ENPM673 – Perception for Autonomous Robots – Spring 2022

Report for Project 1

By

Prateek Verma (118435039)

Under the guidance of -

Samer Charifa, Gokul Hari, Sakshi Kakde, Jayesh Jayashankar

Introduction:

This project will focus on detecting a custom AR Tag (a form of fiducial marker), that is used for obtaining a point of reference in the real world, such as in augmented reality applications. There are two aspects to using an AR Tag, namely detection and tracking, both of which will be implemented in this project. The detection stage will involve finding the AR Tag from a given image sequence while the tracking stage will involve keeping the tag in "view" throughout the sequence and performing image processing operations based on the tag's orientation and position (a.k.a. the pose).

Data :

A reference video is given to test your code against. The intrinsic camera parameters used in this project for the Intrinsic matrix which is used to transform 3D camera coordinates to 2D homogeneous image coordinates are given.

Problem 1 – Detection :

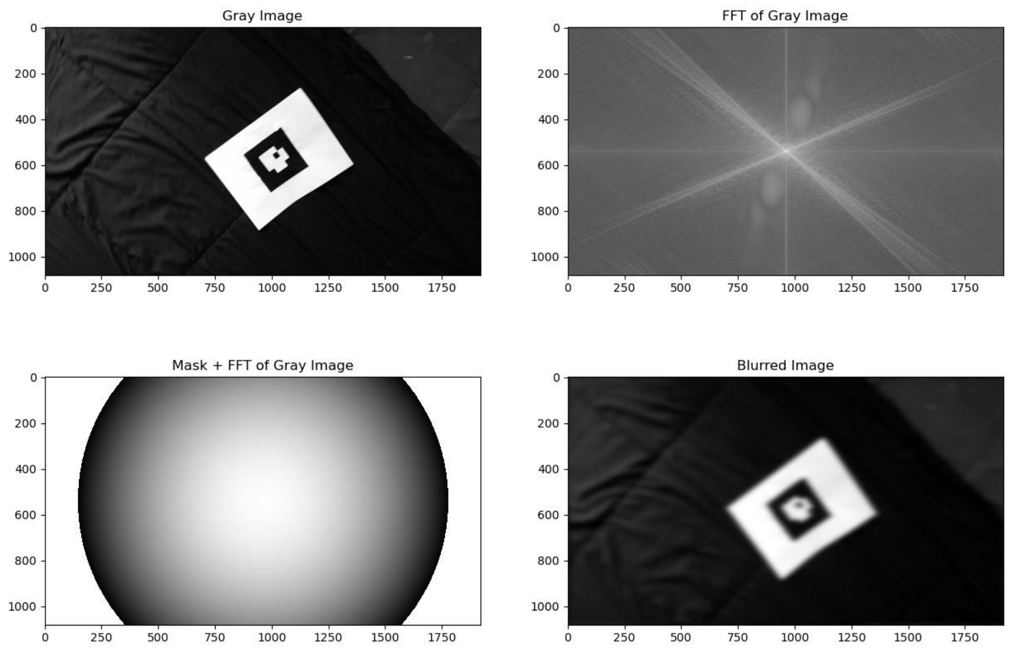
Problem 1.a) AR Code detection:

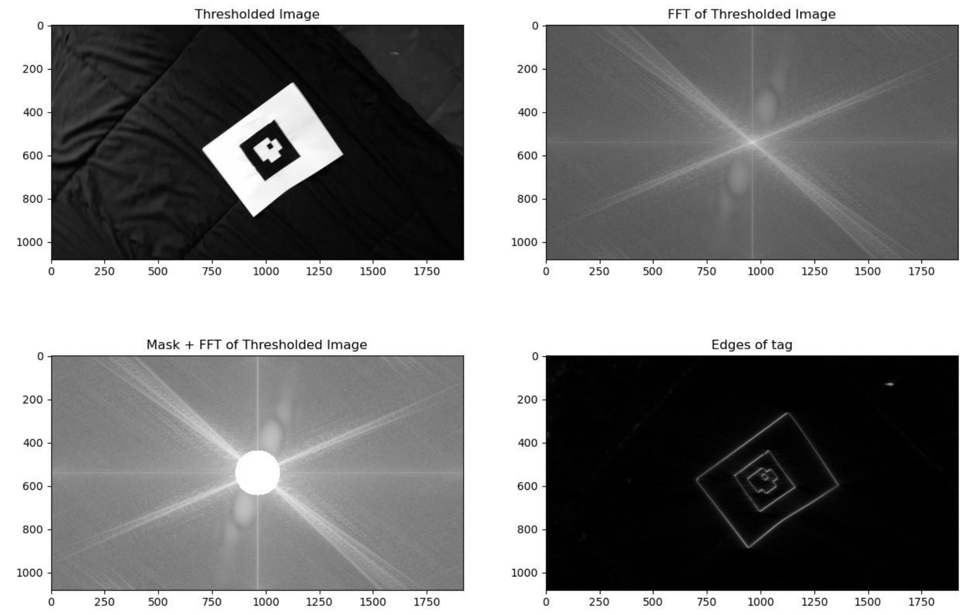
The task here is to detect the April Tag in any frame of Tag1 video (just one frame). The steps for the same are as follows:

1. Read the video file frame by frame using the cv2.VideoCapture() function.
2. One frame is chosen from the video to detect the AR Code.
3. The image frame is converted to grayscale then blurred using Fast-Fourier Transform (FFT) and masked using Gaussian mask.
4. Then the image is masked with Circular Mask and Inverse FFT is performed to find the edges of the AR Tag.

Program File: **Prob1A.py**

Final Output:





—---

Problem 1.b) Decode custom AR-tag:

A custom AR Tag image, as shown in Fig. 1, is given for reference. This tag encodes both the orientation as well as the ID of the tag.

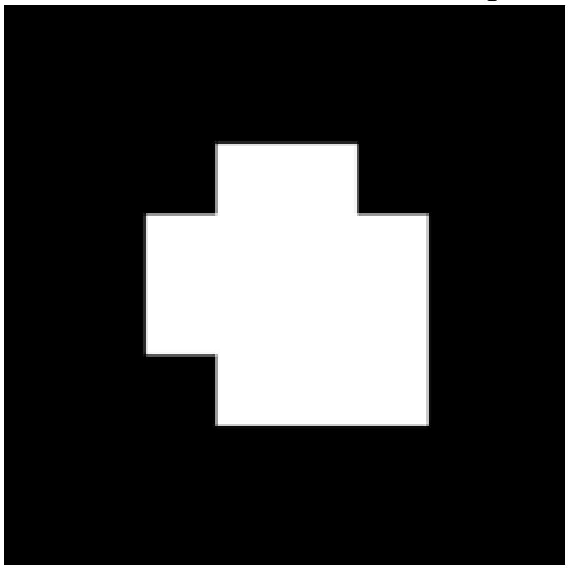


Figure 1: Reference AR Tag to be detected and tracked

Encoding Scheme:

To properly use the tag, it is necessary to understand how the data is encoded in the tag. Consider the marker shown in Fig. 2:

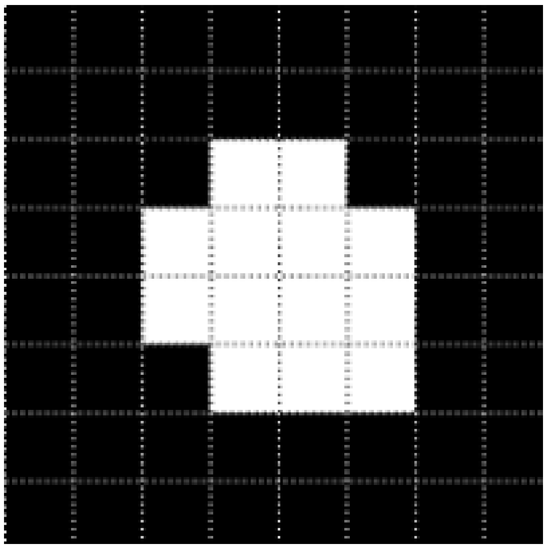
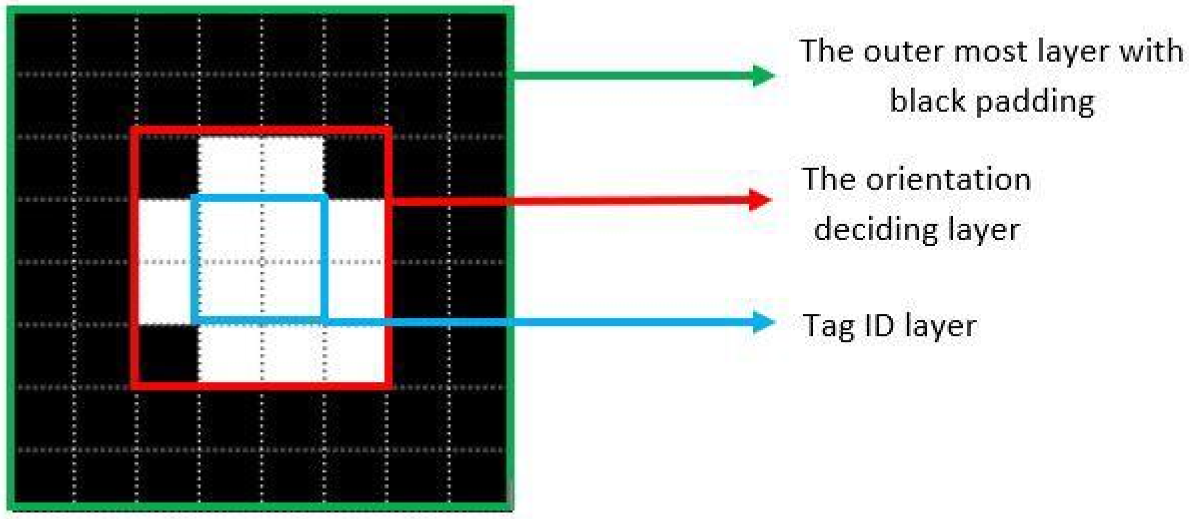


Figure 2: Grid pattern overlayed on reference AR Tag

- The tag can be decomposed into an 8 x 8 grid of squares, which includes adding 2 squares width (outer black cells in the image) along the borders. This allows easy detection of the tag when placed on any contrasting background.
- The inner 4 x 4 grid (i.e. after removing the padding) has the orientation depicted by a white square in the lower-right corner. This represents the upright position of the tag.
- Lastly, the inner-most 2 x 2 grid (i.e. after removing the padding and the orientation grids) encodes the binary representation of the tag's ID, which is ordered in the clockwise direction from least significant bit to most significant. So, the top-left square is the least significant bit, and the bottom-left square is the most significant bit.


The outer most layer with black padding
The orientation deciding layer
Tag ID layer

There are two files for this implementation:

Prob1B_video.py
Prob1B_tag.py

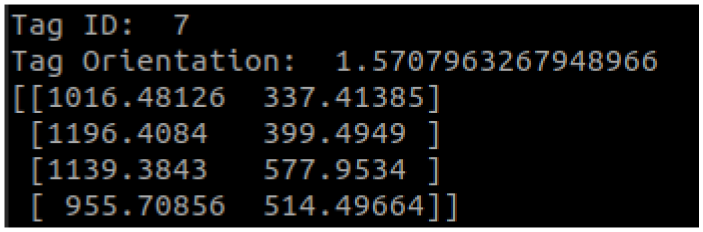
The steps Prob1B_tag.py are as follows (Decode Reference AR Tag):

1. To perform the encoding, the reference marker was split into equal space 8x8 grid.
2. Crop and remove black paddings in the tag.
3. Now inner 4x4 grid is plotted on the reference tag.
4. The tag is then rotated to the reference position to consider the inner 2x2 grid for decoding.
5. The decoded output is 1111 which in decimal terms is equal to [15.0].

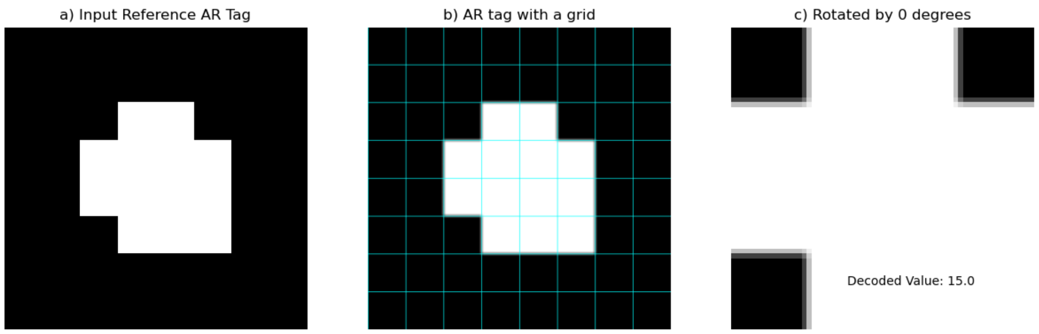
The steps for Prob1B_video.py are as follows (Decode AR Tag from video):

1. Detect corner edges of the AR Tag using different methods. At first, the cv2.findContours() function was used but later after seeing the discussion forum on Canvas, I had to use the Harris Corner Detection method to find the corners.
2. Once the four corners are found, the corners are sorted as top-left, bottom-left, top-right, and bottom-right using homography and warping techniques.
3. Once the tag is detected we use the same steps as above to extract information.

Output:



```
Tag ID:  7
Tag Orientation:  1.5707963267948966
[[1016.48126  337.41385]
 [1196.4084   399.4949 ]
 [1139.3843   577.9534 ]
 [ 955.70856  514.49664]]
```



Problem 2 – Tracking

Problem 2. a) Superimposing image onto Tag:

Once we have acquired the four corners of the tag, we use homography estimation on this in order to perform some image processing operations, such as superimposing an image over the tag. The image you will use is the testudo.png file provided, see Fig. 3. Let us call this the template image.

Figure 3: testudo.png image used as a template

There are two ways for this implementation:

1. The first one is using cv2.findContour() function.
2. The second one uses Harris Corner Detection Method. This is because later it was discovered through the discussion forum that the use of the cv2.findContour() function is not allowed.

Prob2A_Testudo.py
Prob2A_Testudo_harris.py

Steps to solve:

1. The first step is to compute the homography between the corners of the template and the four corners of the tag which are detected using the previous steps.
2. Depending on the orientation acquired we rotate our corner points in that direction in the video. For example, if the white cell is located at the top-left position, then we need a rotation of 180 degrees to make the tag upright.
3. The corners are obtained from testudo.png. Now we compute homography and warping on the two sets of ordered points.

Problems faced:

1. In the second method (using Harris Corner detection): At multiple instances, the frames were a bit distorted and hence corners were not detected properly.
2. At times, there was no detection of the tag as the warped images were

too distorted to process.
3. The Orientation of the superimposed image differed.

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

Augmented reality applications generally place 3D objects onto the real world, which maintain the three-dimensional properties such as rotation and scaling as you move around the "object" with your camera. In this part of the project, we will implement a simple version of the same, by placing a 3D cube on the tag. This is the process of "projecting" a 3D shape onto a 2D image.

The "cube" is a simple structure made up of 8 points and lines joining them.

There are two ways for this implementation:

3. The first one is using cv2.findContour() function.
4. The second one uses Harris Corner Detection Method. This is because later it was discovered through the discussion forum that the use of the cv2.findContour() function is not allowed.

Prob2B_Cube.py
Prob2B_Cube_harris.py

The steps are as follows:

1. Compute the homography between the world coordinates (the reference AR tag) and the image plane (the tag in the image sequence).
2. Build the projection matrix from the camera calibration matrix provided and the homography matrix assuming that the points are planar, i.e. z=0 for all the points.
3. Assuming that the virtual cube is sitting on the top of the marker, and that the Z-axis is negative in the upwards direction, we are able to obtain the coordinates of the other four corners of the cube.
4. This allows us to now transform all the corners of the cube onto the image plane using the projection matrix.

—---

Google Drive Link for Final Output files:

https://drive.google.com/drive/folders/13CmGxQx2gAwObgxnjXrI3An KQn57v5ls?usp=sharing