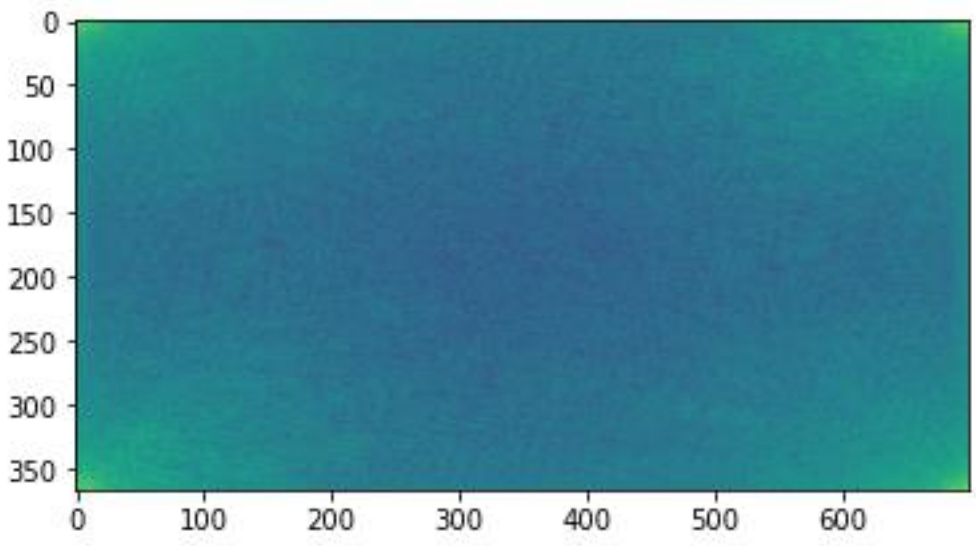
ENPM673

Project 1 Report

- **Problem 1.a – AR Code Detection:**

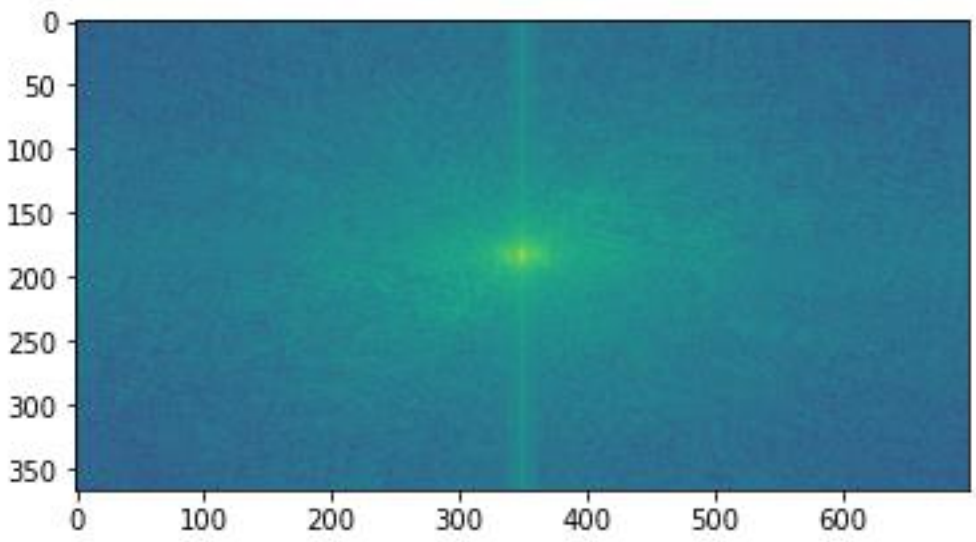
  - Steps performed for the output of this step:

1. Implemented Fast Fourier Transformation to transform gray scaled image into frequency spectrum:



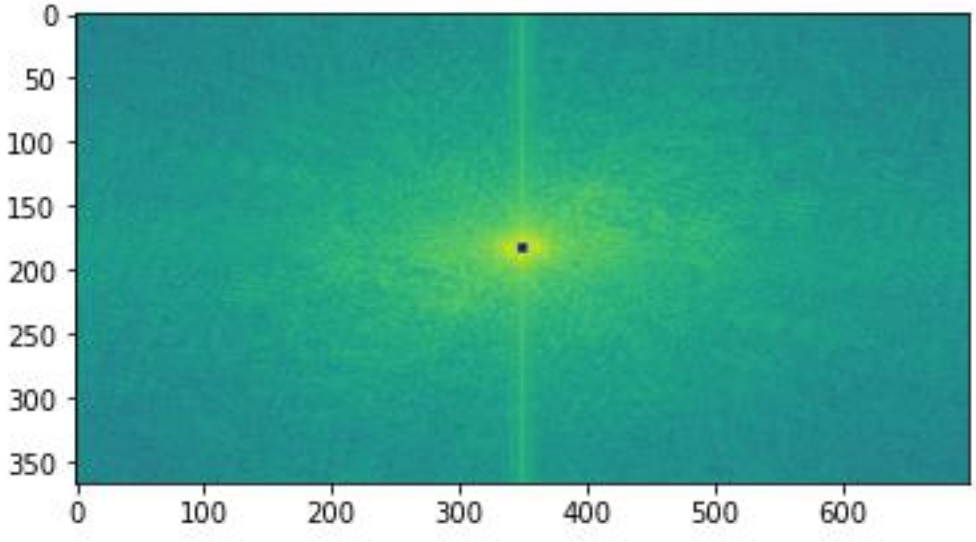
The symmetric white areas in the four corners represent the low frequency component. These white areas indicate that there is high energy in low/zero frequencies.

2. Visualized and Centralized the zero-frequency component:



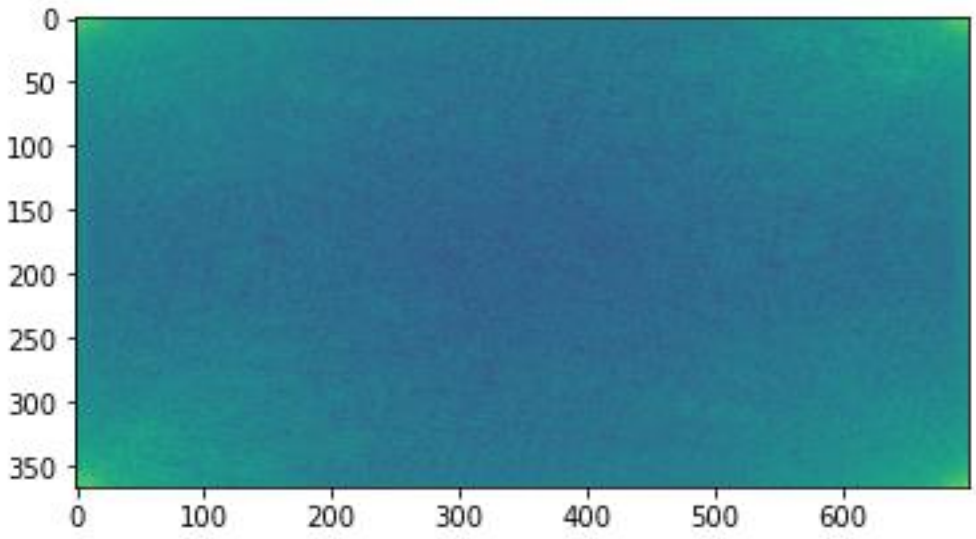
The low frequency component of the image shifted at the center of the frequency spectrum. 2-D FFT has translation and rotation properties, so we can shift frequency without losing any piece of information. I shifted the zero-frequency component to the center of the spectrum which makes the spectrum comprehendible. Moreover, this translation helps in implementing the high pass filter.

3. Applied high pass filter to filter frequencies:



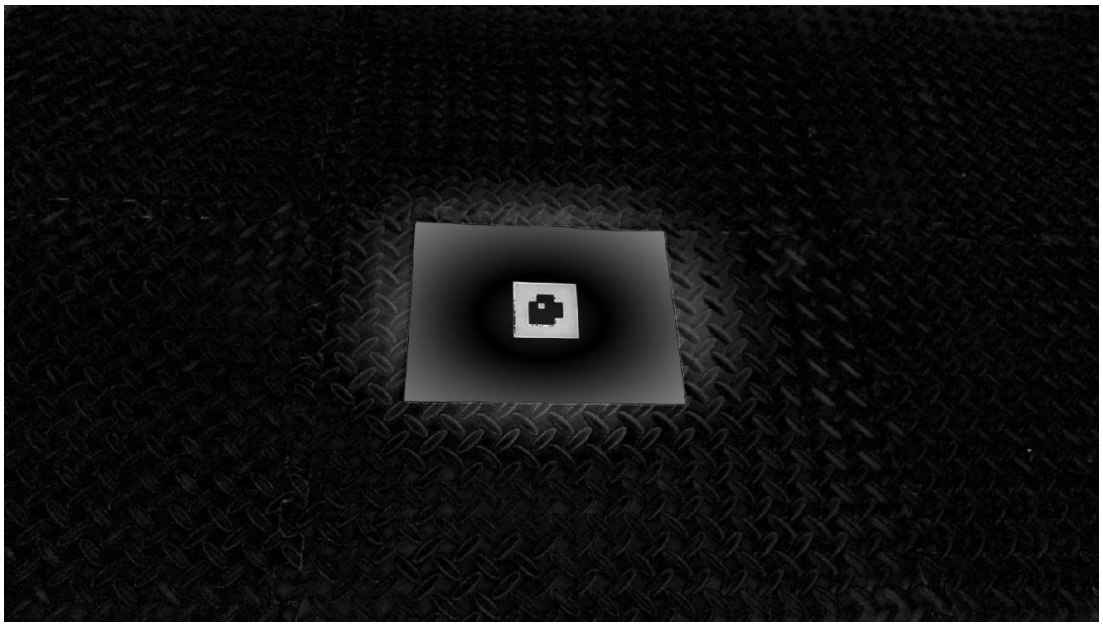
Applying the High Pass Filter to remove the low frequency content.

4. Decentralize:



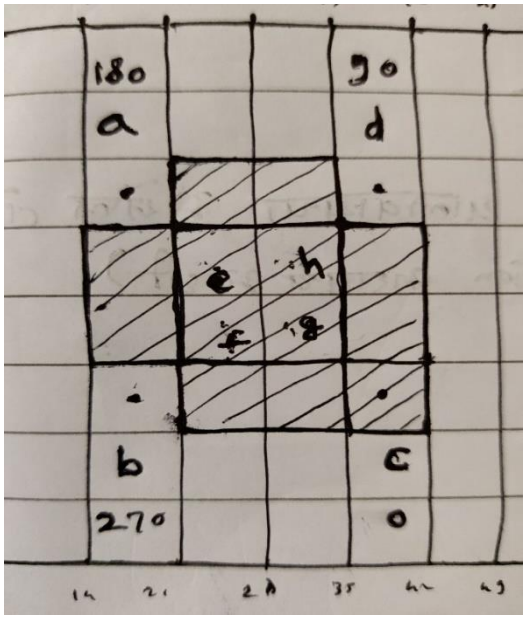
The processes of step 3 and step 4 are converting the information from spectrum back to gray scale image. It is done by applying inverse shifting and inverse FFT operation.

5. Implement inverse Fast Fourier Transformation to re-generate the original image:



Problem 1.b – Decode custom AR Tag:

- To detect the orientation and decode the data, I have decomposed the reference AR Tag image into 8 X 8 square grids.



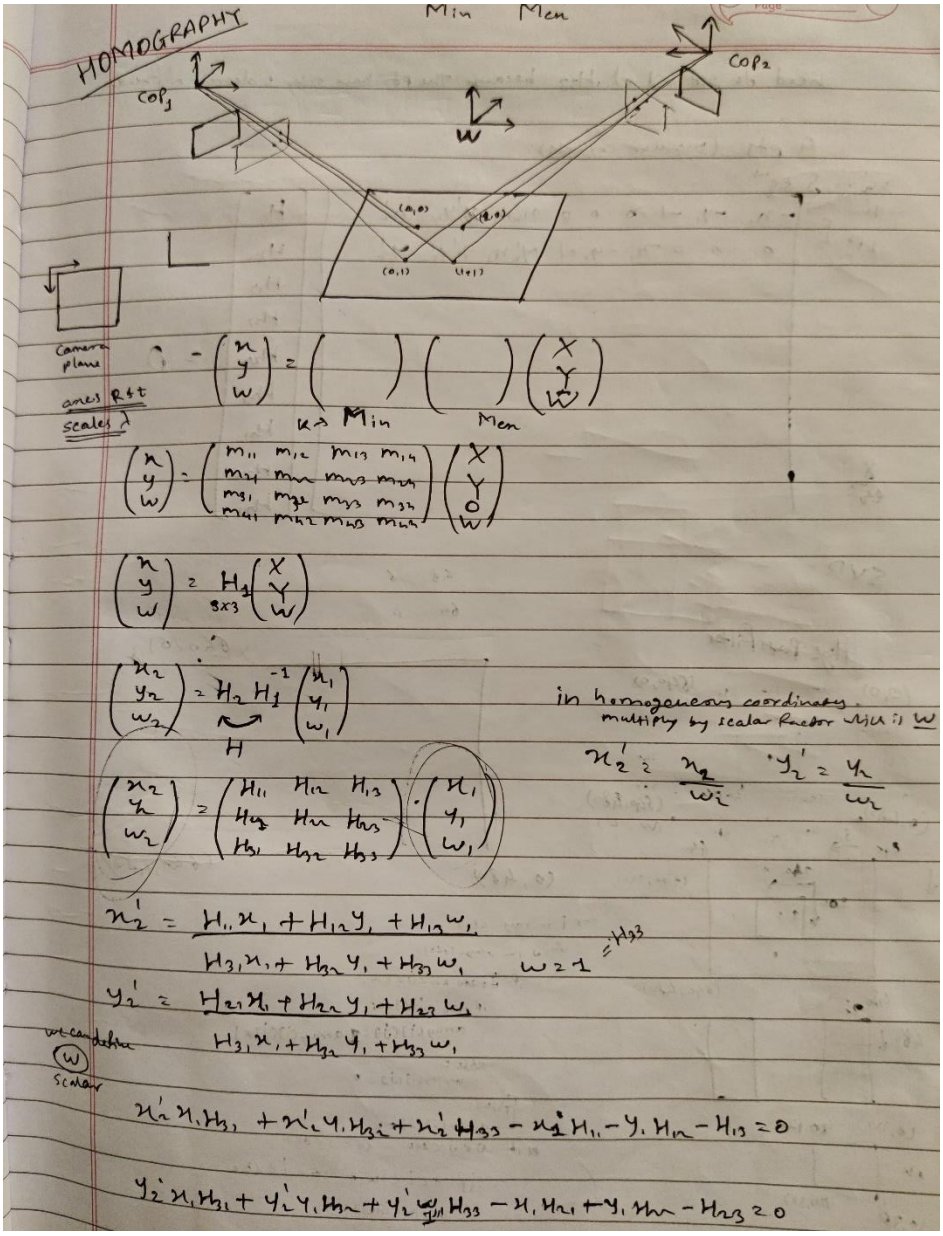
- The center pixels of the inner 4 X 4 grid and the inner-most 2 X 2 grid are considered to calculate the ID and the Orientation of the AR Tag. The values of pixels a, b, c, d contribute to calculating the orientation while values of pixels e, f, g, h are considered for the ID calculation.
- White pixel at C = 0 Degrees; White pixel at D = 90 Degrees.
- White pixel at A = 180 Degrees; White pixel at C = 270 Degrees.
- Pixel F = MSB (Most Significant Bit), Pixel E = LSB (Least Significant Bit)

```
ID :  [0, 1, 1, 1]
Orientation :  270 Degrees
ID :  [1, 1, 1, 0]
Orientation :  0 Degrees
ID :  [1, 1, 1, 0]
Orientation :  0 Degrees
ID :  [0, 1, 1, 1]
Orientation :  270 Degrees
ID :  [0, 1, 1, 1]
Orientation :  270 Degrees
ID :  [0, 1, 1, 1]
Orientation :  270 Degrees
ID :  [0, 1, 1, 1]
Orientation :  270 Degrees
ID :  [0, 1, 1, 1]
Orientation :  270 Degrees
```

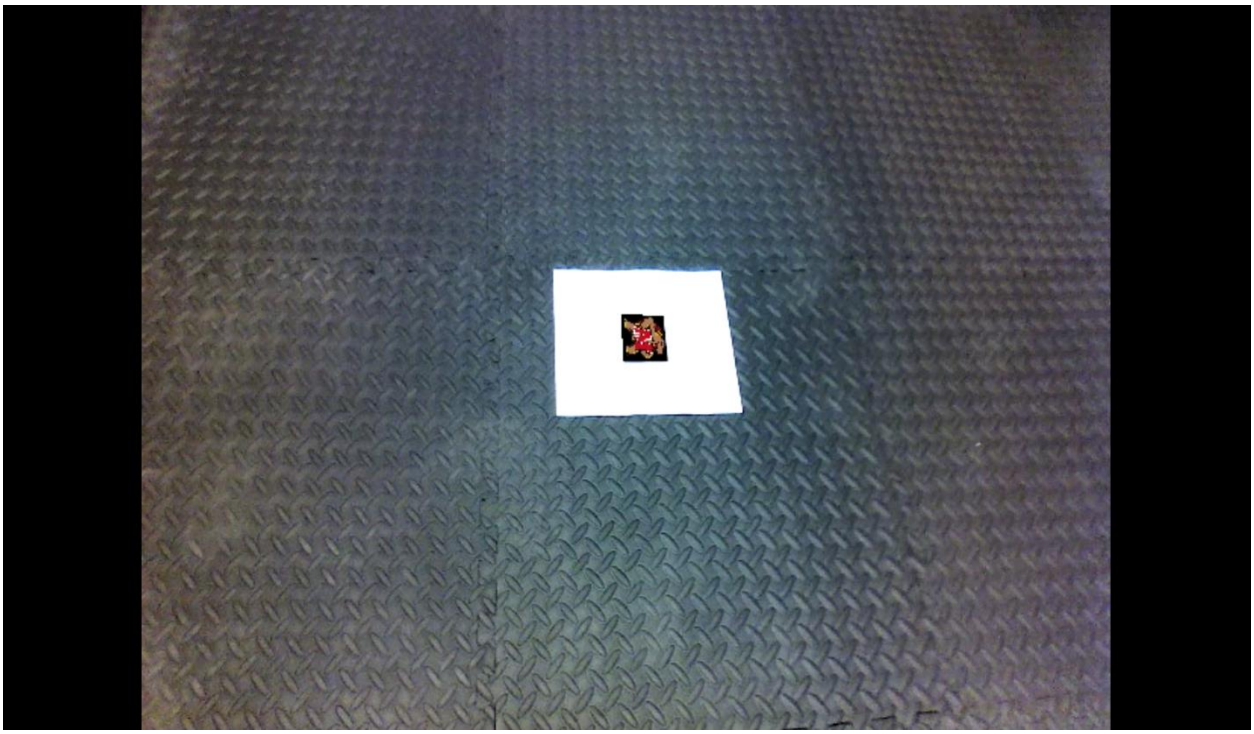
Problem 2.a – Superimposing Testudo on Tag:

- **YouTube Link of the output video: Link**

- This problem required the use of calculating the Homography Matrix between the points of camera and world planes.

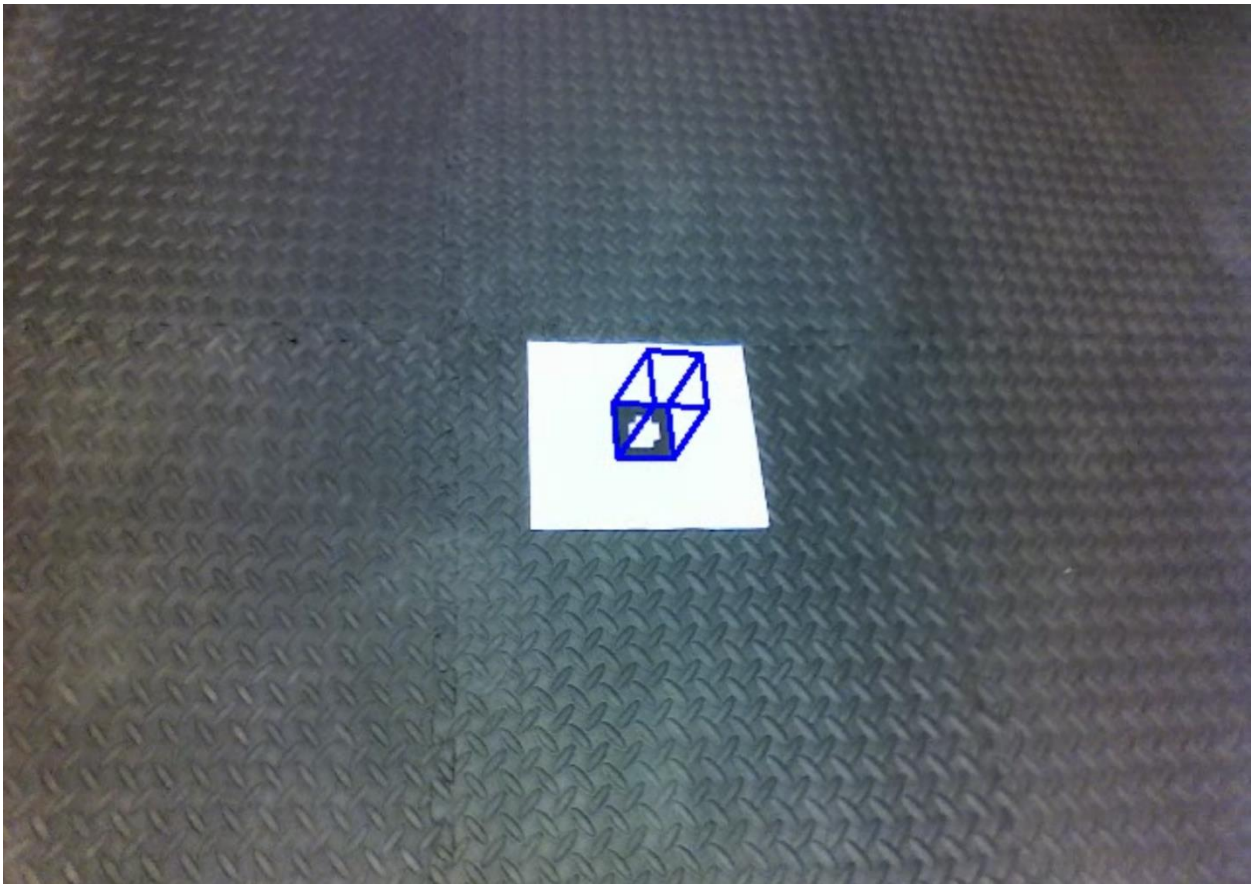


- In order to apply Homography, the corners of the AR Tag are detected in the image plane. This is achieved by first giving the FFT image to canny edge detector which gives the edges of the 8 X 8 grid of the tag.
- Then several features of the Contours method in OpenCV are used such as cv2.contourArea, cv2.arcLength, cv2.approxPolyDP to locate the four corners of the tag in each frame.
- These points are given to a Homography Matrix to strengthen the tag which can be used for calculating the ID and Orientation.
- After calculating the ID and Orientation, the World coordinates of the Testudo Image and the four corners found from the contour method are given to the Homography function again to calculate the homography between them to finally put all the pixel values of the original testudo into the tag.



Problem 2.b - Placing a virtual cube onto Tag:

- **YouTube Link of the output video: Link**



- For this problem, the Projection Matrix P was calculated using the equation, P = K R | t where K is the intrinsic camera parameter matrix and R and t are the Rotation matrix and translation vector respectively.
- The homography between the AR Tag points and the world co-ordinates of the Cube is given to the projection function to calculate the Projection Matrix with the help of K and calculated Rotation matrix and Translation Vector.
- First the Bnew Matrix is calculated which is then used to calculate the rotation and translation vector.
- The rotation matrix and translation vector is then scaled by Lambda to get them in unit length, after which r1, r2, r3 and t are concatenated to get the projection matrix.
- After this simple lines are drawn on the points gathered from the projection matrix to get the cube placed onto the TAG.