Project1_ENPM673

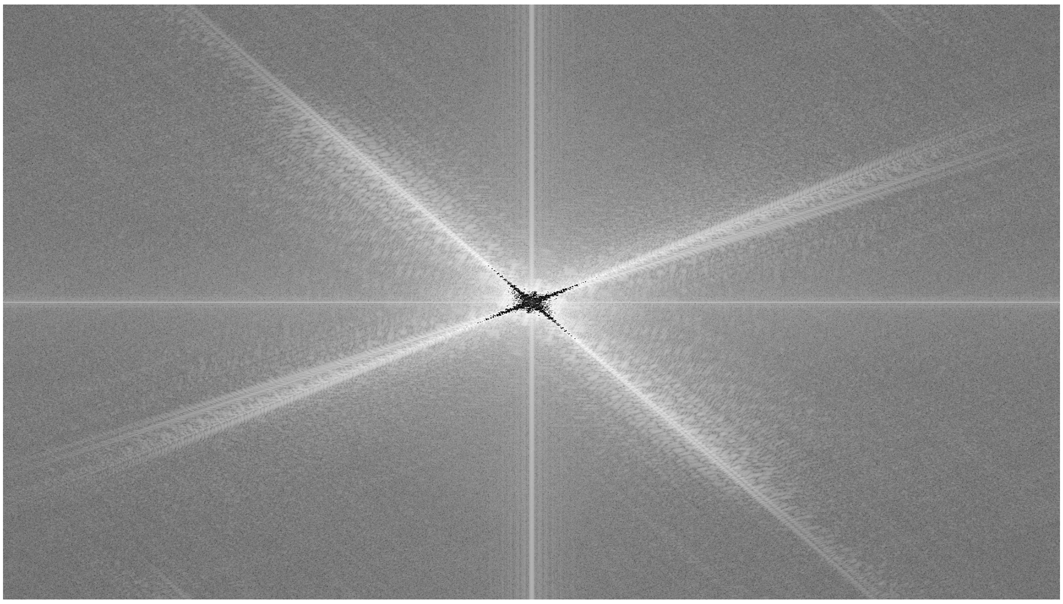
Problem 1 - Detection

1.a) AR Code detection

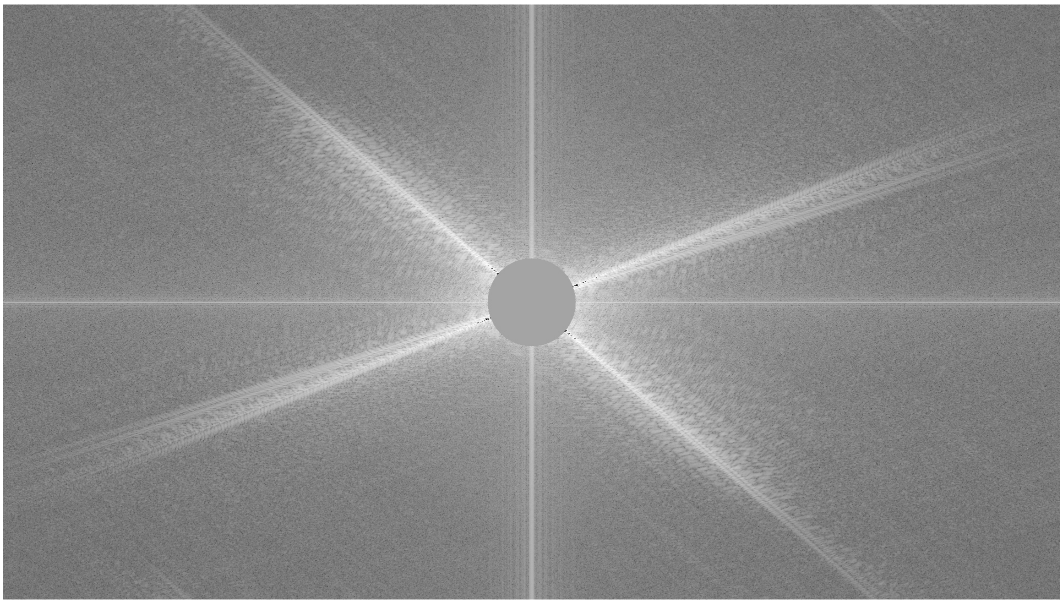
To detect the AR Code, the background of the picture must be removed.

This can be done through applying FFT to transofrm the image to the frequency domain.

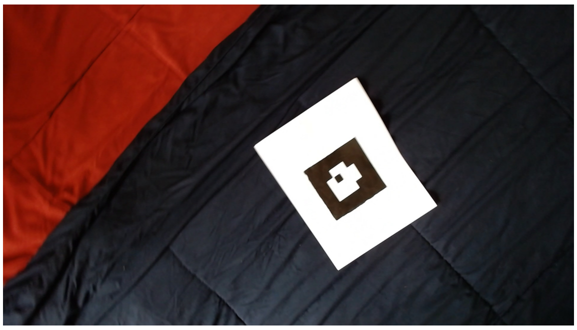
The center part of the frequency spectrum has lower frequency and the outer part has higher frequency.

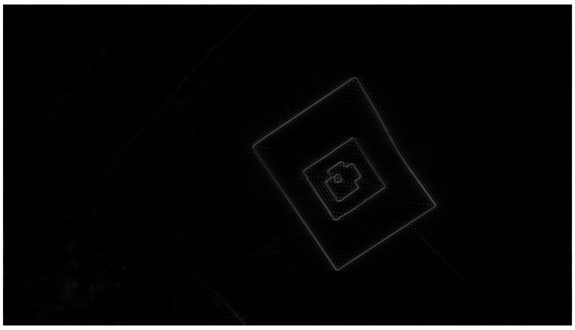


In the frequency domain, low frequency parts which represent the smooth background in the original image can be removed, and the high frequency parts which represent the lines in the image will remain.



After croping the frequency image, the filtered image can be made by transforming the image back from the frequency domain.

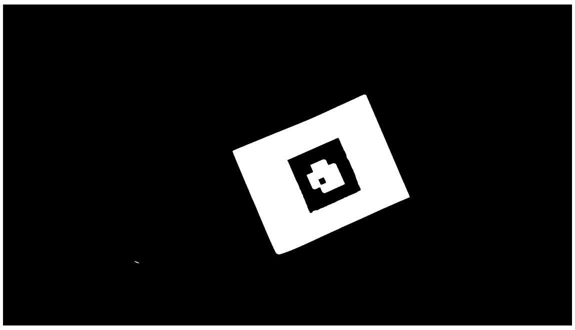




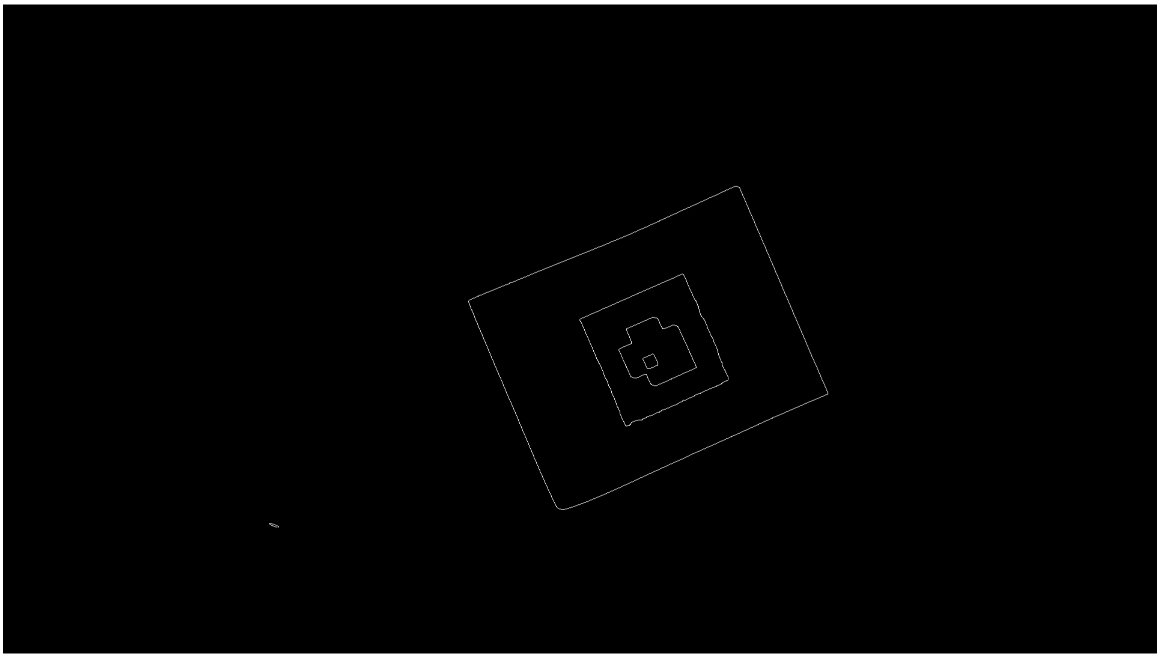
In the filtered image, the background is removed and the line on the AR tag is preserved.

1.b) Decode custom AR tag

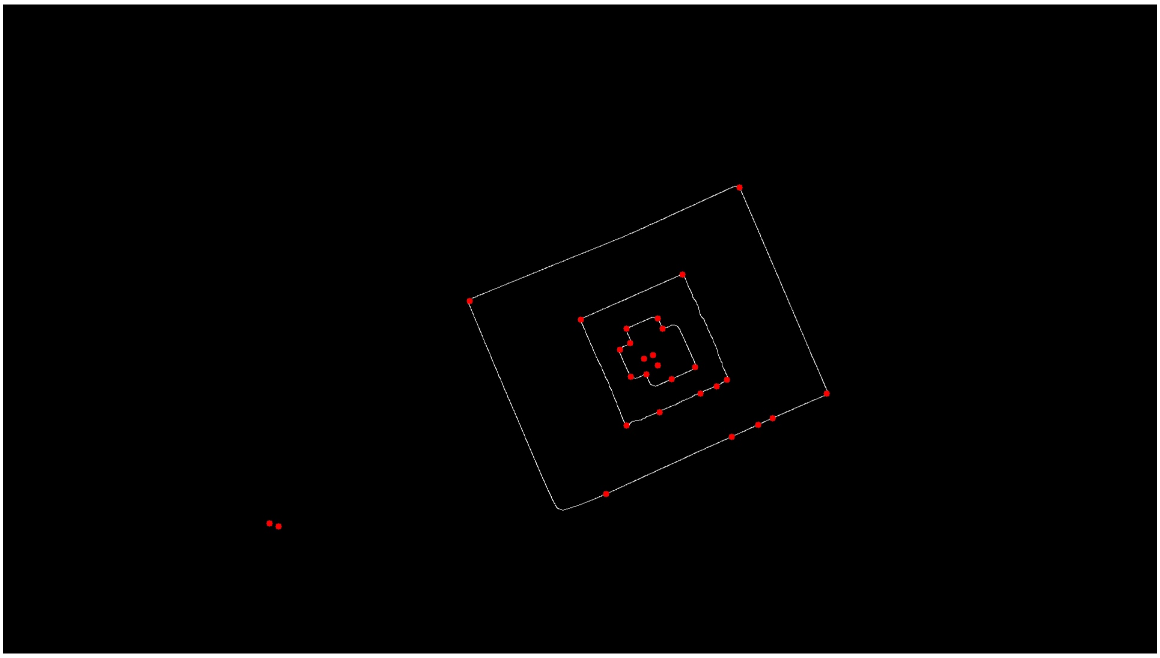
Step 1. Transform the image into gray scale then apply thresolds to get clearer image.

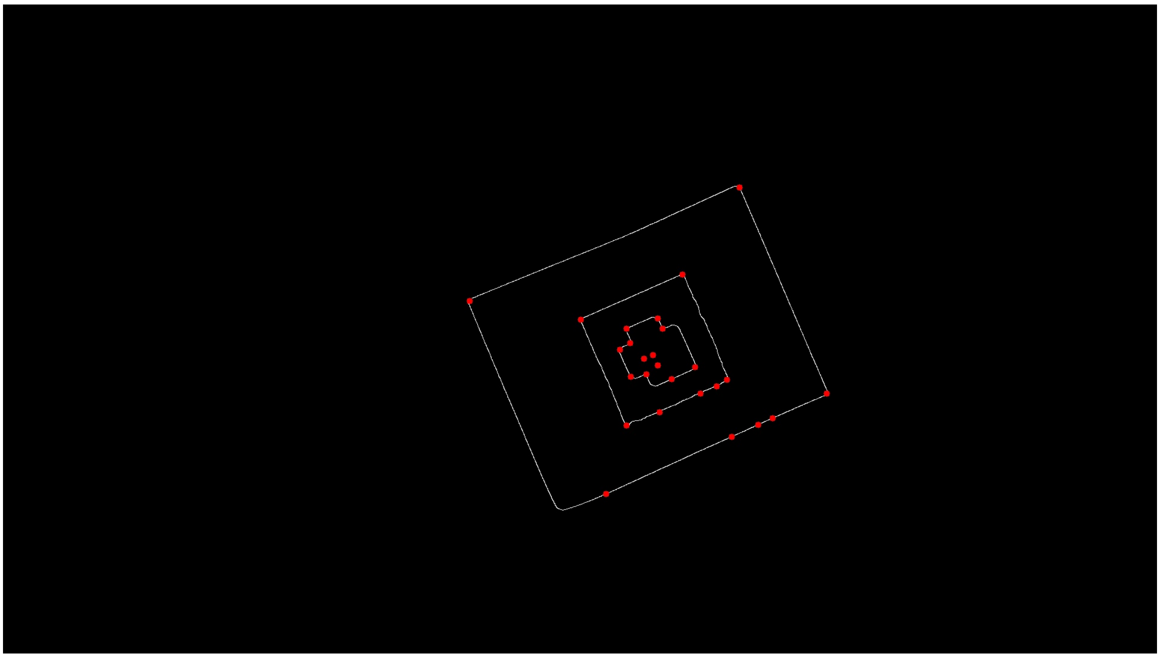
Step 2. Use canny to find the edges in the image



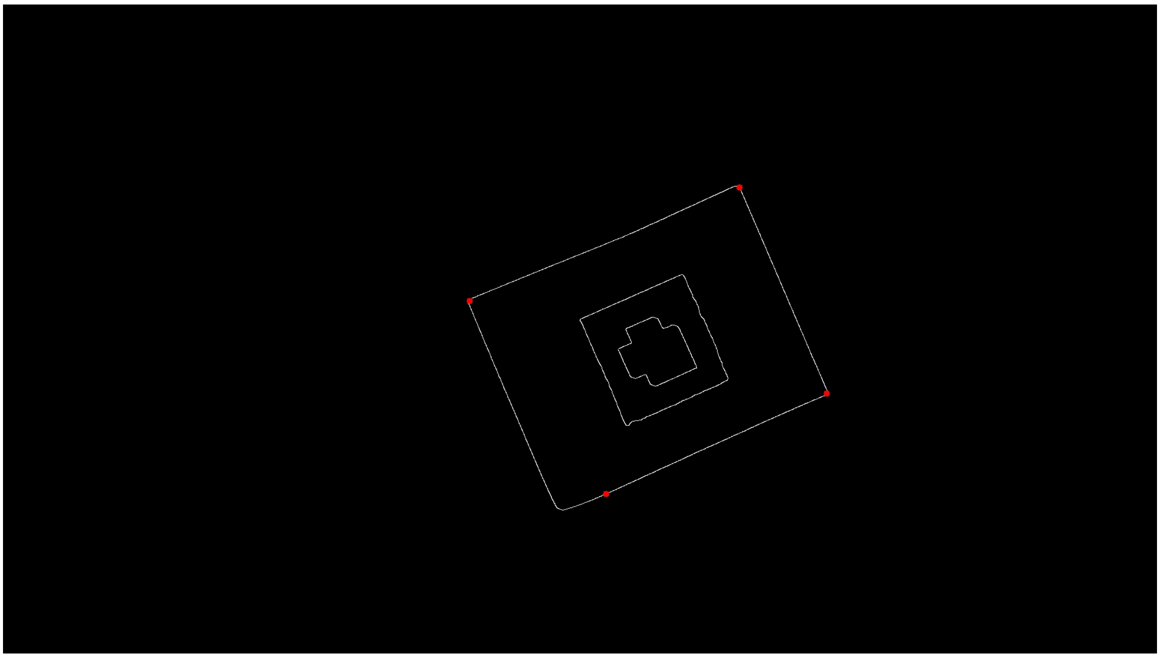
Step 3. Use goodFeaturesToTrack (Shi-Tomasi Corner Detector) to find the corners.



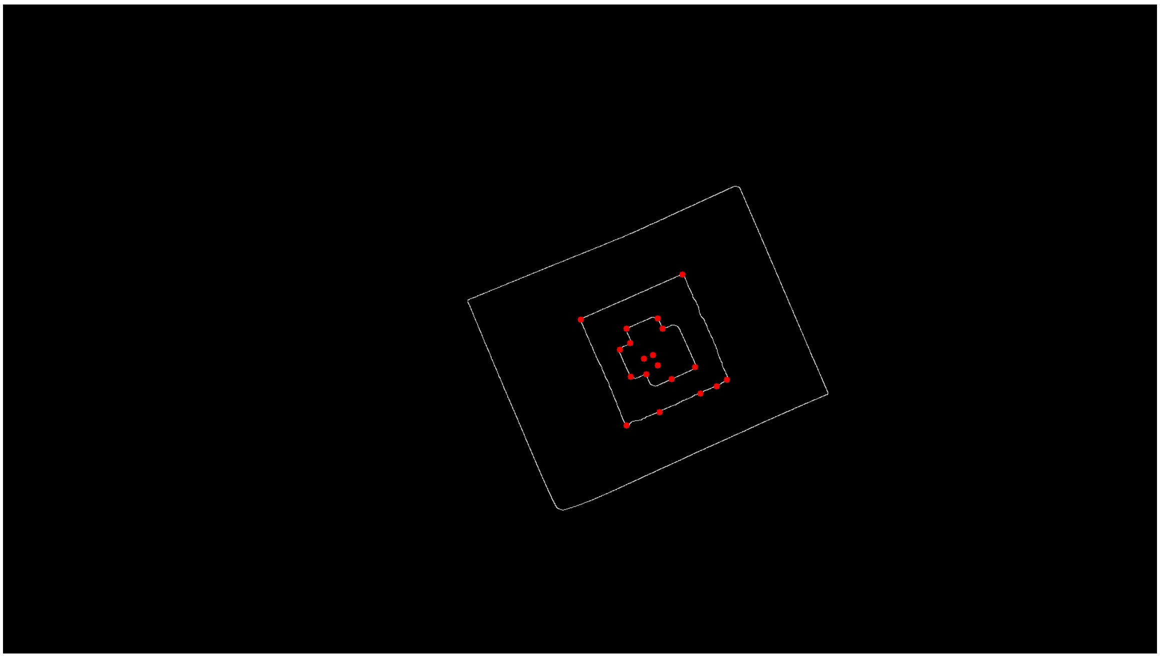
Step 4. Find the median row and column of the corners, and use them to reject outliers.



Step 5. Find the vertics by selecting the top, left, bottom, right-most corners

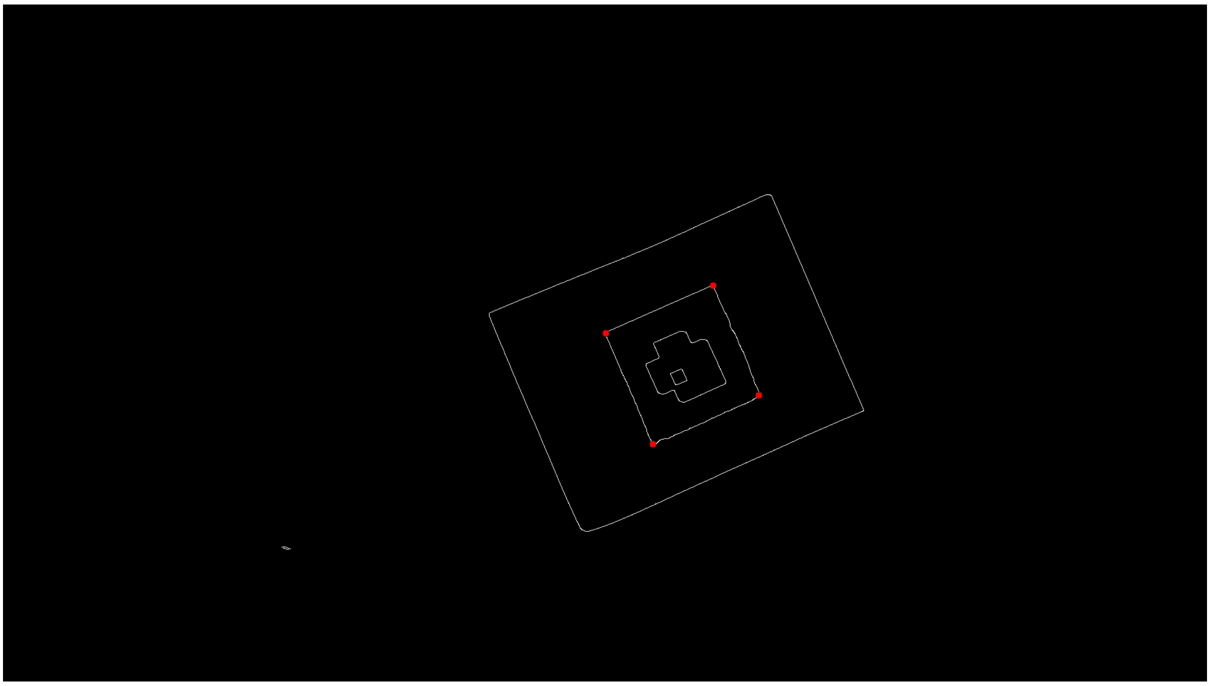


Step 6. Remove the outer vertics by create line equations and find out the corners that are in the lines.



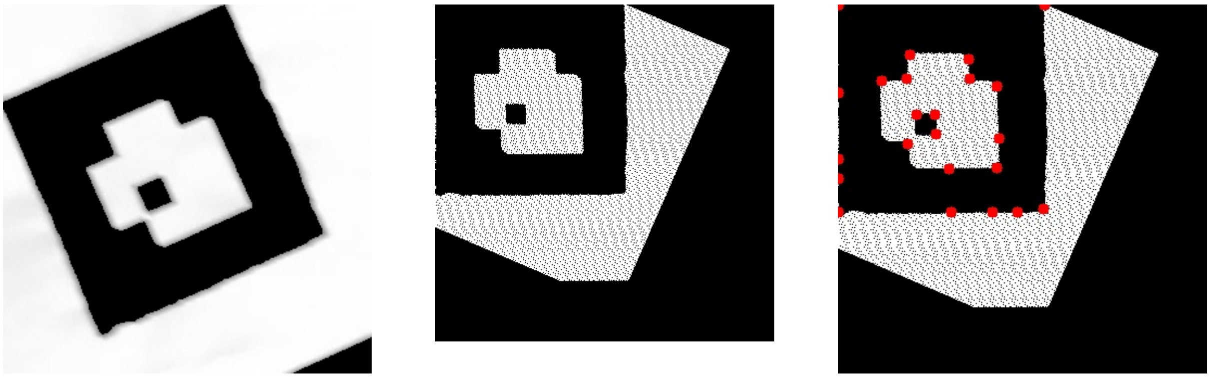
Step 7. Find the four inner vertics using the methond in Step 5, then apply homogrphy transformation to the image within the vertics.

The homograpy can be calculated through finding the right-most vector in V vector produced by SVD.



Apply homogrphy to the AR tag. The left most image shows the original tag and its orientation.

The other two images are images after homography trasformation.



Step 8. Decode the AR tag

Transform the image in to 8x8 array

```
[[0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 0. 1. 1. 0. 0.]
 [0. 0. 0. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
```

The index for identifying the orientation of the code is (2, 2), (2, 5), (5, 2), (5, 5).

One of them should be one and represent the bottom-right corner of the picture.

The four inner codes is the binary representation of the tag's ID.

Tag = 7 in this case.

Videos

https://drive.google.com/file/d/15XaZBMkJdooOftZrxs65q8CqtDfESq7g/view?usp=sharing

Issues

The method I used to find the corners are not very stable, the detections in many frames in the video are not very accurate. And because I only get the corner points from the top, left, bottom, right - most points, if any of the points is misdetect, the homogrphy matrix will not be correct.

Moreover, this detection methonds is a bit slow. Therefore, I did the testing on part of the video frames, and not using the video stream directly.

Problem 2 - Tracking

2.a) Superimposing image onto Tag

Step 1. Calculate the homograph as problem 1

Step 2. Calculate the inverse homograph matrix and use it to find the cooreponding position of the testudo image on the original tag image.

Videos

https://drive.google.com/file/d/1-UhmaP3z1butZm8x8zR3RWRhiKSbOW8V/view?usp=sharing

Issues

Because this section is using the same corner detecting pipeline, the issue in previous section remains, including slow run time and high-failure rate.