# ENPM 673 – Perception for Autonomous Robots

## Project 1 - Spring 2021

Diane Ngo – 117554960

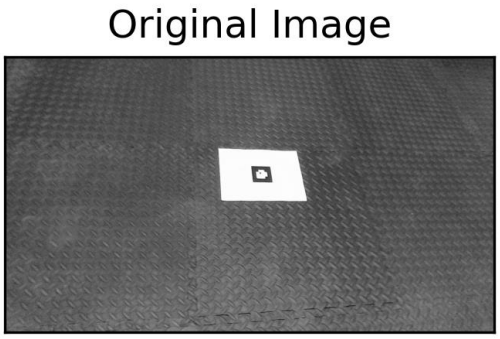
# Problem 1 – Detection

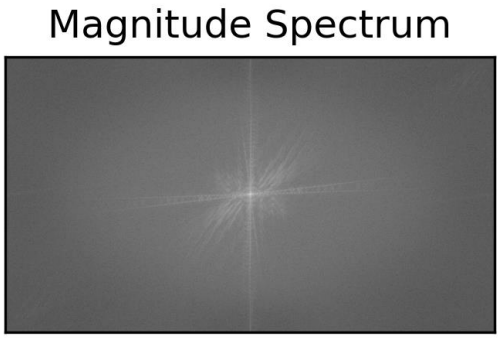
## 1a) AR Code Detection

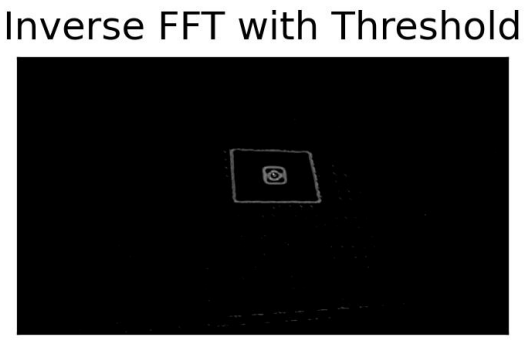
The task here is to detect the April Tag in any frame of Tag1 video (just one frame). Notice that the background in the image needs to removed so that you can detect the April tag. You are supposed to use Fast fourier transform (inbuilt scipy function fft is allowed) to detect the April tag. Notice that you are expected to use inbuilt functions to calculate FFT and inverse FFT.

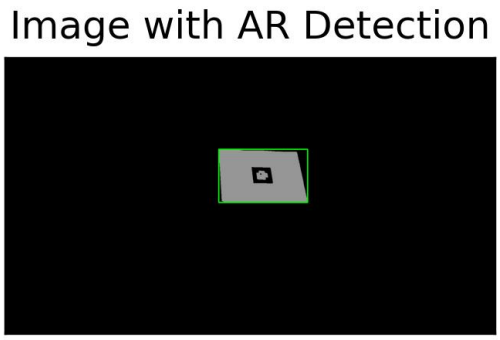
# Solution:

Starting with the Tag1.mp4 video, the first frame is taken and then converted into grayscale using the cvtColor(COLOR_BGR2GRAY) inbuilt function from OpenCV. The fast fourier transform is then utilized by Numpy's functions: fft2, fftshift, ifftshift, and ifft2. The fft2 function is the fast fourier transform and it works by taking the grayscale image as an array of integers and computes the 2-Dimensional discrete Fourier Transform on the last two axes, resulting in a complex array. In Python, the origin, or zero frequency component, resides in the top left corner. This component needs to be shifted to the center of the array, and is computed by the fftshift function. The magnitude spectrum is computed and shows the high and low frequencies found in the image. This is then operated on with a high pass filter and inverse fourier is used to shift the center back to the top left corner. Because the resulting array are complex numbers, the absolute value is taken. To detect the paper containing the AR code, the resulting Fourier image is filtered with a Gaussian Blur and threshold to zero using the cv2.THRESH_TOZERO function. The threshold image is converted to binary in order to detect the 1s (white) in the image and a rectangle is drawn as a bounding box for the paper.



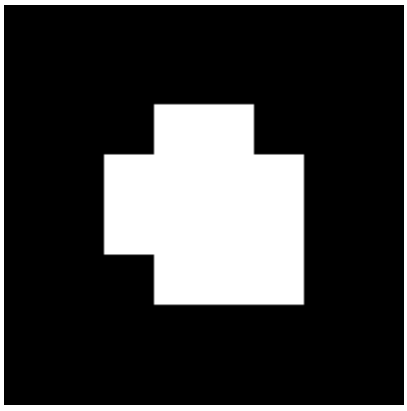



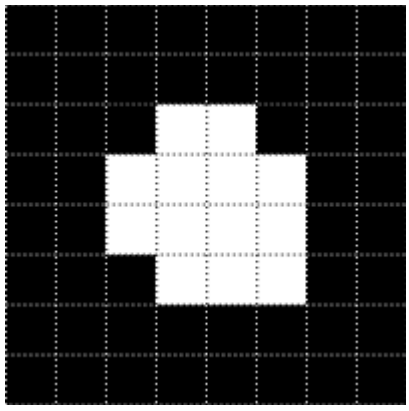




One interesting problem I had was when I got the magnitude spectrum, my image was just inverted colors. I had to set the inverse to solve for white. I had trouble getting the bounding box to properly work on just the inner AR Tag, and the angle is not correct.

## 1b) Decode Custom AR Tag

Given a custom AR Tag, it is used as a reference to determine the orientation and ID of the tag.



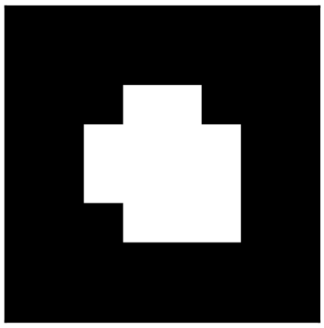


The marker is decomposed into an 8x8 grid of squares and then the outer 2x2 black cells are removed to give the inner 4x4 AR tag. Use any detection pipeline to return the proper orientation and ID of the tag.

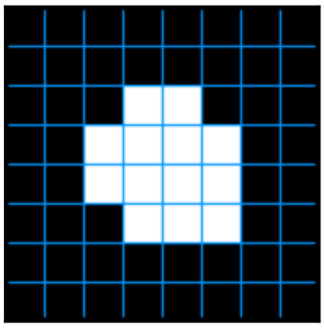
## Solution:

The reference marker is first imported into the program by cv2.imread() and then divided to 8x8 squares. A grid is drawn onto the image with cv2.line function. The outer black tiles are cropped out to result in the inner 4x4 grid. The program first checks the other square to determine which has the white corner, which is 255. The program outputs the orientation and angle of the tag. Then, the program checks the inner 2x2 grid to determine the Tag ID. White cells result in that cell having a "1" binary number, and black cells result in "0". The Tag ID is returned as a string of 4 binary numbers.

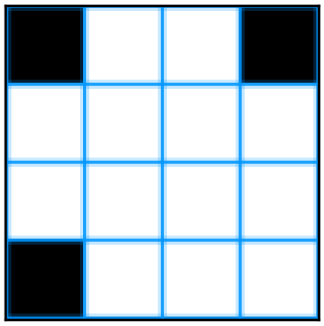
Reference Marker Image



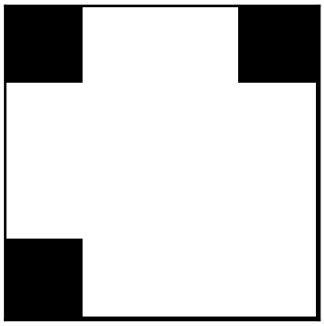
Marker with Grid



Cropped Image 4x4 Center



Threshold Binary Image



```
Drawing grid on the image
Cropping image to inner 4x4
Cropping image to inner 4x4
Checking outer border
==============================
Top left corner is not white
================================================
Top right corner is not white
================================================
Bottom left corner is not white
================================================
Bottom right corner is white
Tag is upright at 0 degrees
================================================
Checking inner border
==============================
(Inner) Top left corner is white
================================================
(Inner) Top right corner is white
================================================
(Inner) Bottom left corner is white
================================================
(Inner) Bottom right corner is white
================================================
AR Tag ID is [1, 1, 1, 1]
```

# Problem 2 – Tracking

## 2a) Superimposing image onto Tag

For this section of the project, we have the 4 corners of the tag from 1b, and are given a template image of Testudo to overlay on top of the tag. Its orientation is specified by the white corner in the tag. The purpose of this problem is to overlay testudo on top of the AR tag in the video.

# Solution:

The video of course is first imported into the program, filtered with a grayscale, and then threshold binary applied. Contours of the video are found through the findContours function. A user defined function called get_contours() returns the new threshold video with contours drawn on it, a list of final contours of the corners, and a list that has x, y coordinates of the corners. To find the square defining the AR Tag, I used the hierarchy given by the threshold function RETR_TREE, which gives parent and children contours. After getting the contours the program goes into a for loop to go through all the contours of the corners throughout the video. Source corners and testudo corners are converted to float format, and then passed through the user defined find_homography function. The homography is found by first following the equation given in the supplementary document. After calculating the A matrix, Singular Value Decomposition is found by the NumPy function which gives the V matrix.

$$\begin{bmatrix} x_k^{(w)} & y_k^{(w)} & 1 & 0 & 0 & 0 & -x_k^{(c)}x_k^{(w)} & -x_k^{(c)}y_k^{(w)} & -x_k^{(c)} \\ 0 & 0 & 0 & x_k^{(w)} & y_k^{(w)} & 1 & -y_k^{(c)}x_k^{(w)} & -y_k^{(c)}y_k^{(w)} & -y_k^{(c)} \end{bmatrix} \begin{bmatrix} \boldsymbol{h}_1^T \\ \boldsymbol{h}_2^T \\ \boldsymbol{h}_3^T \end{bmatrix} = \mathbf{0}_{2\times 1}. \quad (5)$$

The last column of the V matrix gives the homography matrix elements. It needs to be normalized. The H matrix is then shaped into a 3x3 matrix using the reshape function.

$$\boldsymbol{h} = \frac{[v_{19}, \ldots, v_{99}]^T}{v_{99}}.$$

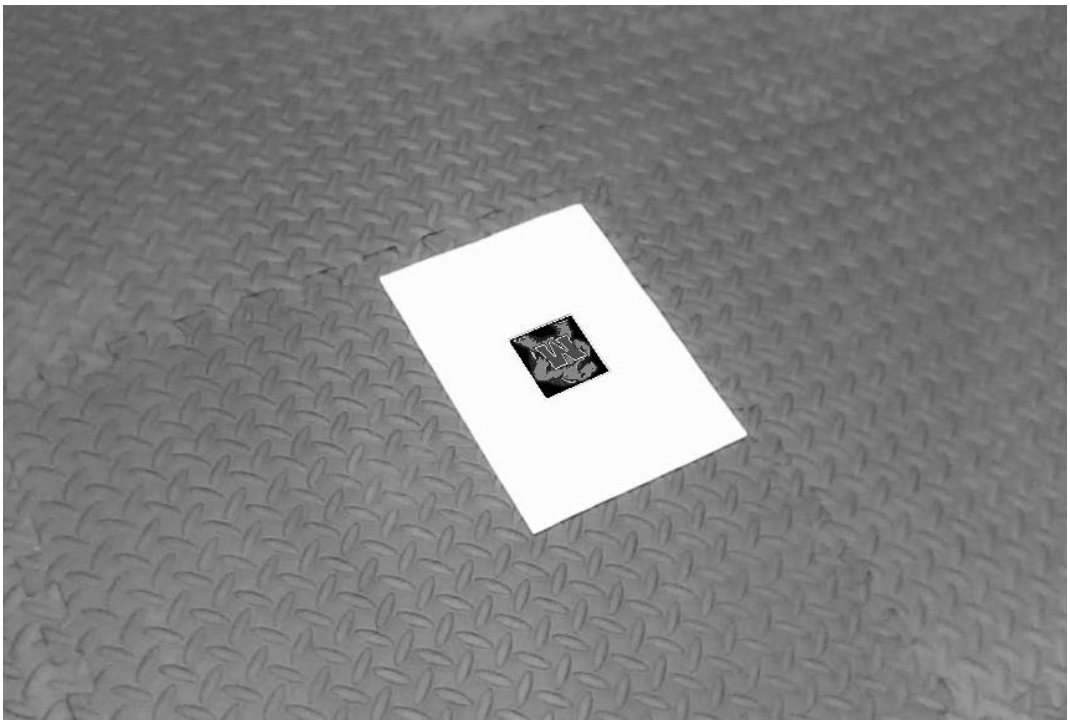
After finding the homography between the source and video locations, the warp perspective functions are then used. To mimic the OpenCV function, we start by inverting the homography variable, and then looped through the size of the image. The x, y, and z coordinates are multiplied with the [I, j, 1] matrix. The formulas are shown below.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ a_{31}u + a_{32}v + 1 \end{bmatrix} = \mathsf{w}$$

$$\frac{1}{w} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} u/w \\ v/w \\ 1 \end{bmatrix}$$

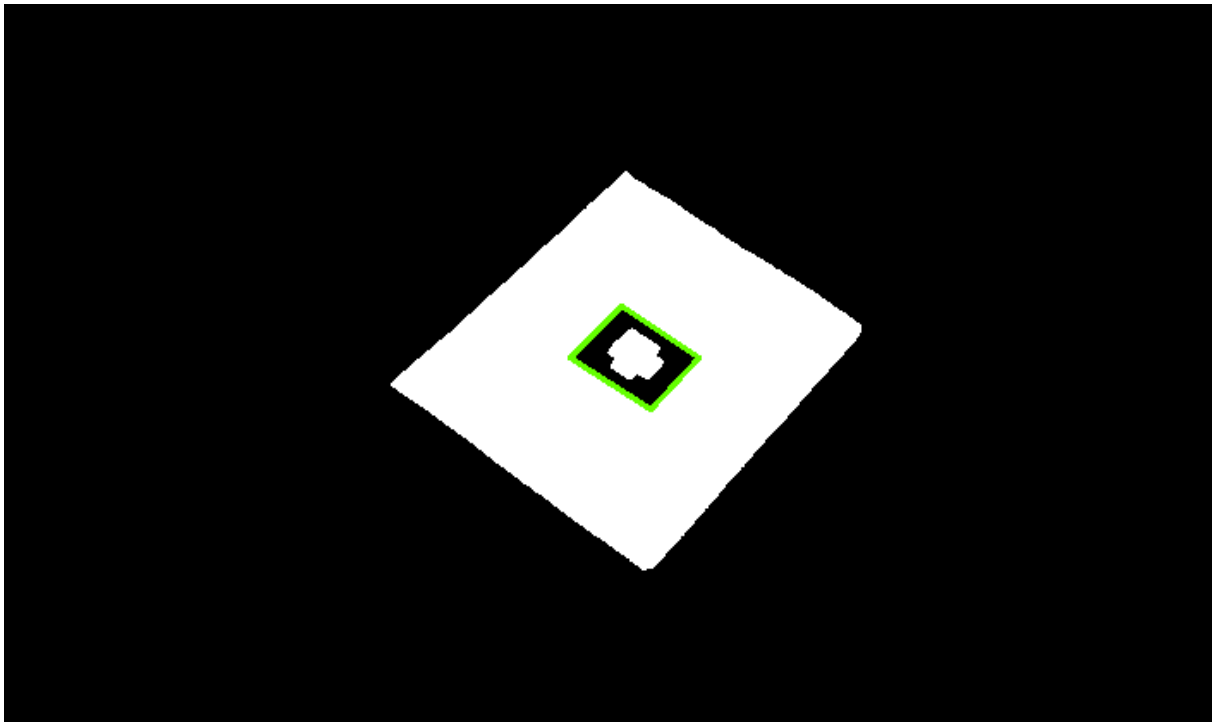
The warp perspective of the tag is cropped and then processed to find the orientation of the tag through get_ar_tag_info(). The function orient_pos then rotates the testudo image accordingly.

Images of the program are shown below.



Testudo on Image


Warped Perspective of AR Tag



Threshold with Finding Contours

I am new to Python so I had a lot of issues with this problem. At first, my warp perspective would just return black screens and I couldn't get the AR tag to be warped from a video. After research, I learned Python uses coordinates in the form of y, x, with inverted positions. Oh unfortunately the multiple tags video just doesn't work. It doesn't detect any contours.

**Videos**

**Tag0**:
https://drive.google.com/file/d/1GtFHqMCxo4GyUCzcnPGnpGDfWShCnWUi/view?usp=sharing

**Tag1**:
https://drive.google.com/file/d/1gqtbiMywAklbqHoiIyw5ziKZ41DccMWP/view?usp=sharing

**Tag2:** https://drive.google.com/file/d/1r9ZzIloTSYNz5riVJkn8jmIce63__ezc/view?usp=sharing

## 2b) Drawing Cubes

Using the supplementary documentation, I coded the projection matrix function by first calculating the homography and taking the first two rows. The camera intrinsic calibration matrix K was transposed. Lambda was solved by taking the multiplication of the h1 and h2 with the K matrix, divided by ½ and normalized. B hat was solved by multiplying lambda and the multiplication of the homography matrix and K. Seeing if the determinant of B was greater than or equal 0, B was then found and split into 3 rows to solve for the rotation matrix.

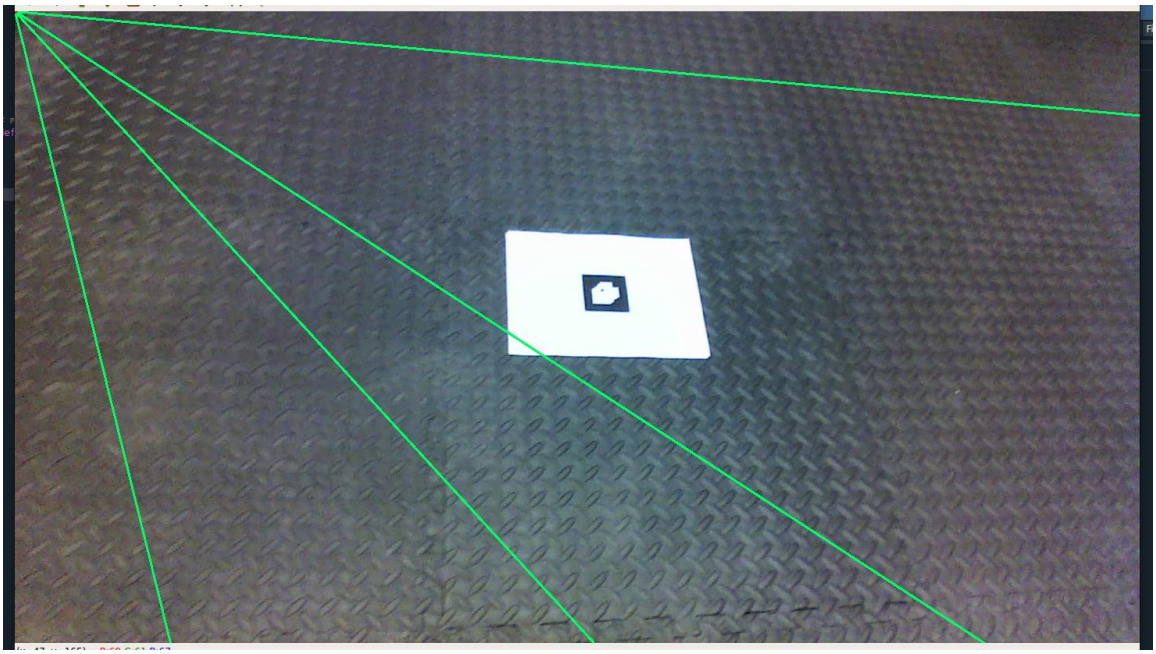
```python
def projection_mat(H, K):
    h1 = H[:,0]
    h2 = H[:,1]
    # h3 = H[:,2]
    K_inv = K.T
    lmda = 2/(np.linalg.norm(np.matmul(K_inv, h1)) + np.linalg.norm(np.matmul(K_inv, h2)))
    B_h = lmda * np.matmul(K_inv, H)
    det_B = np.linalg.det(B_h)
    if det_B > 0:
        B = B_h
    else:
         B = -1*B_h

    r_1 = B[:,0]
    r_2 = B[:,1]
    trans = B[:,2]*lmda
    # trans = B[:,2]*lmda
    # print("Translation\n", trans)
    # print(trans.shape)
    r_3 = np.cross(r_1, r_2)
    r_3 = r_3/lmda
    r = np.column_stack((r_1, r_2, r_3))
    r_t = np.column_stack((r_1, r_2, r_3, trans))

    proj_mat = np.matmul(K, r_t)
    return proj_mat, r, trans
```

The cube points are set to 8 points: axis_points = np.float32([[0,0,0,1], [0,200,0,1], [200,200,0,1], [200,0,0,1] , [0,0,-200,1], [0,200,-200,1], [200,200,-200,1], [200,0,-200,1]])

Then I need to take the axis points and project them on the plane. Unfortunately I couldn't get OpenCV to display the cube.

My draw cube function:

```python
def draw_cube(frame_copy, cnr_pnts):
    cnr_pnts = np.int32(cnr_pnts).reshape(-1, 2)
    # cv2.imshow("cube test", frame_copy)
    print("Points\n", cnr_pnts)
    frame_copy = cv2.drawContours(frame_copy, [cnr_pnts[:4]], -1,(0, 255, 100), 2)
    for i,j in zip(range(4),range(4,8)):
        fm = cv2.line(frame_copy, tuple(cnr_pnts[i]), tuple(cnr_pnts[j]), (100,255,0), 2)
    fm = cv2.drawContours(frame_copy, [cnr_pnts[4:]],-1,(0,0,255), 3)
    return fm
```



**Video 2b :** https://drive.google.com/file/d/1k1I4Wvuv8F-YXhdoXUvh1dFsPBoXOKiX/view?usp=sharing

**References**:


https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_transforms/py_fourier_transform/py_fourier_transform.html

http://docdingle.com/teaching/cs545/presents/p12b_cs545_WarpsP2.pdf

https://note.nkmk.me/en/python-opencv-numpy-alpha-blend-mask/

https://docs.opencv.org/3.4/d9/d8b/tutorial_py_contours_hierarchy.html

https://docs.opencv.org/3.4/d7/d53/tutorial_py_pose.html