Project 1

PERCEPTION FOR AUTONOMOUS ROBOTS ENPM 673 SPRING 2021

UNIVERSITY OF MARYLAND, COLLEGE PARK

Siddharth Telang
stelang@umd.edu
116764520

Problem 1

Detection of APRIL TAG using Fourier Transforms

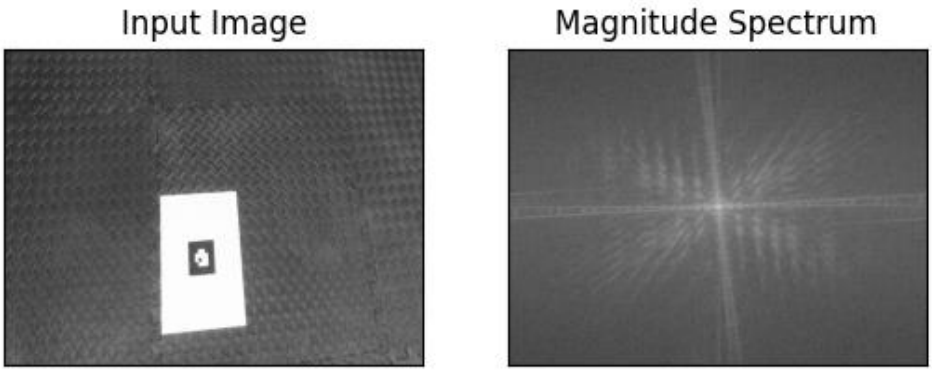
Detection

In this problem, our task is to detect the April Tag in one frame of the video provided. For this, we use Fourier transform and inverse Fourier transform to get the region of interest.

Fourier transform shifts the current domain to frequency domain, which helps us to check the frequency of a particular item. A high frequency indicates more repetitions of the same thing – in our case – pixels, and a low would indicate less presence of those pixels.

We apply Fast Fourier Transform to the frame having the AR TAG and shift the zero or less frequency component to the center of the spectrum for better visualization and further operation decisions. Then, we compute its magnitude and plot the same.

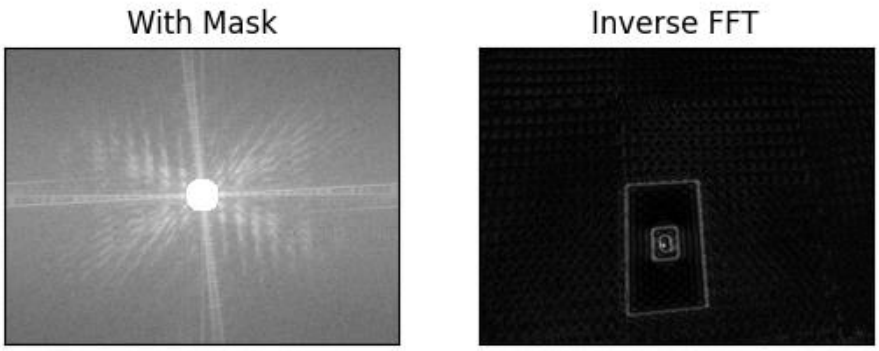
The figure below shows the input image and the magnitude spectrum.



Here, we see that in the magnitude spectrum, the center region denotes lower frequencies, while as we move outwards from the center, the frequency component increases.

Next, we need to perform some operations on the FFT version of the image to get the region of interest. If we use a Low pass filter, it ends up blurring our image and hence the desired region. So, we use a circular High pass filter with a chosen radius and center on the center-shifted FFT image.

As convolution changes to simple multiplication in the frequency domain, we multiply the high pass filter with the shifted version of FFT. The FFT image's center is shifted back to its position and then inverse FFT is applied to this to get the processed image. The figure below shows the magnitude spectrum with HPF mask and inverse FFT.

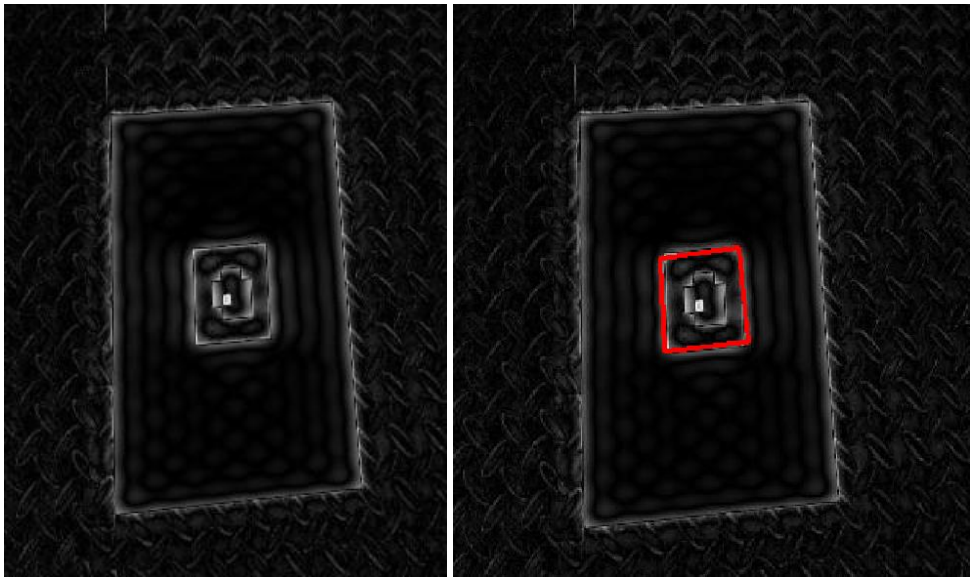


FFT with mask and Inverse FFT

We see that after the inverse FFT, we get these strong edges and corners of the white paper and AR TAG in video frame.

Next, we want to locate the corners of the AR tag. For this, we perform binary thresholding of the grayscale version of the inverse FFT with a selected threshold value.

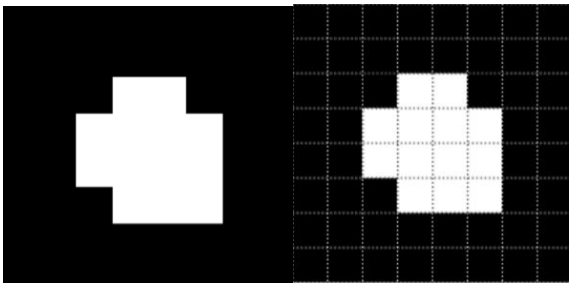
With the usage of methods: "*cv2.findContours*" and "*cv2.approxPloyD*P", we aim to detect the region of interest and get the corner points of the tag. However, the contour function gives us numerous unwanted contours, which we filter out based on the area. Next, the *approxPolyDP* function helps to approximate the corner points, after which we draw the final contour over the tag.



The figure on the right shows the detected AR tag from the left image.

Decoding

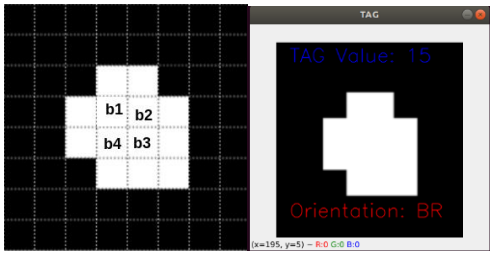
After we successfully extract the AR Tag, the next step is to decode its contents. For this, we are given a reference AR TAG shown in the images below.



AR TAG and grid patterns drawn on it.

The AR tag depicts the orientation with the white corner inside the padded black boundary. In the above image, the orientation is Bottom right, which is importantly the upright position. With the change in camera position, this white corner would change – either of top right, top left and bottom left. Knowing this orientation, we can figure out the translation of the camera. In addition, if we want to superimpose any image on top of this tag, this orientation is important to rotate the image to be placed on top of tag (top right: -90°, bottom left: +90°, top left: 180°)

The tag innermost 2x2 region encodes the binary information -



In the current orientation (bottom right), the innermost 2x2 top left corner is the least significant binary bit and moves clockwise from there to bottom left corner, which is the most significant bit. In the above figure, the binary is: b4b3b2b1

If the orientation were to change to any of the others, the position of b1,b2,b3,b4 would change accordingly.

To get these values, we first divide the marker into 8 parts, then slice the array for all outer corners inside the black padding to get the orientation. Then, we calculate the "*median*" of the sliced array. To get the binary, we slice the innermost 2x2 region and calculate the median. Median does a very good job, especially when there is distortion in shape due to noise or movement of image. However, in some cases, more than one corner of the inner 4x4 report high value (255), hence, we cannot detect the exact orientation at that time. We will look at a possible solution to this in the next problem statement.

Problem 2

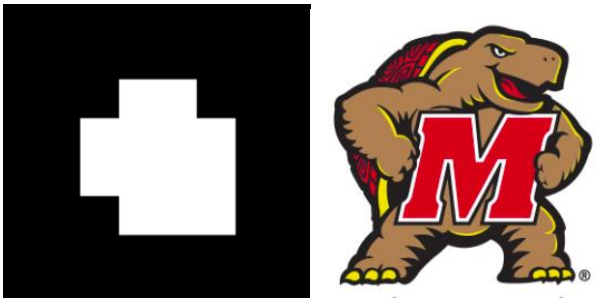
Tracking

Part. (a) Superimpose image onto tag

To superimpose any image on the AR tag, we need the four corners of the tag in the frame. Using this, we will compute the homography matrix which will take us from one plane to another. Then, we will do a forward warping to copy each pixel from source to destination.

If the frame is constant, homography matrix needs to be calculated just once. However, in this case, as there is translation and rotation involved, homography matrix will have to be calculated for each frame, and so would be the superimposing be done.

We saw earlier how the AR tag depicts the or orientation. Using this orientation, we will align our image to be superimposed on the tag for each frame, such that the alignment is as per below – upright position with respect to the white corner,



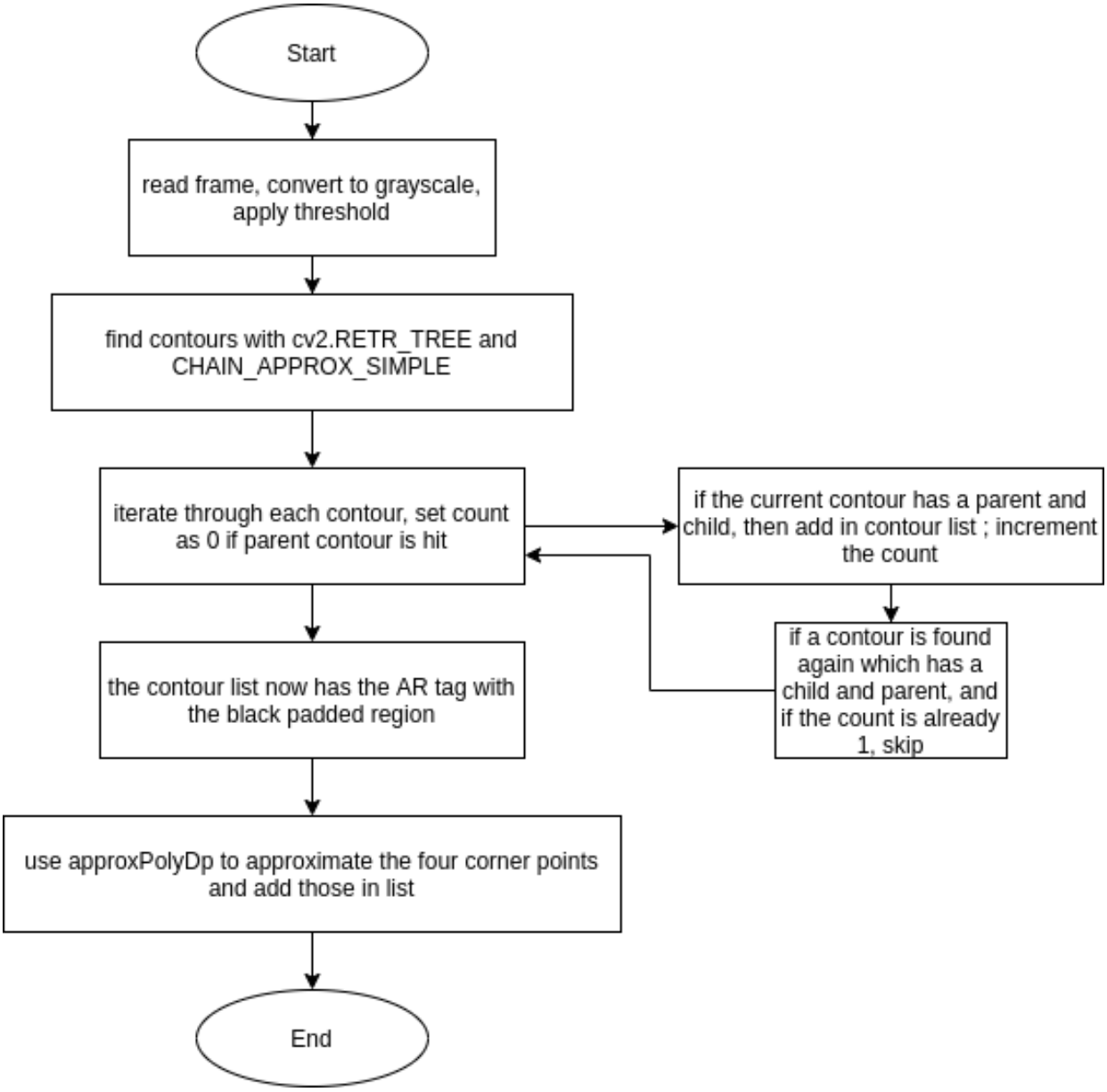
AR TAG and the Image to be superimposed.

This is known as object tracking – we track the object of interest and perform desired operations.

The important step here is to get four corners of the AR tag. The flowchart on the next page describes the process to get the four corners of the tag. When we have finally got the corners, we find the homography matrix and perform warping to extract the tag. The warped size of the tag in the current process is kept as 64x64 for two reasons:

1. Small size corresponds to small noise factor from a large image.
2. The time taken to warp the image is less.

However, while warping the turtle to the AR tag, the size is kept variable (multiple of 8) and for now set as 200x200. This can be increased to maximize resolution, but time would be compromised.



Flowchart describing the corner points detection method used.

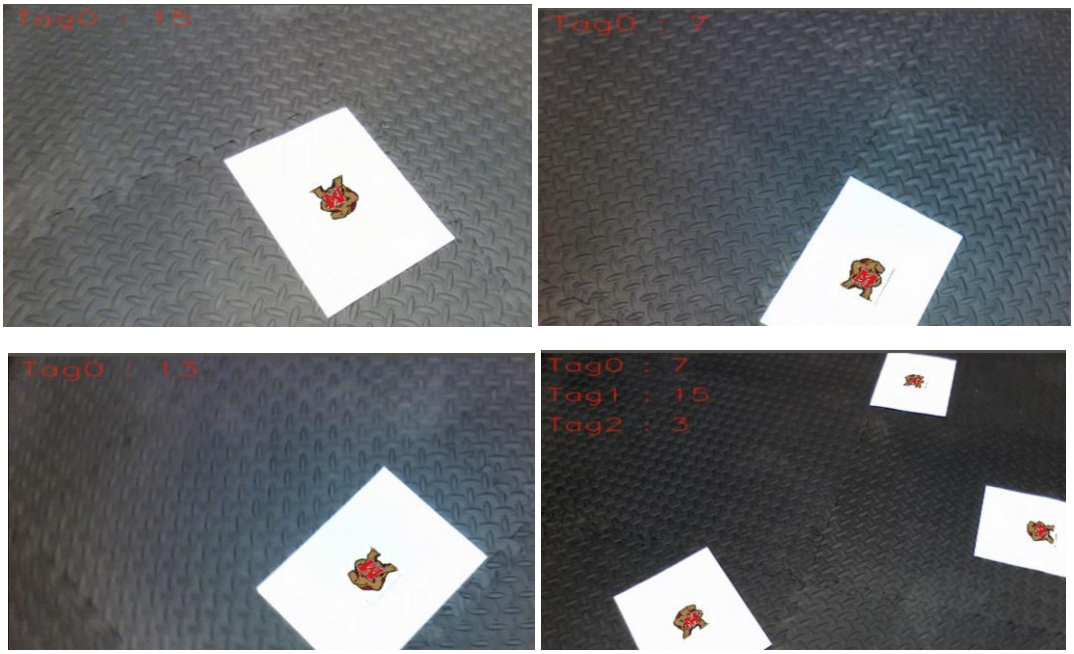
The warped AR tag is decoded then after as per the logic explained in previous section. As the image is not static, we face an issue here. There are a lot of frames, which are distorted due to camera movement (warping makes them even worse), and hence the exact orientation cannot be determined - we get more than one white corner.

A possible solution to this is to check how many corners report 255. If those are more than one, take the last well-known orientation and degree count. If only one corner reports 255, save the good configuration for later use. Then, rotate the turtle.

The above method helps to eliminate the spinning of the turtle when we try to orient it for each frame and the frame is noisy / pose cannot be determined.

This worked very well for the videos with only one tag. However, when it comes to multi tag this fails. There is no "***reliable***" distinguishing factor between the multiple tags. There is binary information available, but due to camera position and orientation, that information is not stable. Hence, the logic is not used for multi tags.

Finally, after the turtle's orientation is matched with the tag, we perform warping to copy the turtle to the AR location.



Screenshots from all the videos

The videos can be viewed here

Part. (b) Placing a virtual cube onto Tag

Placing a virtual cube onto the tag involves projecting a 3D shape on 2D plane, which is finding the camera coordinates from world coordinates.

After determining the corner points of the AR tag, and the homography matrix, we need to calculate the camera projection matrix.

P = K [R | t], where K is the camera calibration matrix; R is the rotation vector and t is the translation.

Let A = [r1 r2 t], then $A = \tilde{A}(-1)^{|\tilde{A}|<0}$ (A is always positive)
$\tilde{A} = \lambda K^{-1}H$
and $\lambda = 2 / ((\text{norm}(K^{-1} h_1) + (\text{norm}(K^{-1} h_2))$;

h_1 and h_2 are column vectors of homography matrix H

Thus, we can find A and from A, we can find r1, r2, t -> along the column. Also, r3 = r1xr2

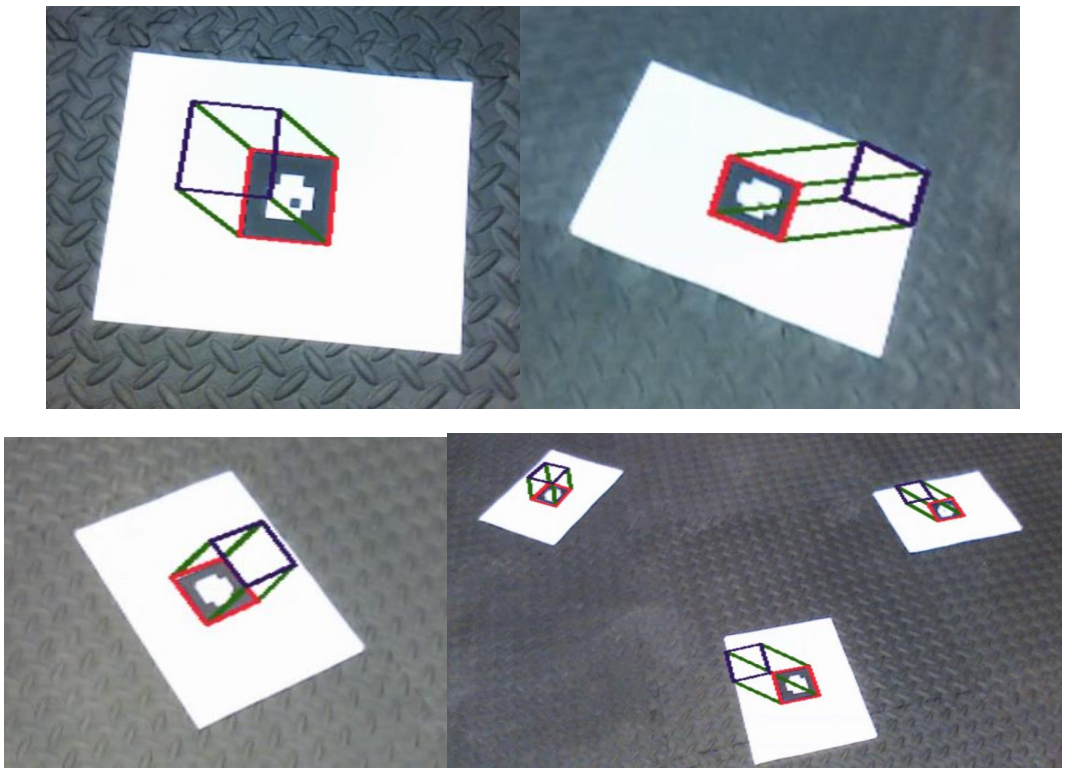
This gives us all the elements of the Rotation matrix: [R|t], which when multiplied with K, gives us the projection matrix P.

After computing Projection matrix P, we can project any point from the world coordinates to camera coordinates: $\mathbf{x^c = Px^w}$

As $\mathbf{x^c}$ is in homogenous coordinates, the last element needs to be 1. Thus, we divide the point by the last element, which is z.

For a cube, we define eight set of planar points and calculate the homogeneous image coordinates using projection matrix.

Then, to draw the cube, we use cv2.line to connect these points, which gives us a 3D image on top of the AR Tag.



Screenshots for 3D cube over AR Tag from all video files

The videos can be viewed here

Problems faced and solutions implemented

1) Corner detection:

I have used contours to detect the corners in this project. I tried Harris corners, but that did not work very well compared to contours. Contours in a noisy environment gives a vague hierarchy and I was not able to detect the AR tag contour using the parent child logic (the AR tag has one parent and one child). I added conditions to reset a counter whenever a parent contour is hit (hierarchy's first or second element is –1) and increment the counter as soon as the first contour inside the main parent is found – this would be the area of interest (contour having one parent and one child – the AR Tag's black boundary.

We have to skip detection of the inner contour of AR tag and hence, when the count is already incremented, we skip that inner contours of AR Tag. By this, I was able to get perfect contour on AR Tag's main boundary in almost all frames.

Despite this screening, there were frames in which the inner AR Tags's contours were getting detected. To overcome this, I increased the epsilon while doing approxPolyDp and rejecting the contour which gave more than four points (the inner contours of AR Tag). Thus, I was able to get four corners of AR Tag in all frames.

2) Tag decoding:

Initially, I was warping the AR Tag to a size of 300x300 and then decoding it, just because I did not want to calculate homography again, as the turtle's size was also 300x300. However, I realized that 300x300 AR tag has a lot of distortions and cannot be decoded properly. Then, I warped it to 64x64 and there was less distortion as compared to 300x300

3) Spinng Turtle:

Due to distortion, there a lot of frames in which you get more than one corner points as "white", meaning you cannot determine the orientation of the tag. If inaccurate orientation is determined, the rotation of the turtle would be improper for every frame and the turtle will rotate in a different direction which looks like its spinning.

To solve this, for each frame while decoding the tag, I checked how many corners are "white". If there is only one, then save the orientation as last good orientation. If there is more than one "white" corner in tag, then refer to the last orientation and rotate the turtle based on last good pose. Using this, the turtle is very stable.

However, this method fails in multiple tag case, because there is no "***reliable***" distinguishing factor between the multiple tags. There is binary information available, but due to camera position and orientation, that information is not stable. Hence, the logic is not used for multi tags.