# ENPM673

Mano srijan Battula
M. ENG. Robotics
University of Maryland, College Park, MD
Email: mbattula@umd.edu

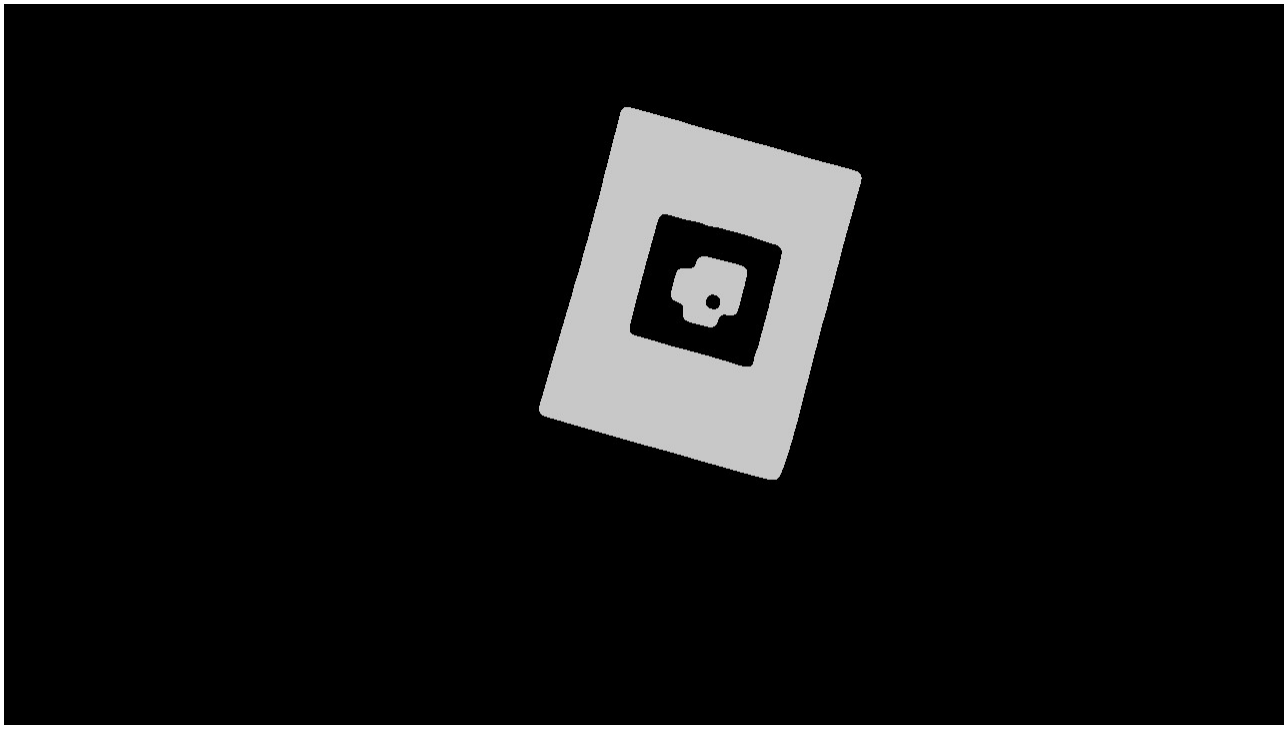
**ABSTRACT**

In this project we focus on detecting and tracking a custom AR Tag in a video sequence. The detection part of the project focused on finding the location and identity of the AR Tag. The tracking part of the project involved superimposing an image and placing a 3D cube over the tag. For improving the estimated homograph, Harris corner detection was applied to keep a track of four corners of the AR Tag

## 1. PART 1 – DETECTION

### 1.1 Filtering the image

The video was converted to a gray-scale color space In order to make the detection smoother as three channels is converted to a  binary channel after the gray-scale image was then blurred out using a Gaussian blur with values of 27,27 to remove the unwanted noise and to just focus on the edges. Later, Gray-scale is thresholded using in built threshed function and using a trail and error method to determine the values which will reduce the noise and give a better lines without noise so that just the AR tag is clearly visible in the video and eliminated all undesired image elements. You can view a threshed version of the video

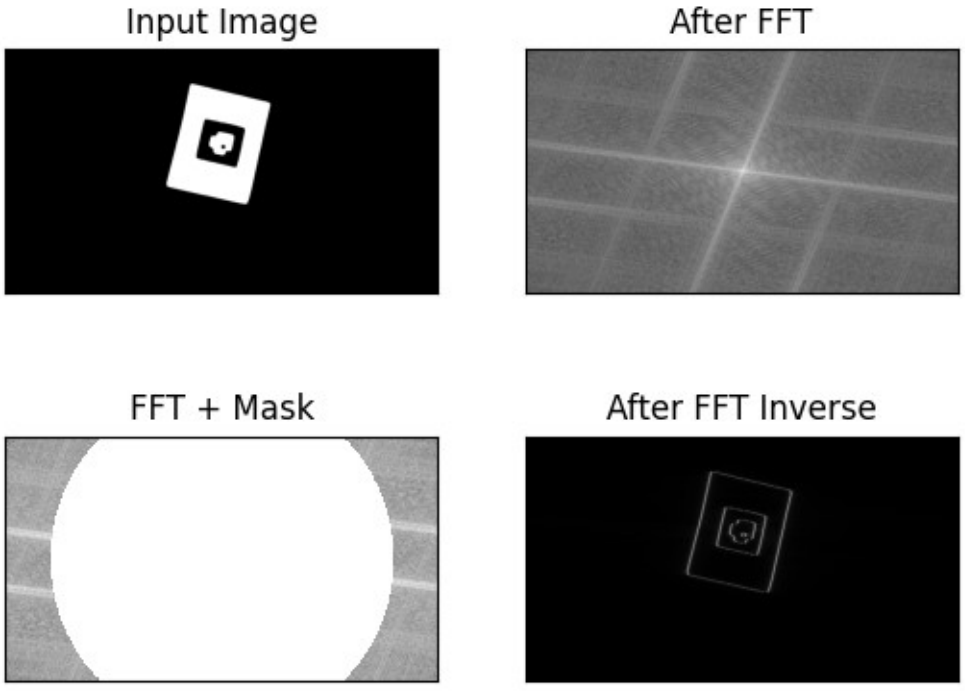


### 1.2 Fast Fourier Transformation and Edge detection

A Fourier transform helps in breaking down a incoming signal into its building blocks. Fourier transform can reveal important characteristics of a signal, namely, its frequency components.

The edges of an image are high frequency signals. If we filter out the low frequency signals from an image, we can obtain the edges. Inbuilt FFT and inverse FFT is used to calculate FFT and Inverse FFT. A circular masking (High pass filter) with radius of 500 is passed to the Fourier transformation by multiplying the mask with fft and the resultant output is converted back to spacial domain using Inverse FFT function. The following steps were followed to do the FFT and detect the edges:

1) Compute FFT for a gray-scale(blurred then threshed) image.
2) Define a circular mask(High Pass Filter) and multiply it with the FFT from step 1. to filter out low-frequency signals.
3)FFT and circular mask are multiplied
4)inbuilt FFT function is called to
5)Compute the inverse of FFT to get the edges of tag.



### 1.3)Getting Corners using Harris corner

Once the edges are detected and noises are removed. I applied Harris Corner detection which finds the white points in the corner to detect the points so I gave the values of the pixels in which it detects  corners of the tag but as it was detecting the outer corners of my tag. I had to create a new logic to detect the inner corners , used (found points which are less than 0.6 of distance ) of the tag where I have to superimpose my image later on.


### 2.Decoding AR Tag

Steps to decode the AR Tag:

1) Convert the image to binary and resize it to a size of 160 × 160.
2) Divide this 160 × 160 into 8 × 8 grids, i.e. each block in the grids is of size 20 × 20.
3) Take the median for each block and replace the 20 × 20 block with a single value depending on the median. It will be either 255 or 0 since the image is binary.
4) Extract the central 4×4 grids which have the rotational and ID information.
5) Check for the values at the four corners. Only one of them should be high. If values at more than one corner are high or no values are high, then AR-tag cannot be detected, and hence shall be discarded.
6) The upright position of the tag is determined by the bottom right grid. Keep rotating the tag till the bottom right grid is high.
7) After the orientation is set, extract the central 2×2 grids for the tag ID.

8) Flatten the 2 × 2 matrix. The leftmost digit is the least significant bit and the rightmost is the most significant bit. Convert it to decimal to get the tag ID

The white corner position gives the position of the tag and give a value of zero and center as 1.

**2.1 Homography Computation:**

Homography is a transformation that maps the points in one point to the corresponding point in another image. it is a 3x3 matrix. We normalized with Z co ordinate to create to 2d dimension.  To find a homography matrix that will allow us to convert between our camera coordinates and the square coordinates we need eight points in x,y co ordinates. Four of the points are the inner corners of the tag  and The other four points are the corners of our destination image. The dimensions of the destination are somewhat arbitrary but if it is too large we will start to have holes, since there are not enough pixels in the source image, and if it is too small we will have a lower resolution and will start to lose information. To fix this we found the number of pixels that were contained within the four corners of the tag and took the square root of that number to get an ideal dimension dim. The four corners are then [(0,0), (0, dim), (dim, dim), (dim, 0). I defined a homography matrix which takes 2 variables as input, which are of testudo tag points and inner corner points.

Now that we have the eight points we can generate the A matrix below:

$$A = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1 * xp_1 & y_1 * xp_1 & xp1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1 * yp_1 & y_1 * yp_1 & yp1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2 * xp_2 & y_2 * xp_2 & xp2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2 * yp_2 & y_2 * yp_2 & yp2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3 * xp_3 & y_3 * xp_3 & xp3 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3 * yp_3 & y_3 * yp_3 & yp3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4 * xp_4 & y_4 * xp_4 & xp4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4 * yp_4 & y_4 * yp_4 & yp4 \end{bmatrix}$$

From this A matrix we can solve for the vector $h$, which is the solution to:

$$Ah = 0$$

By reshaping our vector $h$, we can find the homography matrix:

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

To solve for the vector h, I used inbulit Singular Value Decomposition function .This will decompose A matrix into three matrices U, Σ,V. Inbuilt numpy function is used to calculate SVD and the corresponding homography for the four corners were found.

### 2.2 Creating the Square Image

After the homography matrix has been computed,we need to generate a new set of points in the square frame, and then create a new image using the pixels from source image in their new location. For any point in the camera coordinates Xc (x,y), we can compute that point in the square coordinates Xs (x′,y′), using the following equations:

### 2.3 Warping:

After finding homography from source coordinates to warped(destination) coordinates, i performed warping by multiplying the source coordinates to the obtained homography matrix. I defined a warp perspective function to do warping with three variables which take image, homography and size of image as inputs. to find which coordinate of the warped image matches with which coordinate of the source image, and then using the pixel value from the source image and pasting it back on the warped image. This way we

ensure that all the pixels of the warped image are covered, Thus pixel is left empty no empty.

**3. Superposing Testudo Image**

To place the Testudo image on top of the tag, I created a blank black image which is placed on the AR tag where the testudo will be super imposed. I calculated Homography matrix by using Testudo corners as source coordinates and the detected AR tag corners as destination coordinates. We apply this Homography on the Testudo image to obtain transformed coordinates. To superimpose Testudo on our original image, we overwrite the pixel values of our original image at the transformed pixels with the corresponding Testudo pixel values.

After we have determined the position, orientation, and ID of the tag in a given frame, the Testudo can be superimposed onto the tag. The steps to do this are as follows:

**1)**. **Match the tag ID to an image.**

The IDs of the tag and Testudo photo is hard coded at the start of the program.

**2). Rotate the image**

The function we use to decode a tag returns a number of rotations necessary to match the rotation of the tag. This number is either 0,1,2,3. We can then apply either no rotation or a multiple of 90à to the Testudo image using the function cv2.rotate()
(a) Upright Image (b) Right turned image

**3). Use homography to generate a new image**

Use the same process as above to generate a warped version of our Testudo image. In this case however instead of generating a square image of arbitrary size, instead create an image that is the same size as the original frame. When we generate our new homography matrix we need to use the corners of the frame in addition to the corners of the tag. We can then follow the same process as in equation: 2.5, to generate a new image that is the same size as our frame but has a warped image of the dog and is black everywhere outside of the frame (Figure 8a).

**4). Blank the original region of the tag**

In order to perform the next step we need to blank the region of the image where the tag is. We can achieve this by drawing a filled black contour in the region defined by the corners of the tag.

**5). Add the new image to the existing frame**

With the image generated from homography and the original frame with the blank region we add them together. Since the first image (Figure ) is black (0) everywhere except the tag and the second image (Figure ) is black (0) only in the tag region, the output is the Testudo superposed onto the tag

**3)Placing a virtual cube onto tag:**

Since a cube is a three dimensional entity, we need a 3 × 4 projection matrix to project 3D points into the image plane. The basic assumption while calculating the homography matrix is that the points are planar, i.e. z = 0 for all the points. Now, we have z values as well which we divide to normalize the equation. I'm using inbuilt line function to create a cube by defining the lines where these 16 lines intersect will give a cube. Thus we need to modify this H matrix to suit our requirements. To steps to obtain a projection matrix(P) from a Homography matrix(H) and camera calibration matrix(K) are as follows:

$$P = K[r1, r2, r3, t].$$