ENPM-673 PERCEPTION FOR AUTONOMOUS ROBOTS

Project 1
AR Tag Detection and Tracking

Jeet Patel (117430479)

Contents

1. Introduction

This project focuses on detecting and tracking a custom AR Tag which is used in applications of augmented reality for obtaining a point of reference in the real world. There are multiple input video files (Tag0.mp4, Tag1.mp4, Tag2.mp4 and multipleTags.mp4). The problem statement is to superimpose the image of Testudo and a 3D cube on the given videos.

2. Detection

2.1 Filtering

For identifying AR codes certain pre-filtering process is required. There are as follows:

- Converting the image into greyscale color space
- Applying binary thresholding to remove the unwanted image elements which help to detect the tag clearly.
- Using Fast Fourier Transform to detect the edges.

2.1.1 Applying Fast Fourier transform

Fourier transform in image processing is used to convert the spatial domain of input image into frequency domain. After applying the Fast Fourier transform to the image, high frequency and low frequency is separated which is clearly seen in plot below.

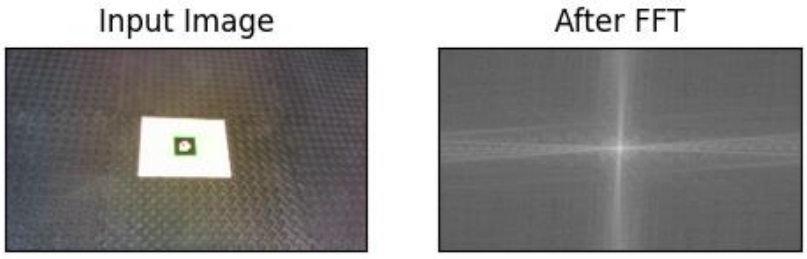

Figure 1: Input image after applying Fourier transform

This FFT transformed image can be used to detect edges by applying high pass filter which is useful in detecting contours instead of applying “imgcanny” function.

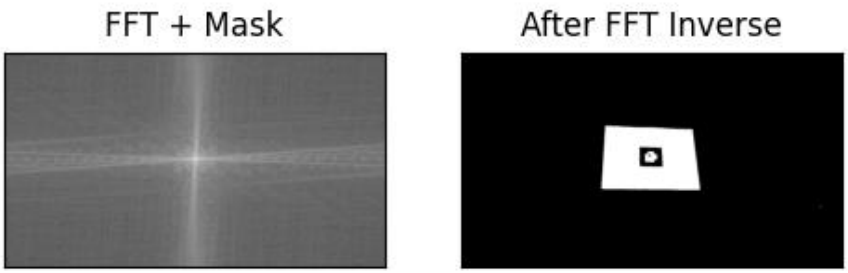

Figure 2: Edge detection after applying High pass filter

2.2 CONTOUR DETECTION

The contours are a useful tool for shape analysis and object detection and recognition. On the transformed image cv2.findContours is used for detecting the contour of the AR tag. Here the retrieval mode selected as RETR_TREE which gives the information of hierarchy of detected contours. This hierarchy provides the parent-child relationships of all contours. The required contour is selected w.r.t hierarchy, areas and numbers of corners.

After separating the contours, cv2.approxPolyDP is used for finding corners of the quadrilateral shape. All the corners are stored in python list called corners; each element of the corner provides the list of four coordinates of the corner.

By setting a single frame in the video, contour is drawn using bounding boxes which is shown in the figure below.

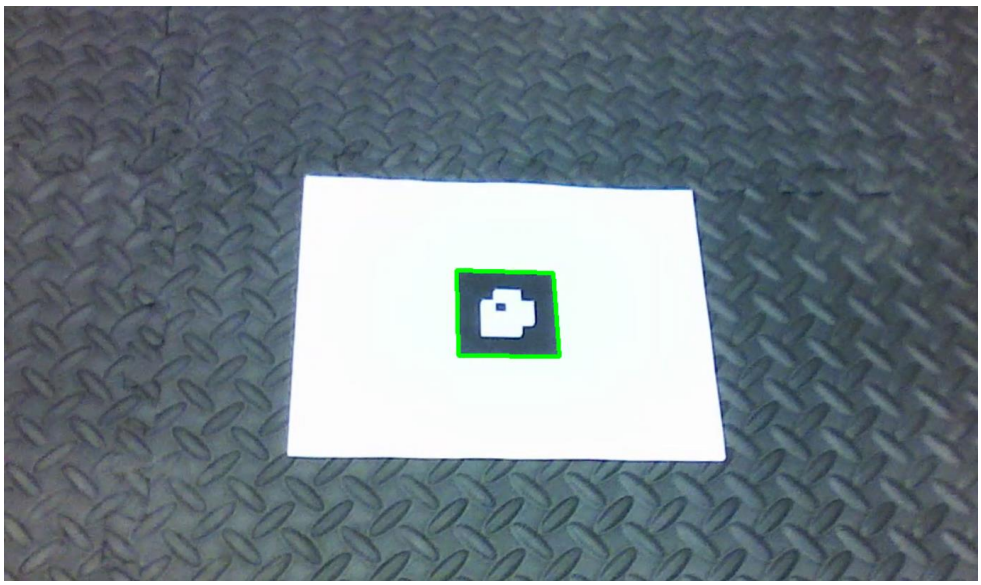

Figure-3: Detected Contour

2.3 Decoding the tag

- To decode the tag, the square image is converted in greyscale image and applying the binary thresholding.
- The tag is divided into 8x8 region, but our region of interest is 4x4 matrix which reveals the identity and the orientation of tag.
- As it in binary form, black is considered as zero and white is considered as 1.
- If white block is present at the bottom left corner, then the orientation of the tag will be bottom left.
- The innermost 2x2 matrix can be decoded based on least significant and most significant bit. The orientation of tag will be help for superimposing the testudo image on the tag.

3. Superimposing an Image on AR tag

The aim here is to superimpose the template image(testudo.png) provided onto the detected AR tag.

- Firstly, the image is resized to the dimension of 200 x 200 to match the world coordinates.
- Then, reorientation of the image is done using the function “changeOrient” before adding the image to existing tag. This function will rotate the image according to the orientation of the tag.
- After doing transformations and applying inverse homography on the rotated image, the area of tag is warped using the image of testudo instead of tag, while background remains black.
- Also, the function “imposetestudo” will generate the new frame using homography which provides us the portion of the tag filled with black and background as original image as shown in figure.

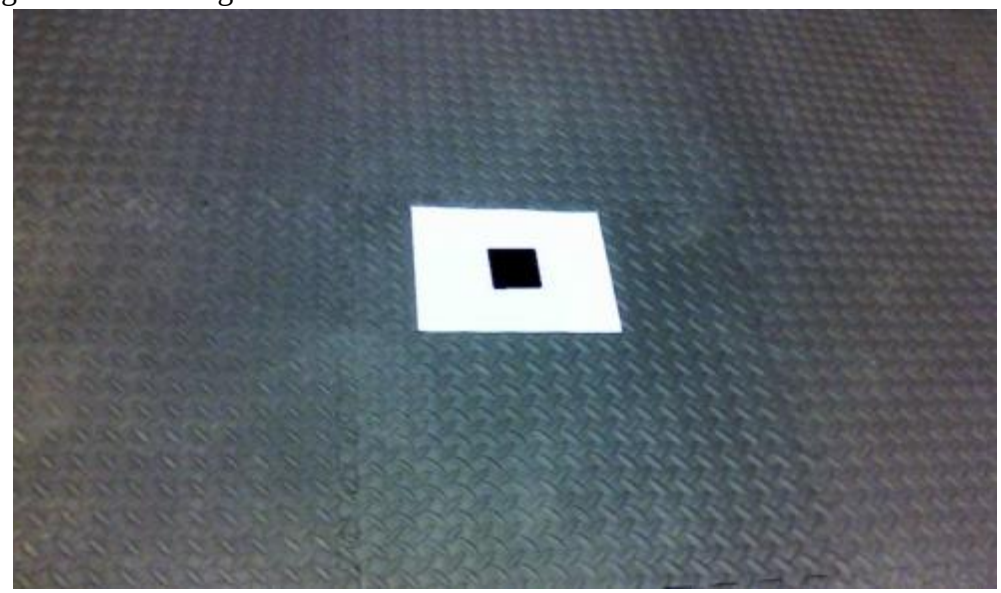

Figure 5: Tag replaced with black patch

- Finally, adding a new image on the existing frame with black region with the function cv2.bitwise_or() will superimpose testudo on AR tag.

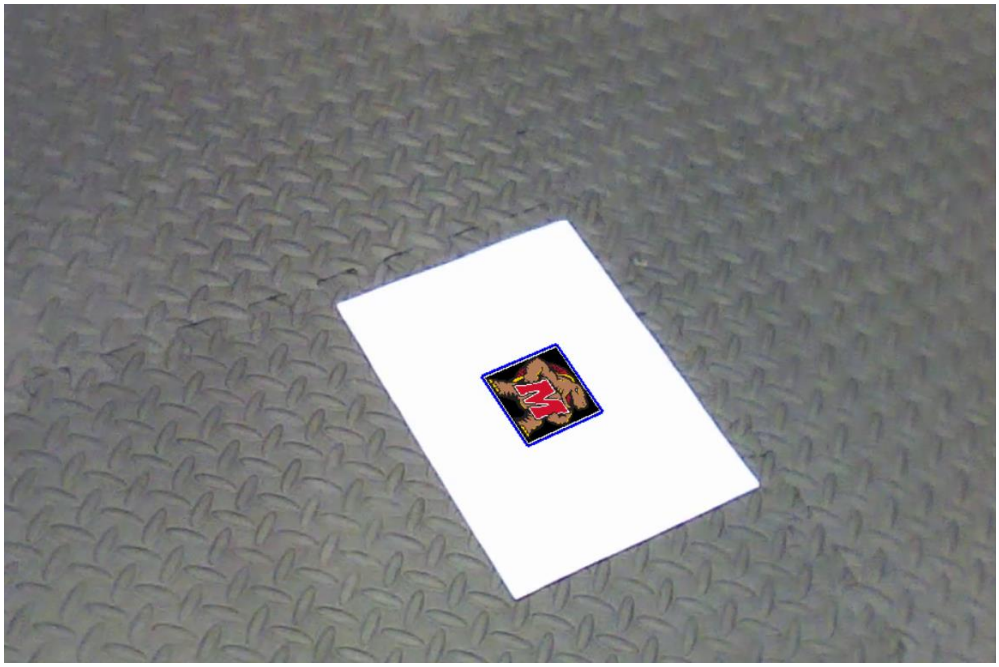

Figure 6: Superimposed testudo image

4. Superimposing a 3D cube on AR tag

- Firstly, conversion of coordinates of cube from camera frame to world frame is required which is done using homography for obtaining square shaped tags.
- To create a 3D cube, projection of coordinates from world frame to camera frame is done using projection matrix.

$$\mathbf{X}^{(c)} = \mathbf{P}\,\mathbf{X}^{(W)}$$

Where, $\mathbf{X}^{(c)}$ = points in image frame

Projection matrix $\mathbf{P}$ =[R|t] = [r1, r2, r3, t], R= rotational matrix, t = translational vector

$\mathbf{X}^{(W)}$ = points in world frame

- The corners of the cube are computed using the function “cubePts” with the help of projection and homography matrix.
- A cube is drawn using above obtained points and corners with the help of “drawCube” function.

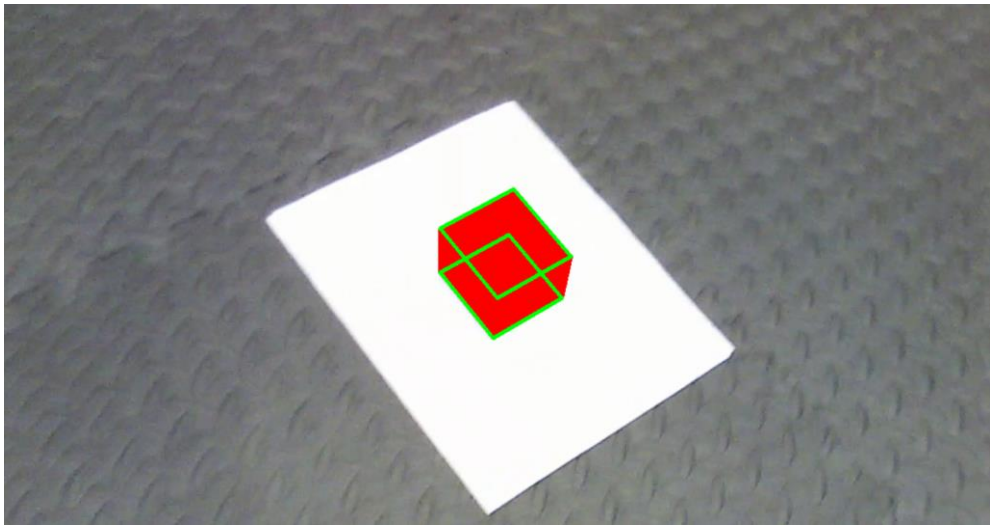

Figure-6: A 3D Cube superimposed on AR tag

5. Challenges faced during implementation

- During contour detection, after inverse FFT, there was problem getting the frame back and finding contour using inbuilt function. Also, there were wrong contours detected. So, applying thresholding and excluding contours with help of hierarchy and studying various retrieval methods help to achieve the desired contour.

6. Video's link

- https://drive.google.com/drive/folders/1H7pNkjezM8P_5nDfCiT1A8DweqveT4E7?usp=sharing

7. References

- https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html
- https://docs.opencv.org/3.4/d9/d8b/tutorial_py_contours_hierarchy.html
- https://docs.opencv.org/2.4/doc/tutorials/core/adding_images/adding_images.html
- https://stackoverflow.com/questions/22180923/how-to-place-object-in-video-with-opencv