2022

ENPM673: PERCEPTION FOR AUTONOMOUS ROBOTS

PROJECT 1

CHUKKALA BHARADWAJ | 118341705

PROBLEM 1: AR TAG DETECTION

ANSWER:

Brief overview before getting into the problem

What is an AR Code and why do we use it?

An AR Code (Augmented Reality Code) is a type of barcode containing the information needed to access a 3D content that can be reproduced in an augmented reality environment. AR Tags help us to calibrate and track camera position in a scene. These markers generally look like QR code sequence and have Tag ID, orientation information and, to make them easy to recognize black padding attached to the outer layer.

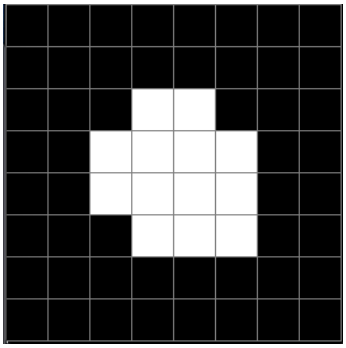

Fig 1: AR Tag

a) AR Code edges and corner Detection

How to detect?

We Know that edges are nothing but high frequency signals. In every image there are edges and they are in combination with many other frequencies. Now channeling out the lower frequencies depending on our requirements will leave us with high frequency signals, which are nothing but the focused edges.

Of course, there is going to be a lot of noise that comes along with it. Now there are ways we can reduce the noise. We can use filters to denoise/blur/smoothen the image. I have used the Gaussian blurring technique to smoothen my image before trying to detect edges. The formula for a Gaussian blur function.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Once we blur the image, we can use a High band Pass filter to single out the higher frequencies in the image. A High band pass filter that I have used is the circular mask. A Circular mask will have a radius component which bounds the frequencies to be high. The Higher frequencies when viewed as a magnitude spectrum will show us the singled-out edges in the image.

All the calculations are to be done in the frequency domain, so we use fast Fourier transform (famously called as fft) to perform the masking and detection. Then we apply inverse Fourier transform to revert the image to its original domain.

Fourier Transform function

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx$$

Inverse Fourier Transform function

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega x} d\omega$$

Steps:

- Firstly, I converted an extracted frame from the video to grayscale and then I computed the FFT of the grayscale image.
- Now we define a Gaussian mask that we will use to blur the image. To do this we shall multiply it with the FFT that we got as on output.
- This will channel out the lower frequencies and the noise is reduced.
- Now we must bring the image to original domain. I have performed an inverse FFT to do that.
- Now for better results, we convert the grayscale image into a binary image by giving it a threshold value.
- Now to single out the higher frequencies, I define a high band pass filter/a circular mask to do that and multiply with the FFT of the blurred image.
- Again, do the inverse of this FFT to get the edges.

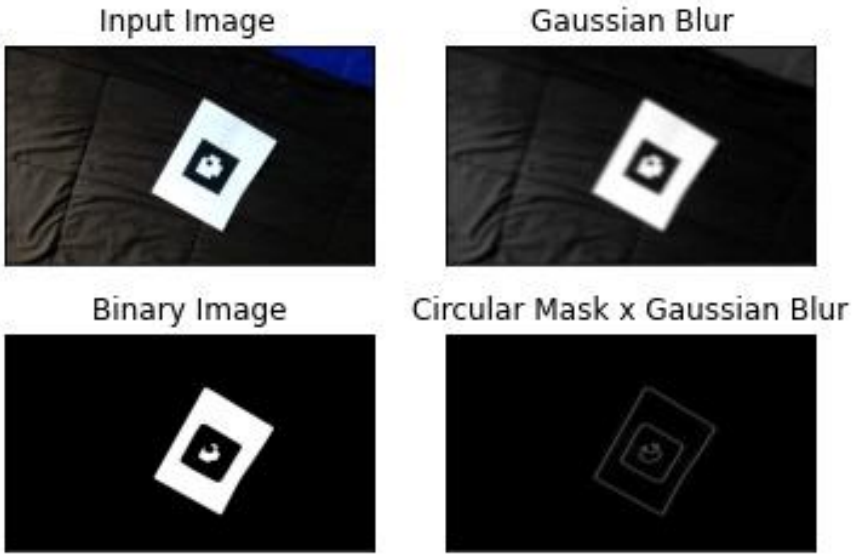

Fig 2

b) Decode custom AR Tag

How is it encoded?

The marker that we will be detecting and tracking in this project are custom markers. These are made up of 8x8 boxes as shown in the following image. The outer most layer of this tag, which is the 2 blocks of black boxed, this is called the padding layer. This is added so that tag detection is easy in the frame of the video on any contrasting background. The following layer of single blocks just inside this padding represents the orientation of the tag. We can see that there is white block is located at the bottom right, this gives us the information that this poses it represents the upright position of the tag. The innermost 4 blocks represent the ID of the tag. The inner most grid adds some significance, the left bottom is the most significant bit, and it goes till top left in an anti-clockwise direction to the least significant bit.

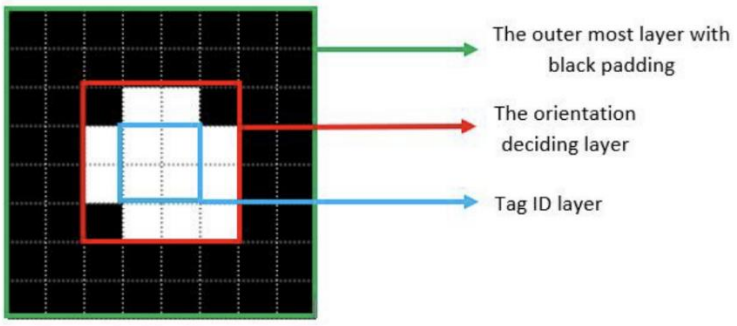

Fig 2: AR Tag Encoding

How to Detect the Tag?

- To detect the Tag corners, I had defined a function which uses, cv2.harrisCorner() inbuilt sub function to detect the corners in an image.
- I have firstly defined a respective kernel that performs the operation of blurring the processed image with applied filters attached to it.
- Once the kernel is defined, I have used the morphological process of erosion to shrink the image.
- Then on the eroded image, I performed another morphological process called dilation which will bloat the detected features in the eroded image.
- On the eroded image, I used Harris corner detection to get a list of points, from which I have extracted the sheet corners first by calling the maximum and minimum X and Y coordinates of the image.
- After I got the sheet corners, I then popped the corners that were detected for sheet and stored the inner corner of the tag as my corner points.

- Fact: The Harris corner detection works by taking in a criterion to bound the coordinates that are detected.

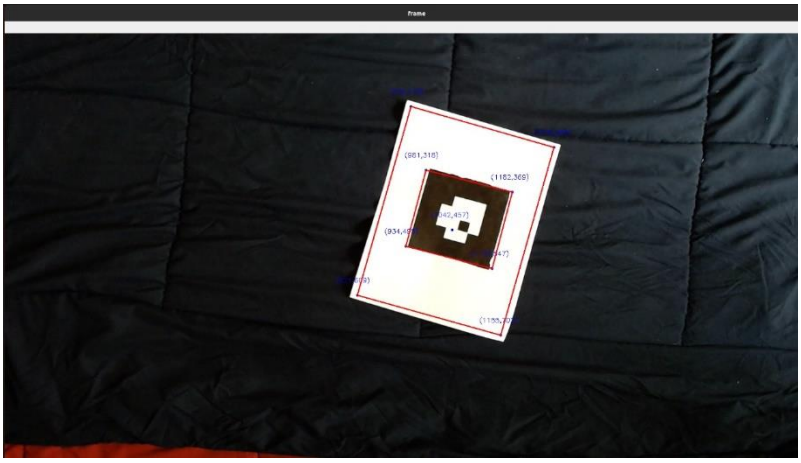

Fig 3: Detected edges and corners

How to Decode?

- Firstly, we convert the image into a binary image, then I have resized its dimensions to 160 x 160
- Now we know that the AR tag has 8 layers in two dimensions, so I have divided it into an 8 x 8 grid. Now each block becomes the size of 20 x 20.
- Now we must extract 4 x 4 innermost grid from the AR Tag, to extract it we remove the padding layer. After the extraction is done, we will check which of the corner blocks has higher value, the one that has the orientation of the AR Tag. We can check this by comparing binary values, where the corner block that we need will have a value of 255, because it is white.
- We can see that the lower right corner block has a higher value, and this gives us the idea of the AR Tag being upright. So now we will rotate the tag to find the upright position.
- Now after this, we must find the most significant bit and the least significant bit from the innermost 2 x 2 grid.
- We can convert the 2 x 2 grid to a 1 x 4 grid by the method called flattening. Now the leftmost bit is the least significant bit and the rightmost bit is the most significant bit.

Warping and Decoding (steps)

- Firstly, after the tag is detected, we need to decode the tag. To decode the AR tag we need to bring it to an approachable state, where we can understand the AR Tag.
- To bring the AR Tag to the upright position and then get the inner layer details of orientation and significance, we need to perform an image warping function.

- I have performed inverse warping to get the AR Tag ready for decoding.
- Now that the AR Tag is ready to get decoded, I have created a function based on the decoding (Decoding details are mentioned above) strategy. The orientation of layer was figured and the inner most layer 2x2 grid is taken.
- Now we flatten the grid into a single layer and find out the least and most significant bit from the flattened grid. The left most block is the LSB, and the right extreme is the MSB.

Fig 4: AR Tag Inverse warping

PROBLEM 2: TRACKING

ANSWER:

Brief Overview before solving the problem

What is image warping?

Image warping is the process of digitally manipulating an image such that any shapes portrayed in the image have been significantly distorted. Warping may be used for correcting image distortion as well as for creative purposes.

What is homography?

Two images are related by a homography if and only if both images are viewing the same plane from a different angle. Homography lets us relate two cameras viewing the same planar surface; Both the cameras and the surface that they view (generate images of) are in the world-view coordinates. The homography matrix is a 3x3 matrix but with 8 DoF (degrees of freedom) as it is estimated up to a scale.

$$s\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

When we divide x,y by (alpha or z), we recover the pixel coordinates of the object in the image. This is mathematically called homogeneous to cartesian coordinate conversion.

$P(x', y', Z)$ and $P(x, y)$

$$x = \frac{X}{Z} \ and \ y = \frac{Y}{Z}$$

We introduce homogeneous coordinate system to make the pinhole derivative nonlinear equations to linear model. The inverse will give us the 3D coordinates of the object in real world

What is Projection of an image?

Projection of an image is nothing but a model that allows us relate object locations in the real world to the corresponding pixel locations in the image through a perspective.

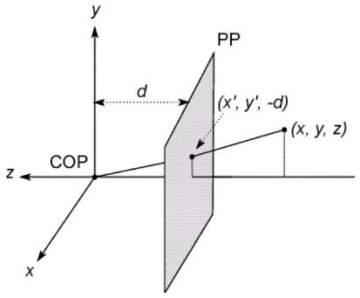

Fig 5: Perspective Projection

While projecting we keep the depth factor(z) as 1 but we divide the x and y coordinates with because it plays a key role in understanding how far objects in the image from each other.

What is a K matrix?

K matrix is a camera calibration matrix that represents the intrinsic parameters of the camera, where the units of k are (pixels/length). The K matrix is a 3 x 3 matrix that provides us with a transformation between a ray and an image point in Euclidean space.

$$K = \begin{bmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Here The intrinsic matrix K contains 5 intrinsic parameters of the specific camera model. These parameters focal length, image sensor format and the principal point of projection. The parameters $\alpha_x = f.m_x$ and $\alpha_y = f.m_y$ represent focal length in terms

of pixels, where $m_x$and m_y are the inverses of the width and height of a pixel on the projection plane and f is the focal length in terms of distance. γ represents the skew coefficient between the x and the y axis, and is often 0. u_0 and v_0 represent the principal point, which would be ideally in the center of the image.

What is Bilinear Interpolation?

Bilinear interpolation is a resampling method that uses the distance weighted average of the four nearest pixel values to estimate a new pixel value. The four cell centers from the input raster are closest to the cell center for the output processing cell will be weighted and based on distance and then averaged. Bi-linear interpolation means applying a linear interpolation in two directions.

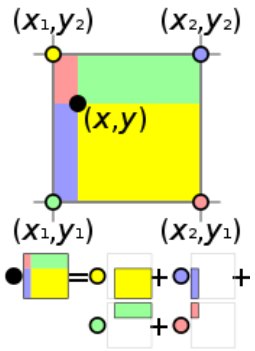

Fig 6: Bilinear interpolation

a) Superimposing image onto the Tag

Now that we have understood how we detect corners, detect the tag and decode it. We will be using this information to superimpose an image onto the AR Tag in all frames of the video.

- First things first, we need to understand that to superimpose an image on the AR Tag in an upright position we need to map the coordinates of the upright image to the upright pose of the AR tag.
- The next step would be to keep the image oriented and aligned with the AR Tag in all frames of the video, where there is rotation and translation.
- We extract the tag from each frame and compute its orientation as discussed above and then depending on what the orientation is, we rotate our corner points in that direction. To do this we store a count value when we warp the tag and use a point transformation function which will translate and rotate the points accordingly.

- The Testudo image that we want to superimpose should have the same orientation as the AR Tag for standing upright in all frames of the video.
- I have used the technique of bilinear interpolation to stabilize the superimposing process for the whole video.

Fig 7: Superimposed Testudo on the AR Tag

b) Placing a virtual cube onto the tag

We know that a cube is a three-dimensional entity, hence to place it on the AR tag we need a 3 x 4 projection matrix to project 3D points into the image plane. When we generally perform 2D homography the basic assumption while calculating the homography matrix would be that the points are planar, hence the depth factor z =0 for all the points. Now that we are projecting a 3D entity on to a 2D plane we have z values as well. Therefore, we modify our H matrix to suit our requirements. To tackle this part of the project I have followed the following steps:

- To do this firstly we have to get the projection matrix and from above briefing about what projection, camera calibration matrix is and what homography is, we can create a projection matrix using the Homography matrix H and the Camera Calibration matrix K.
- To calculate the Projection matrix P, we know that $H = KB'$
- Now from this equation we can get $B' = K^{-1}H$. B needs to be obtained such that $B = \lambda B'$
- We compute the modulus of the B' and if it is negative we manually make it positive by doing $B' = -1 * (B')$

- Now we calculate lambda from the formula $\lambda = \frac{2}{||b_1||+||b_2||}$ b1 and b2 are column vectors of B'.
- B can be further decomposed into two components and written as [r1 r2 t], where r1 and r2 are rotational components and t is translational component.
- We know that the rotation matrix is orthonormal therefore it's last column has to be perpendicular to the first two. Therefore, we get r3 = r1 * r2
- The final projection matrix can be written as $P = K[r_1\ r_2\ r_3\ t]$
- 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
- I created a function called projection_matrix() and with the obtained data, i is calculated the projected point coordinates which then I passed as arguments to the function which is then used to draw the cube on the tag.

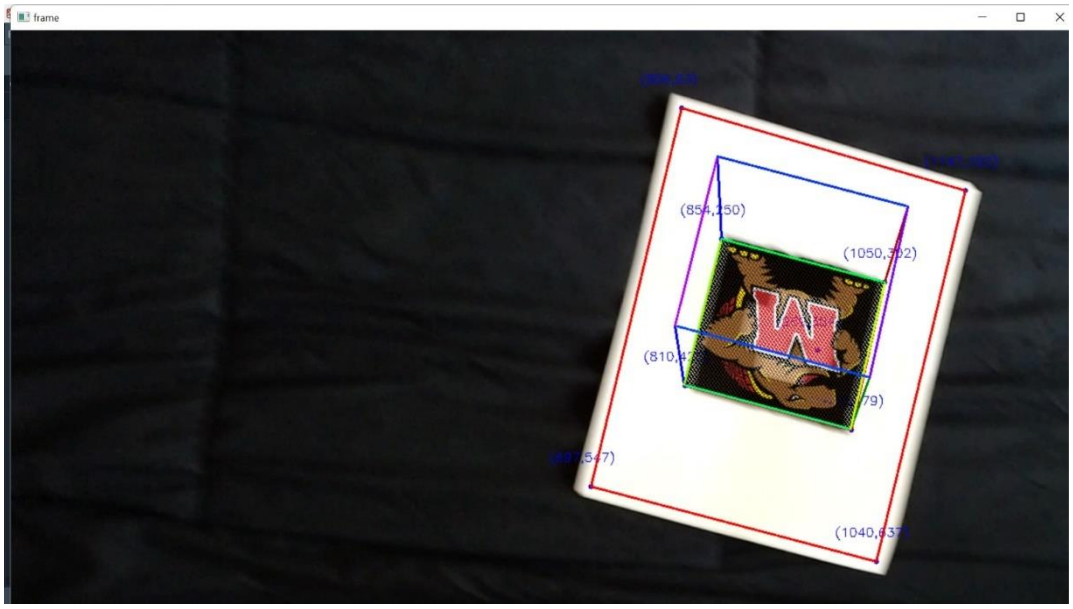

Fig 8: Superimposed testudo and projected cube aligned on the Tag. (Single frame)

Challenges faced:

- The first problem was to detect the edges properly, I used canny and I had a bad output then I shifted to gaussian smoothening and used a high pass band filter to detect edges.
- I have learnt from experience of this project is that Shi- Tomasi corner detection doesn't work properly for rotation and translation of frames. Harris corner detection works much robustly in terms of this issue.

- I even tried to manually write a function to perform hough transform but it didn't work.
- The warping proved to be a bit picky, because forward warping didn't work for several attempts, so I went with inverse warping.
- Superimposing the testudo was the most difficult task, the corners that get detected went sideways in many frames and the output was not as expected. After many attempts of fine tuning, I tried bilinear interpolation to stabilize the error and superimpose the testudo image a bit more consistently.

Output Video Link:

https://drive.google.com/drive/folders/16Z-MYEmpfb_pkcpS2bt4VNSMnPUThi37?usp=sharing

REFERENCES:

- https://www.cse.unr.edu/~bebis/CS791E/Notes/CameraParameters.pdf
- https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html
- https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/EPSRC_SSAZ/node3.html
- https://theailearner.com/2018/12/29/image-processing-bilinear-interpolation/