping errors, this worked on most frames and was thus retained.

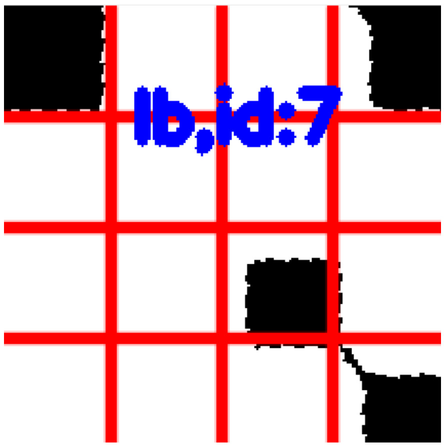


We can count the number of white pixels in the corner regions, and determine orientation based on the maximum number of these white pixels. This resulted in accurate orientation outputs.

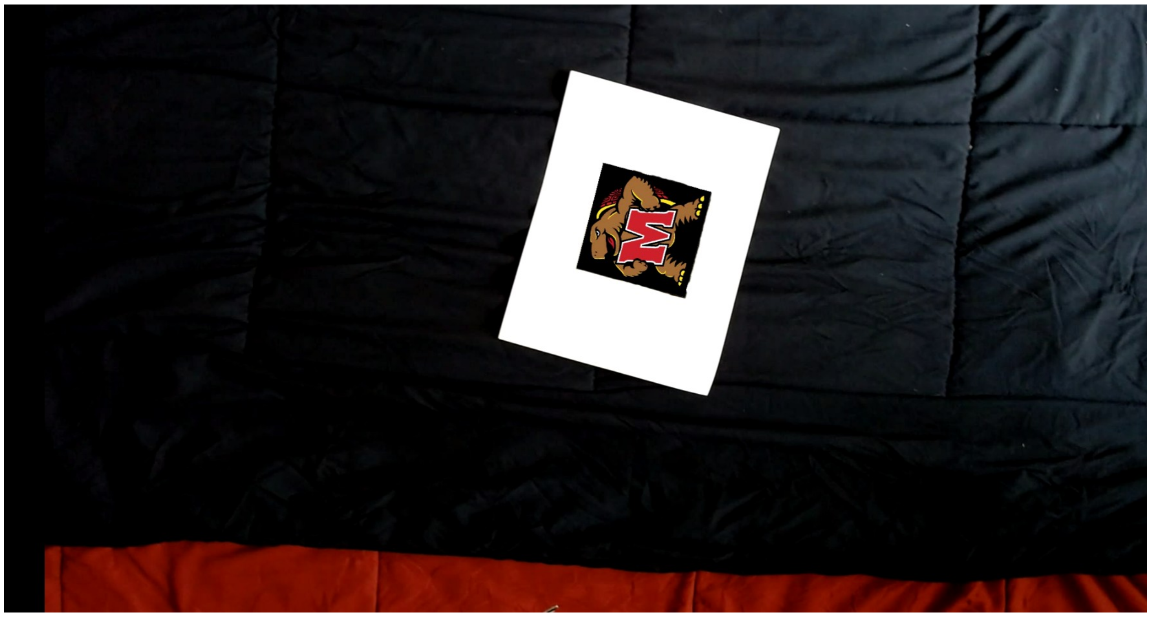
Then, the binary encoding image can further be obtained by dividing the image into equal parts.



We then find the number of white pixels in all 4 sections, and assign it 1 or 0 based on the number of white pixels. A tolerance of 1800 out of 2500 pixels are taken to accommodate for the warping errors. Thus, any square which has <=1800 pixels are set to 1 and 0 otherwise. Then, the binary value starting from the orientation's side is calculated.

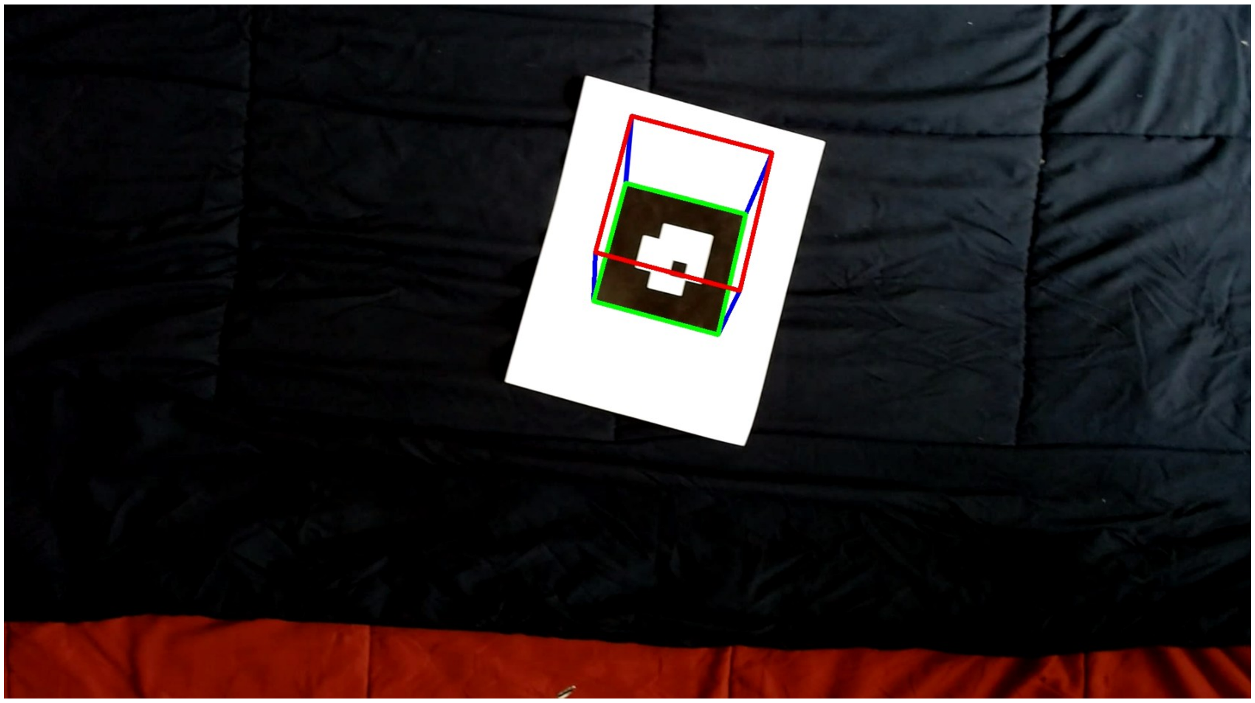


2. a) Superposing testudo on the tag:
For this, the pipeline will remain same till step (ix) of Q1, for getting the orientation of the image to be superposed. Only the binary value is not decoded in this case. With the orientation, image warping can be performed by iterating over the entire testudo image coordinates, and the inverse warping is performed using the homography matrix. If these coordinates are within the range of the original image frame, the pixel value of the testudo image is copied on the frame. This results in the superposition of the image on the tag.



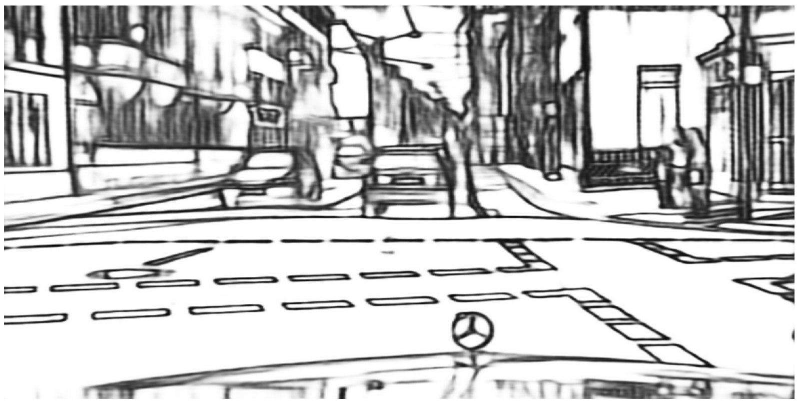
Video link: https://drive.google.com/drive/folders/1nRD2JbMdaFE4-x3DJJigdOfvTnvnhmKF?usp=sharing

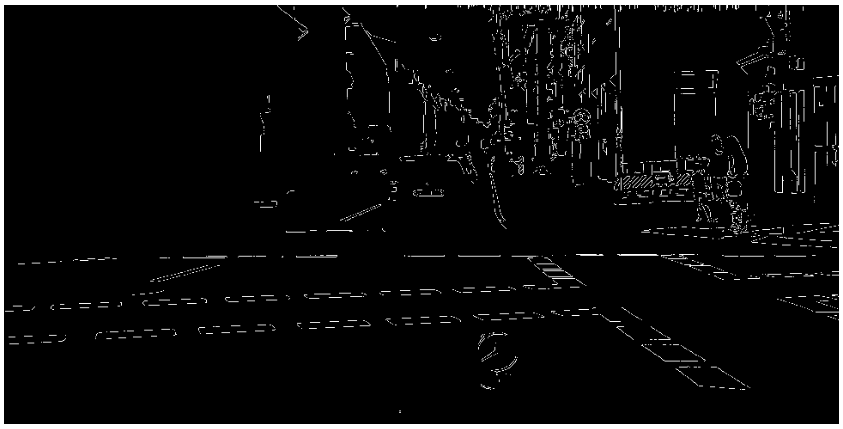
b) For the AR cube, we use the pipeline to obtain tag corners, same as Q1 and Q2(a). Then, cube corners are chosen from (0,0) to (400,400), and the homography between these are calculated, H.
Then, as we require to go from image coordinates to world coordinates to represent a 3-D structure, we require a projection matrix. For this, we use the given K matrix and the H matrix calculated to determine projection matrix P. This is done by K_inv * H, finding r1,r2,r3,t representing the extrinsic parameters. Thus P = [K* Rt] is calculated and used for projection. Then, cube points are defined in 3-D camera coordinates with a size of 400 (same as size used for perspective transform) and these points are multiplied with P to get world coordinates of these points.
Lines are then drawn on this image which connect all adjacent edges, which then represents a cubical structure on the tag.



Video link: https://drive.google.com/drive/folders/1nRD2JbMdaFE4-x3DJJigdOfvTnvnhmKF?usp=sharing

3. DexiNed:
The Dense Extreme Inception Network is a CNN which is used to detect edges, like the low level classifiers such as Sobel. These networks are trained on the BIPED dataset and predict an edge-map of the input image and have shown to predict even thin edges on new,unseen data. The network can be broken down into two components: Dexi and the Up sampling block, both taking different inputs. Dexi takes an RGB input whereas Up sampling block takes feature maps. The Dexi and upsampling blocks are chained constituting one block, and each block has two layers, convolutional and deconvolutional
The network uses standard convolutions, and is composed of 6 blocks which has different number of filters, layers, and pooling within the different blocks.
Within the network, intermediate edge maps are created which are then fed to the further subblocks which are then fed to the next block. It uses the weighted cross entropy loss function for the supervised learning.





Here, we see that the canny detector was able to detect outer edges better but the DexiNed detected edges within the object as well within the contour.