ENPM 673: Perception for Autonomous Robots

Spring 2022

Project 1: Detecting and Tracking Image Features using FFT and Homography

Instructor: Dr Samer Charifa

Teaching Assistants: Gokul Hari, Sakshi Kakdi

Grader: Jayes Jayashankar

By: Yoseph Kebede

Problem 1 – Detection

Problem 1.a) AR Code Detection

In order to detect the AR tag from the video using FFT, the below process was followed:

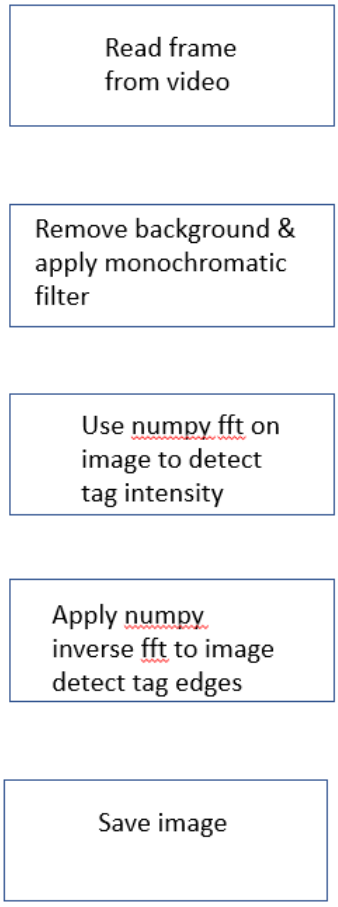
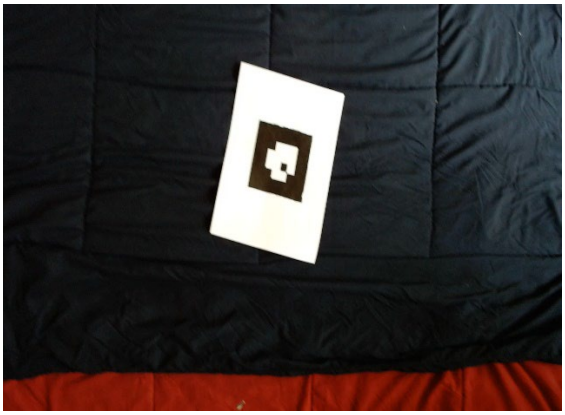
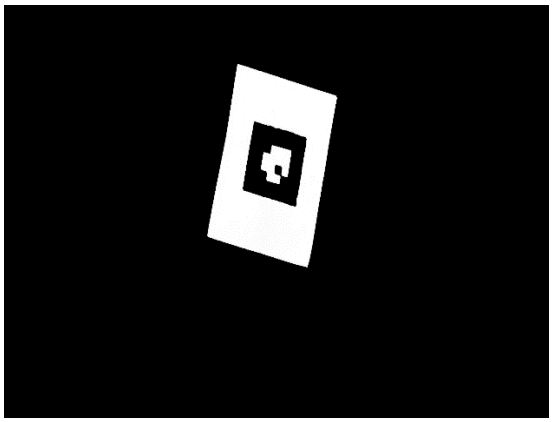


Fig.1 Fast Fourier Transform Flow chart

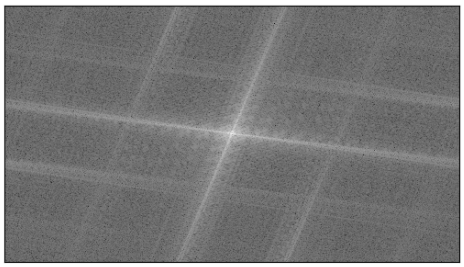
The image transformation looks as shown below following the above steps. The numpy library along with open cv were used to facilitate the image processing.



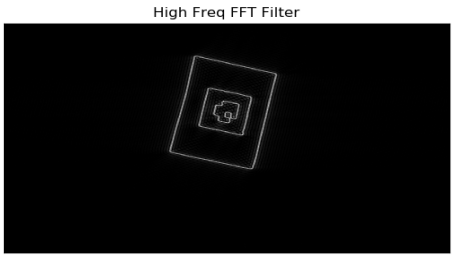
a) Original Image



b) After monochromatic masking



c) FFT of masked image



d) After inverse FFT

Fig2a-d: Application of FFT to detect image ages

Problem 1b.) Decode custom AR tag:

Given a custom AR Tag shown in Figure 3.a, to decode the tag, the orientation as well as the key definitions of the image need be identified. Thus, the first task would be to divide the image into grids following predetermined identifiers.

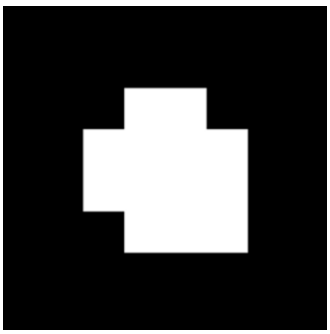


Fig3.a) AR Tag

After computing the image resolution, the tag is then divided into an 8x8 grid, as shown in Figure 3.b.

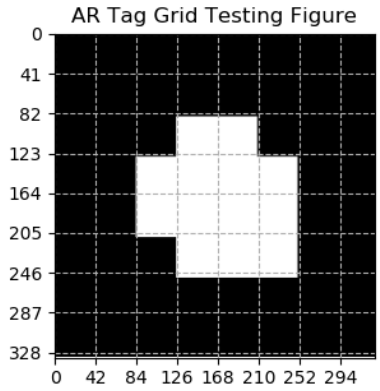


Fig3.b – AR Tag decoding in grids

Then, by following the rubric, it is known that for an image to be the tag:

- It's four outer corners need be black
- Three of the four corner of the white tag should be black, with only the bottom right staying white
- The centermost 4 grids need to be white.

Following this convention, the mean of every pixel in each grid was computed to determine its color. Once the above requirements were met in such manner, the orientation of the tag is checked by referencing upright orientation with respect to the bottom right grid of tag staying white. If not upright, degree of rotation needed to keep tag upright gets recorded. Finally, the innermost four grids are assigned a binary value of '0' or '1', with top left being least and bottom left being most significant bits where order is in clockwise direction. Thus, the bits are arranged accordingly, ie $(\text{MSB } \_ \_ \text{ LSB})_2$, and then converted to decimal yielding the tag identification number (Tag Id). Process flow is shown in Figure 4 as follows.

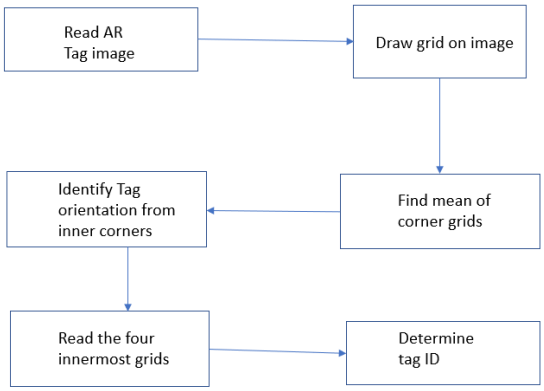


Fig4 – Decoding AR Tag

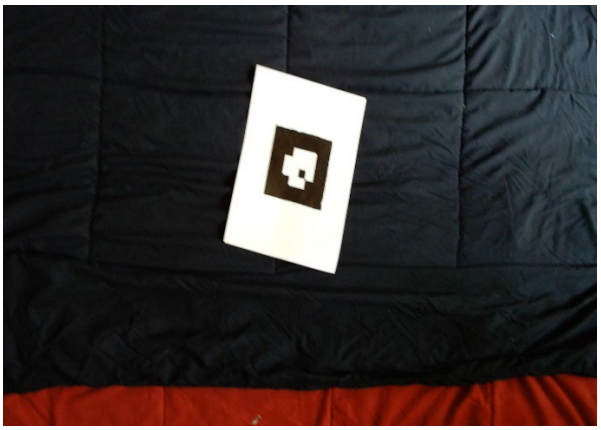
The pseudocode used to execute this task is:

- Read image from path
- Mask image monochromatically (first to grayscale then to black and white)
- Divide image in grid of 8X8
- Record pixel values, partition matrix into the grids
- Take mean of each grid and save
- Determine values of 4 outermost grids
- Determine values of 4 inner grids,
- assign rotation if bottom right is not white
- Determine tag ID from inner most four grids
- Return tag ID and rotation (if needed)
- Terminate program

Problem 2 – Tracking

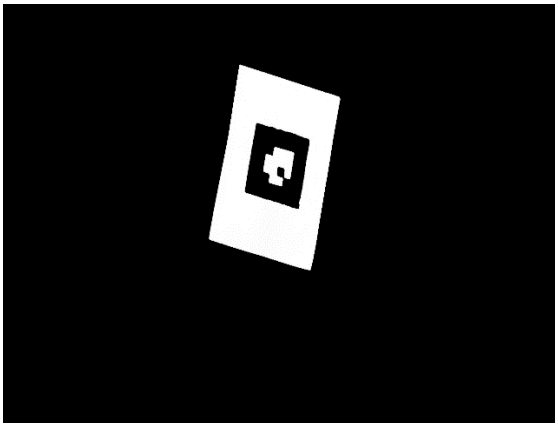
Problem 2.a) Superimposing image onto Tag

This task involved first by reading a video frame by frame, which is followed by identifying the corners of the tag in each frame. Next, the homography between the corners of the AR Tag and the corners of the image intended to get projected was calculated (testudo logo used here). Moving forward, find the homography of the AR tag to the planar image to identify the orientation of the tag after which the amount of rotation (ie if tag is warped as different from upright) was applied to the testudo image. Finally, the testudo image is warped inversely into the tag image by replacing the tag itself. Therefore, by applying such process to each frame and storing the transformation, all of the new frames are stitched together thereby generating a new video which is seen to have the Tag replaced by the testudo logo with the correct orientation. The correct execution of one iteration is shown in Figure 5.



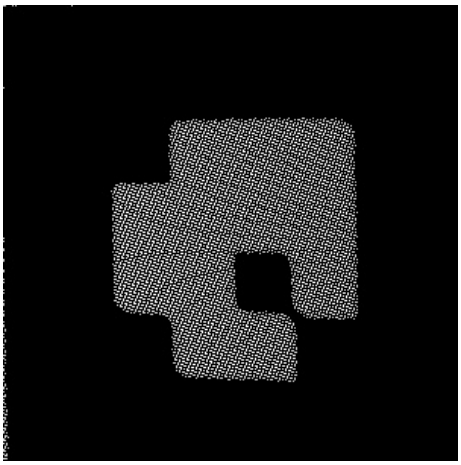
a) Original Image Frame From Video

b) Image to be projected



c) Grayscale + monochromatic masking

d) Four corners of the tag identified



e) AR Tag warped

f) Orientation adjusted following AR Tag

g) Testudo superimposed on AR Tag image

Fig 5 – Superimposition process of testudo on AR Tag per frame

Therefore, in brief, the pseudocode used to execute this task is:

- Read one image frame from video
- Mask image monochromatically (first to grayscale then to black and white)
- Use opencv's harris corner method to identify corners
- Systematically minimize points to 4 corners of tag
- Calculate homography between 4 corners of tag and testudo image
- Warp AR Tag using homography and identify orientation
- Rotate (if needed) testudo logo to match warped image
- Apply inverse warping on modified testudo using inverse homography
- Store new superimposed image in a list
- Repeat the above steps for every frame in video
- Stitch all frames into one video
- Return new video
- Terminate program

Challenges: Finding the desired 4 corners of the AR Tag was incredibly difficult after successfully operating on the first frame. This was due to the expectation that Harris Corner detection would work evenly for any image frame passed on to it. While it discovers many frame on some, it doesn't pinpoint the desired corners on most and hence the filtering mechanism employed after the identified corner points ends up removing most of the needed points. Furthermore, the threshold used to filter the corners changes as the video rolled on making the point identification fed into the homography function inaccurate, thereby yielding wrong superimposed outcomes.

Problem 2.b) Placing a virtual cube onto Tag

This task was accomplished by creating 8 points that makeup a cube in the real world. Then, the bottom and top plane were used as a reference to find the projection of the cube's corners onto the AR Tag image via the projection matrix. Once these 8 points are transposed, a line is plotted connecting the corner points from the top plane of the cube with the bottom.