Project 1
ENPM673

Sakshi Kakde
M. Eng. Robotics
University of Maryland
College Park, MD, 20742
Email: sakshi@umd.edu

I. Problem 1

A. AR Code detection

The edges of an image are high frequency signals. If we could filter out the low frequency signals from an image, we can obtain the edges. However, there is a lot of texture in the background, which makes the detection noisy as shown in figure 1. To make the detection smoother, I tried blurring the

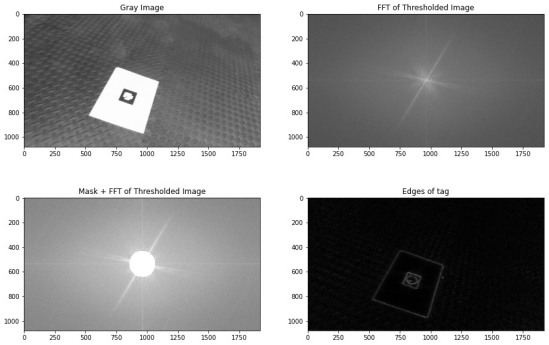
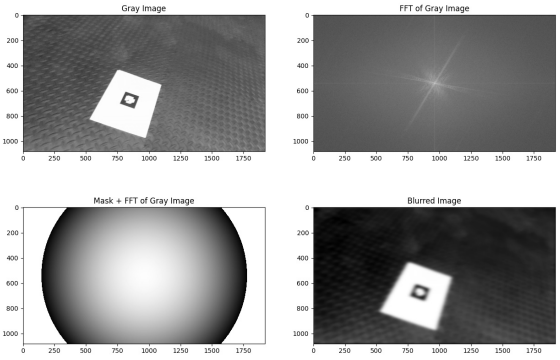


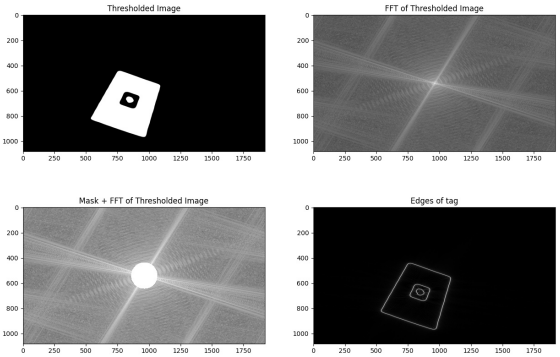
Fig. 1. AR tag detection using FFT(noisy)

image. As discussed in the class, convolution in the spatial domain becomes multiplication in the frequency domain. This means convolution by a Gaussian kernel in a spatial domain will become multiplication by a Gaussian in the frequency domain. I tried using the same idea and the results are demonstrated in figure 2. The steps that I followed are:

1) Compute FFT for a gray-scale image.
2) Define a Gaussian mask and multiply it with the computed FFT. This will filter out the high-frequency signals, thus making the image smooth.
3) Compute the inverse of FFT to get a blurred gray image.
4) Obtain a binary image by thresholding the gray image
5) Compute FFT for this binary image.
6) Define a circular mask and multiply it with the FFT from step 5. to filter out low-frequency signals.
7) Compute the inverse of FFT to get the edges.



(a) Image blurring using FFT



(b) Edge detection using FFT

Fig. 2. AR tag detection using FFT(less noisy)

B. Decode reference AR tag

Steps to decode the AR Tag:

1) Convert the image to binary and resize it to a size of 160×160.
2) Divide this 160×160 into 8×8 grids, i.e. each block in the grids is of size 20×20.
3) Take the median for each block and replace the 20×20

block with a single value depending on the median. It will be either 255 or 0 since the image is binary.

4) Extract the central 4×4 grids which have the rotational and ID information.
5) Check for the values at the four corners. Only one of them should be high. If values at more than one corner are high or no values are high, then AR-tag cannot be detected, and hence shall be discarded.
6) The upright position of the tag is determined by the bottom right grid. Keep rotating the tag till the bottom right grid is high.
7) After the orientation is set, extract the central 2×2 grids for the tag ID.
8) Flatten the 2×2 matrix. The leftmost digit is the least significant bit and the rightmost is the most significant bit. Convert it to decimal to get the tag ID. Please note that in figure 3 last image, the values are in reverse order(left is LSB and right is MSB).
The ID of reference tag is 15.

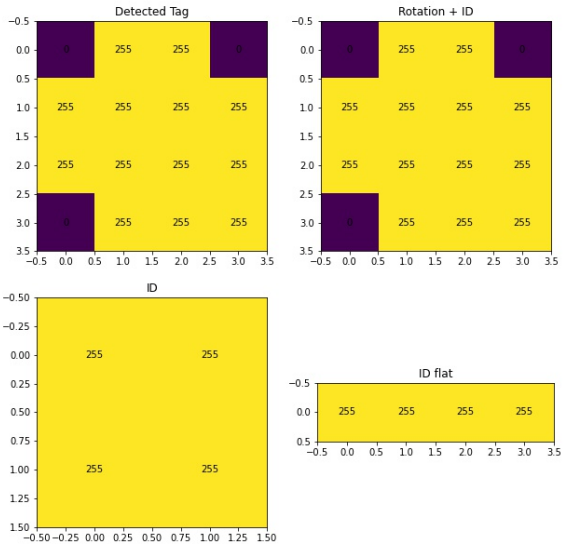


Fig. 3. Decoding AR reference tag

C. Decode AR tag from video

1) Detect Tag: At multiple times in the video, there are a few background objects. To avoid false positives, I am first detecting contours for a tag with white paper and then creating a mask using the detected contours. I multiply the mask with the frame and then use the result to detect the actual tag. The results are shown in figure 4.
For contour detection, I used *cv2.findContours()* function. To make the detection stable, I exploited the *hierarchy* that is a return from the function. For detecting the contours for a white paper, I accepted the contours which have no parent and at least one child, hence detecting only the outer boundary of the tag.
For detecting the actual tag, I again used *cv2.findContours()*, but on the masked image. To make detection stable, I used contour area and contour perimeter as conditions to accept the detected contours. Refer figure 5.

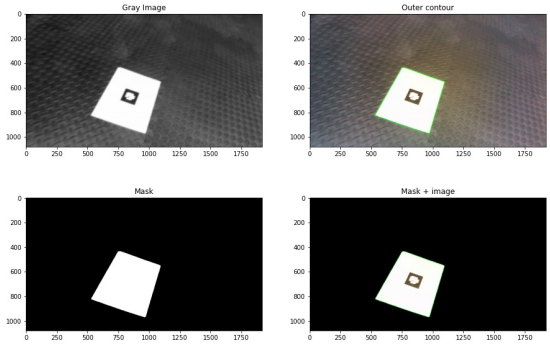


Fig. 4. Outer contours and mask for AR tag

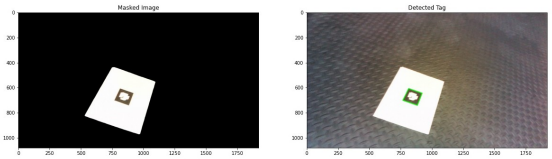


Fig. 5. AR tag detection from masked image

2) Warp tag: Once I get the four corners of the tag, I sorted the corners in order as top-left, bottom-left, bottom-right and top-right. These sorted four corners corresponds to the tag corners as shown in figure 6. Given two sets of points,

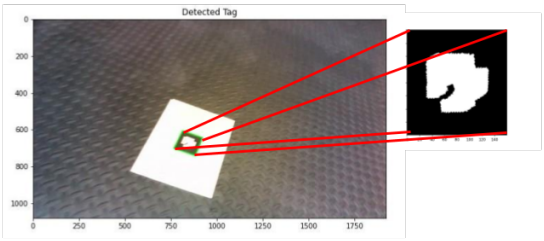
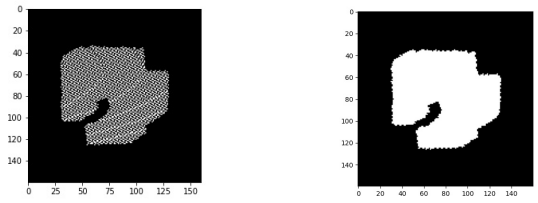


Fig. 6. Tag extraction

a homography matrix can be computed which can be used to warp the image and obtain the AR-tag for decoding. Initially, I tried warping the image using the forward warping technique. However, the results were quite noisy. Later, I shifted to inverse warping as discussed in the supplementary material provided. The result from both approaches is shown in figure 7.



(a) Forward warping (b) Inverse Warping

Fig. 7. Image Warping

3) Decode tag: Once the tag has been obtained, we can use the same process as described in section I-B. Refer figure 8 for the extracted information. Please note that the leftmost bit is LSB and the rightmost bit is MSB in figure 8. So the tag ID will be reverse of [1 0 1 1], which is [1 1 0 1] in binary and 13 in decimal.

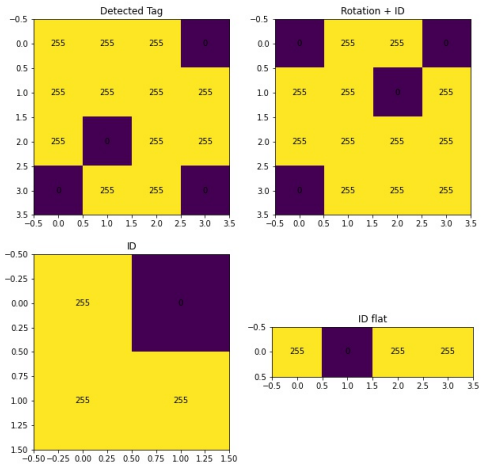


Fig. 8. Decoding AR tag from video

II. Problem 2

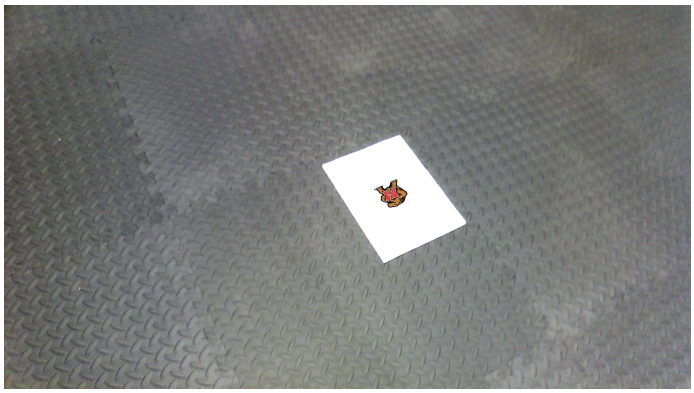
This part of the report discusses, in brief, the projection of an image and a virtual cube on an AR tag. The result videos for this section can be found here

A. Superimposing image onto tag

The first step is to detect the corners of the AR-tag as discussed in section I-C1. Next, we need to make sure that the corners are in the correct order, i.e. top-left, bottom-left, bottom-right, and top-right. We extract the tag from each frame and compute its orientation as discussed in section I-C3. Depending on what the orientation is, we rotate our corner points in that direction. 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. So, we will rotate the corner points twice by 90 degrees.

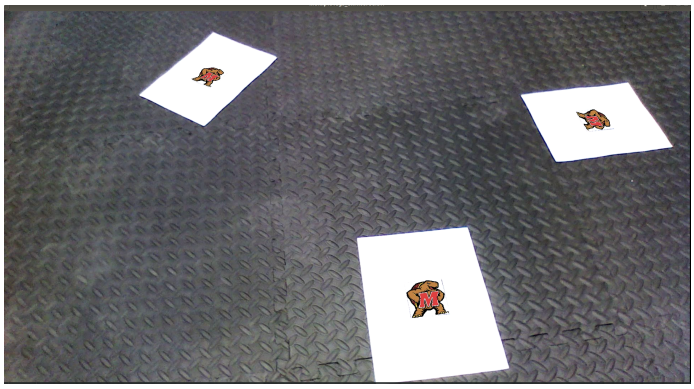
We are trying to superimpose the testudo image on the AR tag. The corners of the testudo image can be easily found. However, the order shall remain the same as top-left, bottom-left, bottom-right, and top-right. Now that we have two sets of ordered points, we can compute a homography matrix and warp the testudo image on the video frame.

In the process, I faced the following problems:



(a)

(b)



(c)

Fig. 9. Testudo superimposed on AR tags

1) At a few instances, the frames were a bit distorted and hence the corners were not detected properly. In such cases, I used the past corner readings. This is not an optimal solution. An ideal solution is to predict the

position of the corners depending on the past values.
2) At times, the warped image was too distorted to process. When there was no detection of the tag, then I used the past orientation of the tag. This enabled me to reduce the frequency of angle flip.
3) Sometimes, the two consecutive angles differed by 180 degrees. In these cases, I used the previous value of the angle.

The frames from video with superimposed testudo are shown in figure 9.

B. Placing a virtual cube onto tag

Since a cube is a three dimensional entity, we need a 3×4 projection matrix to project 3D points into the image plane. The basic assumption while calculating the homography matrix is that the points are planar, i.e. z = 0 for all the points. Now, we have z values as well. Thus we need to modify this H matrix to suit our requirements. To steps to obtain a projection matrix(P) from a homgraphy matrix(H) and camera calibration matrix(K) are as follows:

1) Lets write the homography matrix as $H = K\tilde{B}$.
2) Then, $\tilde{B} = K^{-1}H$
3) We need to obtain matrix B such that $B = \lambda\tilde{B}$
4) Compute mod of $\tilde{B}$. if it is negative, then $\tilde{B} = -1 * \tilde{B}$
5) We will calculate λ as $\lambda = \frac{2}{\|b1\|+\|b2\|}$, where b1 and b2 are column vectors of $\tilde{B}$
6) Therefore, $B = [\lambda b1, \lambda b2, \lambda b3]$
7) B can be decomposed and written as $[r1, r2, t]$, where *r1* and *r2* are rotational components and *t* is translational component.
8) We know that the rotation matrix is orthonormal; the last column must be perpendicular to the first two[2]. Hence, $r3 = r1 \times r2$
9) The final projection matrix can be written as $P = K[r1, r2, r3, t]$.

Once we obtain the projection matrix, we can multiply the homogeneous 3D co-ordinates of the cube and obtain its equivalent image co-ordinates. The frames from video with superimposed cube are shown in figure 10.

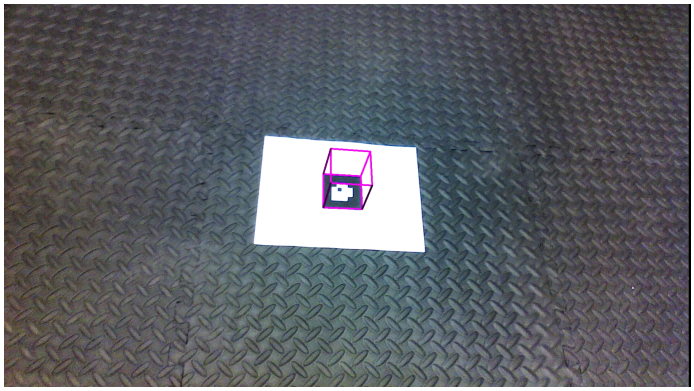
III. Filters to obtain smoother results

It can be observed from the videos that the projection of the cube is not very stable. I think the main reason for this is the camera calibration matrix. Since the K matrix plays a major role in getting the rotation and translation components, an error in it might affect the P matrix calculation, which in turn affects the final results. I tried multiple approaches to filter out this noise, however, could not come up with a robust solution. I will be discussing in brief what I implemented and the drawbacks of the chosen methods.

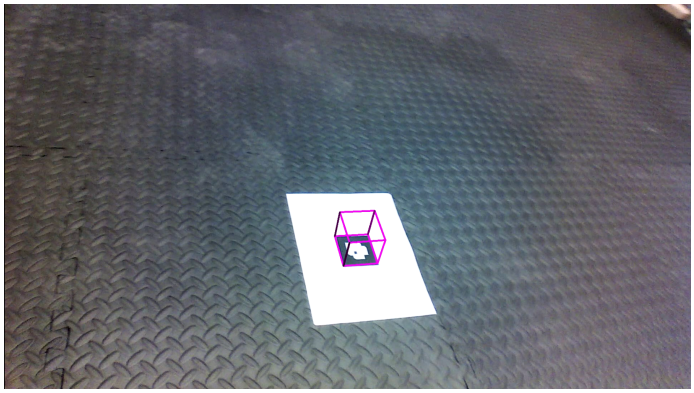
A. Kalman Filter

Kalman filter assumes that the system is linear. I used a Kalman filter to estimate position and velocity for each corner. So, the state space model can be written as,

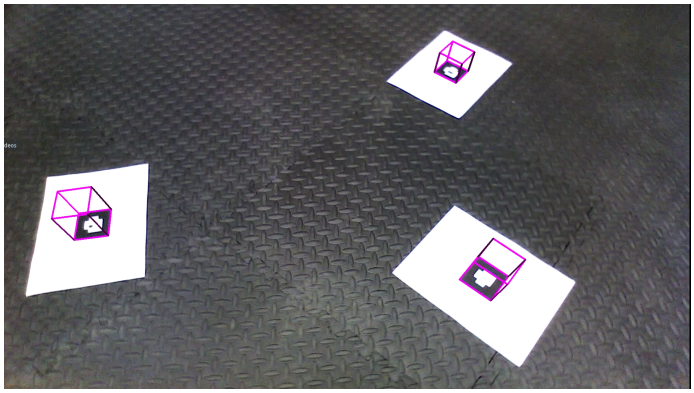
$$x_{k+1} = x_k + v_k\delta t + \frac{a\delta t^2}{2}$$



(a)



(b)



(c)

Fig. 10. Cube superimposed on AR tags

$$v_{k+1} = v_k + a\delta t$$

Where, x_k and v_k are corner position and corner velocity at time *k* and *a* is the input acceleration. Following are the problems with the above model:

1) Initially, I assumed acceleration *a* as 0. However, this would mean the model has a constant velocity in the x and y direction, which is not the case.
2) We can not assume a constant input acceleration. Because, that would mean the point is moving only in a particular direction, which is again a wrong assumption.

3) We need a mean to measure the acceleration. I tried calculating using the current and previous velocities, but the values were way too noisy. And a noisy acceleration will result in a noisy velocity, which in turn will affect the future calculations for acceleration values. So, this approach will not work.
4) Getting acceleration values from an external sensor, like an IMU, might help us in better pose and velocity estimation, but I am not sure about it.

Extended Kalman Filter considers the system as non-linear, so it might help in solving the problem. However, I was not able to figure out the non-linear state equations.

B. Weighted Moving Average

I tried implementing a simple moving average filter. The results are fine when there is little or no rotation, but the pose gets distorted when there is a drastic change in position or orientation. I have attached the filtered results as well for comparison. To understand what is happening, I plotted x values for one corner wrt time. Refer to figure 11, where blue color is for unfiltered pose and orange for the filtered pose.

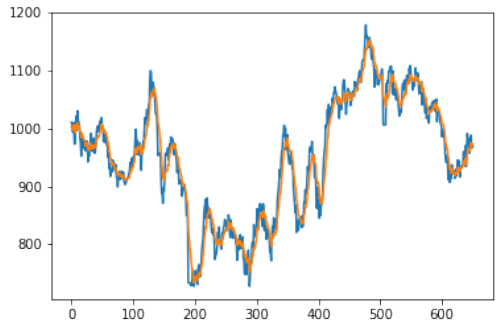


Fig. 11. Moving average filter for x values of corner 1

REFERENCES

[1] https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html
[2] https://drive.google.com/file/d/1q85pRoRiDmX3jzGRCYozY1dJZTx6Tu60/view
[3] https://www.pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/