# ENPM673 - PROJECT 1

Vishaal Kanna Sivakumar
A. James Clark School of Engineering
University of Maryland
College Park, Maryland 20740
Email: vishaal@umd.edu

***Abstract*—This following report provides a brief overview of Project 1 implementation. This project aims at detecting and identifying AR Tags from a given video sequence. Once identified we calculate homography matrices and overlay an image and project a virtual cube over the tag.**

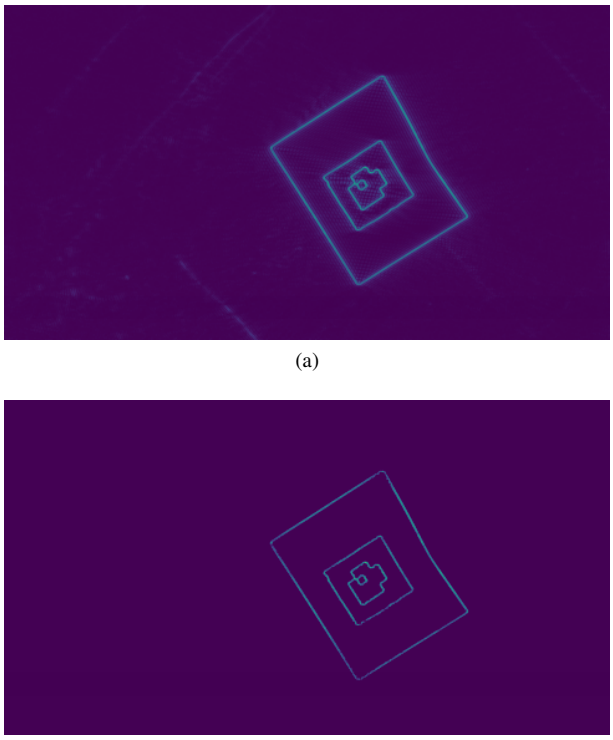
## I. Problem 1

In the first problem, the AR-tag is identified from a given video sequence and the id is decoded from it. The sequence of steps followed to achieve the desired output is detailed in the following subsections.

### *A. Edge Detection*

In order to detect the AR Tag, we start with filtering out the pixels which are not necessary and get edges from each frame. This is done using Fourier Transform (FT). The image is converted to gray-scale and FT is applied over it using inbuilt fast Fourier transform functions in OpenCV. We shift the the output to the center and use a high pass filter by applying a circular mask to it. Inverting this gives an image with only edges as shown in Fig 1. Further applying an appropriate threshold value, we obtain the right image.

The final output contains only ones and zeros after thresholding, which makes it convenient for further processing.



(a)

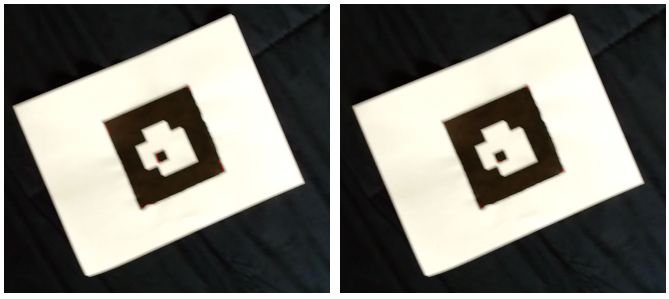
(b)

Fig. 1: (a) Texton (b) FT output before thresholding (c) FT output before thresholding

### *B. Corner Detection*

Applying standard corner detectors such as Canny or Sobel resulted in blocks of points at multiple locations which was not ideal for finding the homography. Thus, we exploit the prior knowledge we have of the video to detect suitable corners. We know that the AR-Tag is within a sheet of paper. Hence, from the edge points detected, if we move in any four perpendicular directions (up, down, left and right in the implementation) and count the directions in which we encounter a dark (black) pixel, we would obtain potential candidates for the corners. We use a small perturbation to all the edges in four directions and keep only those points which encounter three white pixels (or one black pixel). In the implementation, this is done twice with two different perturbations to reduce the candidates for corners.

Four points from the above candidates have to be chosen such that the corners form the bounding box of the Tag in order to compute homographies. We apply an iterative algorithm - four points are chosen at random and the area of triangles formed by points 1,2,3 and points 2,3,4 are computed. The largest sum of area is kept track of and the corresponding



(a) (b)

Fig. 2: (a) Corner candidates by using perturbation of 30px (b) Corner candidates by further processing using a perturbation of 20px

points are finalized as the corners of the AR-Tag. Using area of two triangles and not directly the quadrilateral makes it easier to arrange the points so that it can be sorted anticlockwise in the next step. The corners are depicted by four red points on the image in Figure 3

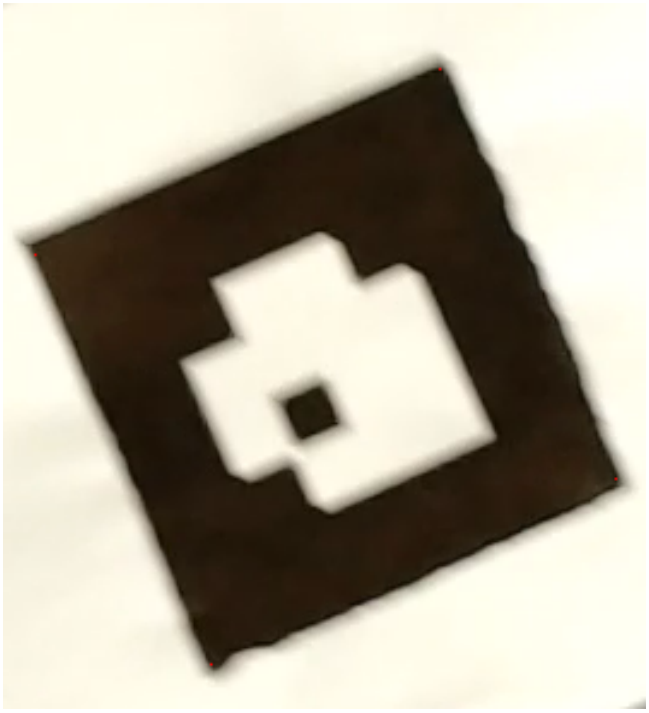


Fig. 3: Final Corners represented by 4 red pixels

The results could be improved by increasing the number of iterations. the algorithm works in images which are blurred as well and is a good alternative to using cv2.detectcounters for tag detection.

### C. Decoding the Tag

To decode the tag we first warp the image frame from the video to the reference image frame, that is, a horizontal plane. We sort the corners of the AR-tag in anticlockwise direction as $(x1, y1), (x2, y2), (x3, y3), (x4, y4)$. The corresponding coordinates of the reference image are taken as $(xp1, yp1), (xp2, yp2), (xp3, yp3), (xp4, yp4)$ and the following equation is solved using singular value decomposition:

$$AH = 0 \tag{1}$$

where

$$A = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_{p1} & y_1x_{p1} & x_{p1} \\ 0 & 0 & 0 & -x1 & -y1 & -1 & x_1y_{p1} & y_1y_{p1} & y_{p1} \\ . & . & . & . & . & . & . & . & \\ . & . & . & . & . & . & . & . & \\ . & . & . & . & . & . & . & . & \\ . & . & . & . & . & . & . & . & \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x_{p4} & y_4x_{p4} & x_{p4} \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y_{p4} & y_4y_{p4} & y_{p4} \end{bmatrix} \tag{2}$$

and

$$H = \begin{bmatrix} h_11 \\ h_12 \\ h_13 \\ h_21 \\ h_22 \\ h_23 \\ h_31 \\ h_32 \\ h_33 \end{bmatrix} \tag{3}$$

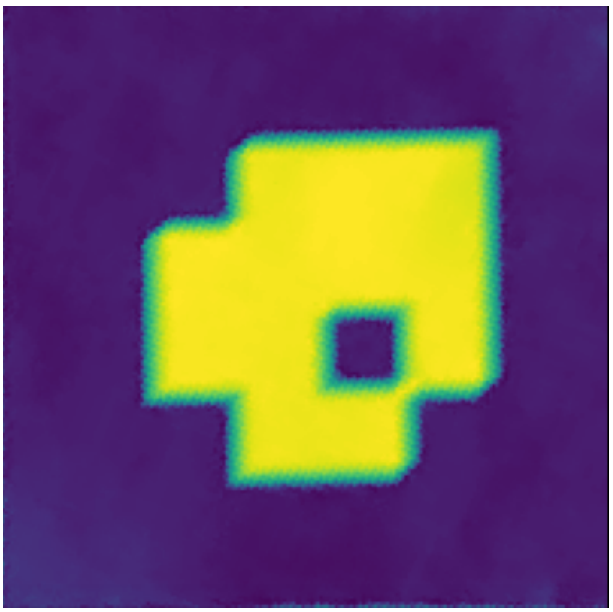


Fig. 4: Tag warped to the reference plane

After warping the image to the horizontal plane, we obtain an image as shown in Figure 4. This is used to obtain the orientation using the white square in the inner $4x4$ grid, which is at the bottom right corner in its upright position. The tag is decodes by computing the pixel intensity at the centers of an $8x8$ grid formed by dividing the warped image appropriately,

$$4x4 - Grid = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{4}$$

The tag is rotated to its upright position to get the $2x2$ pixel values, which is shown below. The encoding scheme is used later to obtain the ID.

$$Tag - ID = \begin{bmatrix} * & * & * & * \\ * & 1 & 1 & * \\ * & 0 & 1 & * \\ * & * & * & 1 \end{bmatrix} \tag{5}$$

## II. Problem 2

This problem deals with overlaying a given image and a virtual cube on the detected AR-Tag.

### A. Overlaying the Image

To overlay the image we need to orient the given image such that corresponding corners of the image and the tag match. For this, we use the number of rotations $n$ we obtained while aligning the Tag to its upright position. The projection matrix is found using the steps given in supplementary document. We now rotate the image in the opposite direction $n$ times. The image is considered to be in the horizontal axis (i.e, $z = 0$) and the homography is applied over all the frames such that the image is now overlayed on the tag as shown in Figure 5

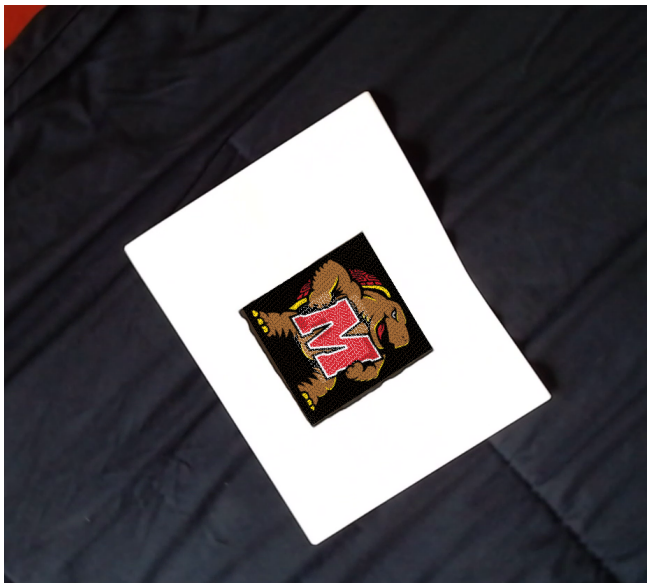


Fig. 5: The given image warped onto the video

### B. Virtual Cube

For this part of the problem, a cube with side length same as the reference image (200 pixels in the implementation) is projected onto the image plane. The $x, y$ coordinates of the reference image are taken as the bottom four world coordinates of the cube with $z = 0$. Similarly the top corners are constructed with $z = 200$. The product of the projection matrix and the world coordinates of the cube results in the pixel coordinates of corners in the image. These corners are connected to obtain the virtual cube on the image.

## III. DexiNed

This section compares edge detection results obtained using DexiNed and Canny Edge detector on images taken from KITTI odometry dataset. The images are resized to 512x512px and the output is recorded for both the edge detection approaches. The fused image from six outputs of the upsampling blocks are shown in the Figure 7

Canny edge detector takes only into consideration only the intensity changes to detect edges. Hence, we can see very minute changes in the intensity as being detected as edges. This corresponds to a lot of noise in the form of unwanted edges. On the conrtrary, since DexiNed has been trained over ground truths having only the necessary edges, it gives better results. We can see from the outputs that most of the edges obtained are clean.

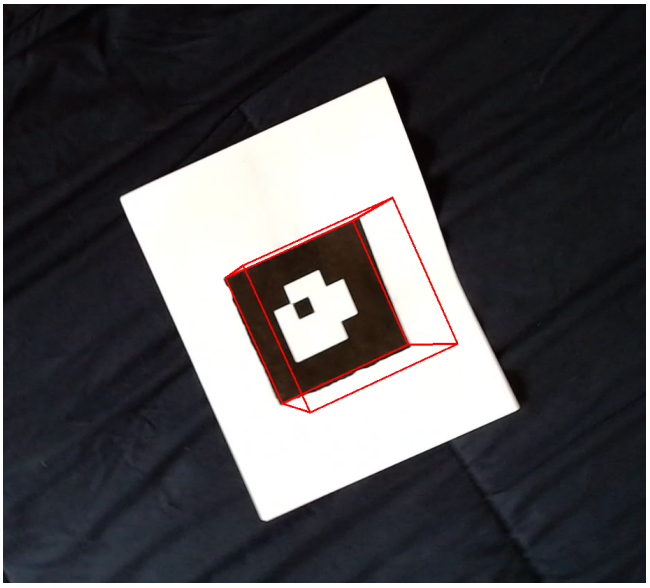


Fig. 6: The virtual cube placed over the AR-Tag

## IV. Conclusion and Future Work

The tag corners are not detected accurately in all the frames. There presented corner detection approach suffers from the pixels inside the tag that are chosen as potential candidates. This could be reduced by increasing the number of iterations the algorithm is run, but will lead to more processing time. This is the bottleneck of the pipeline. If this could be made accurate and efficient, further steps would not suffer from its disadvantages. $cv2.findContours$ and $cv2.contourArea$ could be used to replace this in future. Also the Testudo image when warped onto the image, has missing pixels leading to gaps in the image. Interpolation algorithms need to be explored in order to avoid this.

## References

[1] https://arxiv.org/pdf/1909.01955.pdf

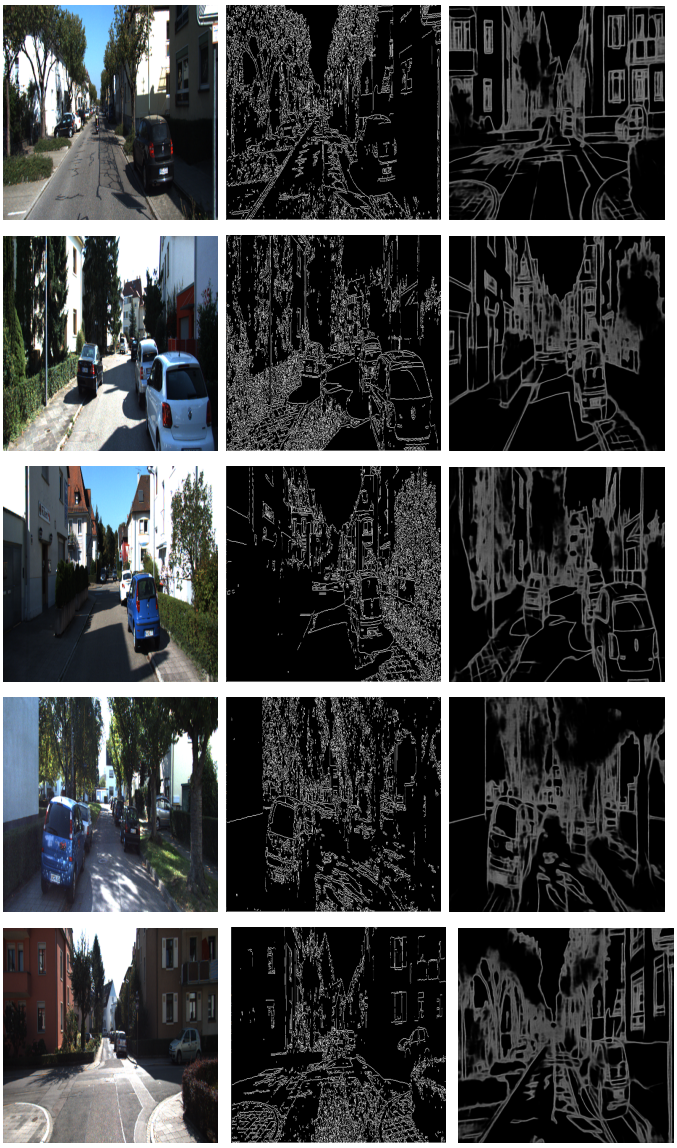


Fig. 7: Left to Right: Original Image from KITTI dataset, Edges using Canny detection, Edges using DexiNed