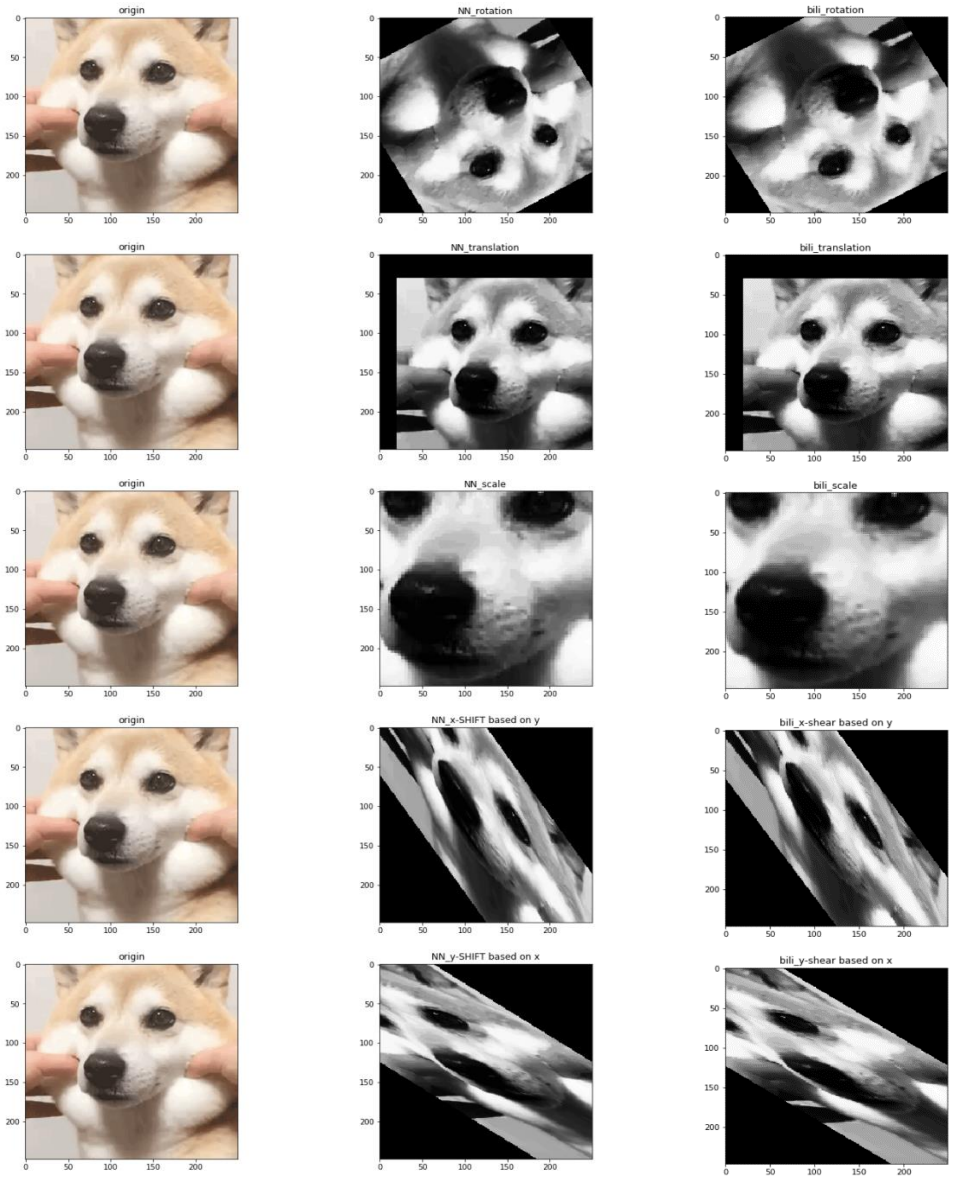
HW4 Theoretical

lw2435 Lifan Wang

==========

Q1(a). Backward mapping, nearest neighbor VS bilinear interpolation

Picture1 for nearest neighbor,Picture2 for bilinear interpolation



Row1: rotation anti-clockwise 210°

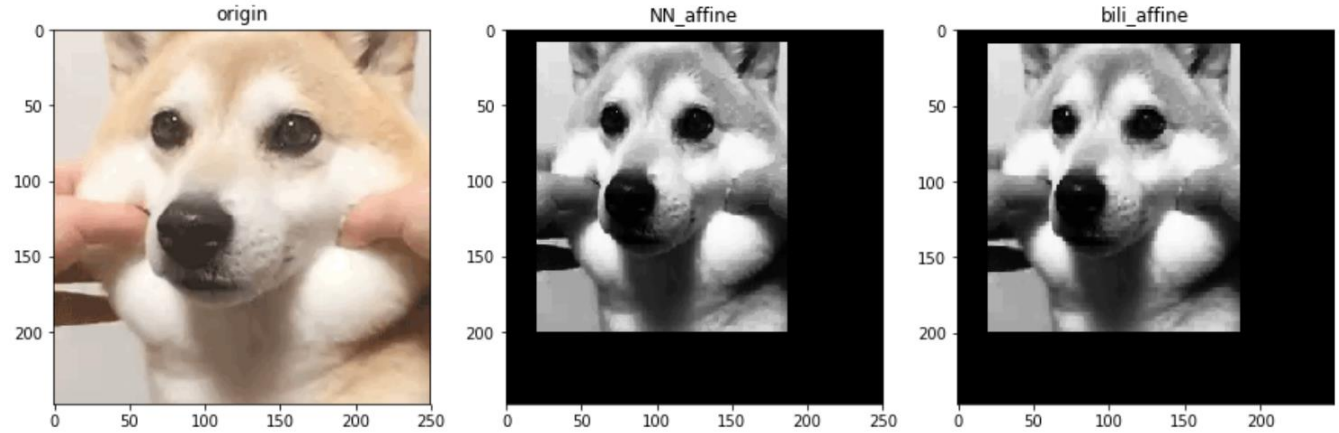
Row2: shift 20 in x and 30 in y

Row3: enlarge x and y both by 2

Row4: shear y by 1.5x

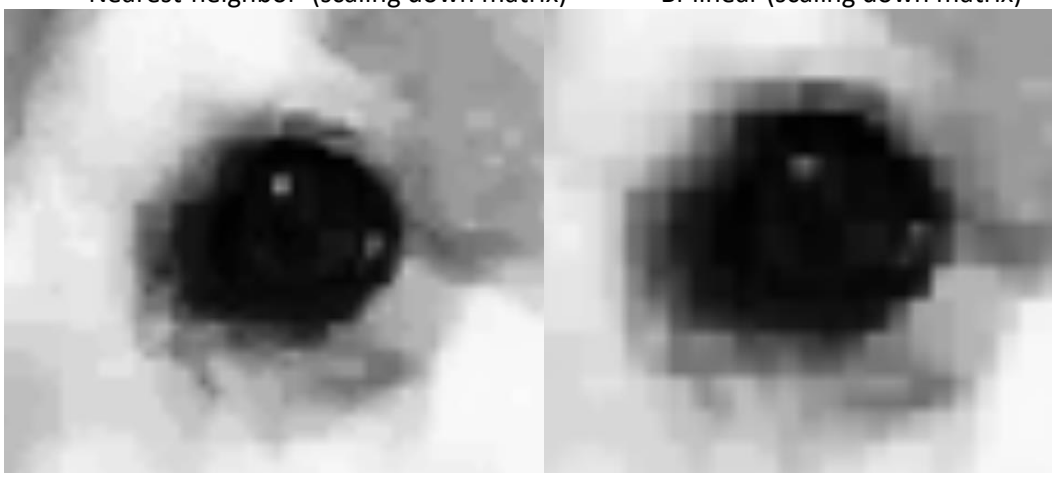
Row5: shear x by 1.5y

Comparison with affine transformation between Nearest-Neighbor / Bilinear interpolation:
We apply two matrixs to the original image and the result is shown as follows:
Scaling down
[1.3 ,0 ,25]
[0 ,1.5 ,33]
[0 ,0 , 1]
Scaling up
[0.5 ,0 ,25]
[0 ,0.5 ,33]
[0 ,0 , 1]

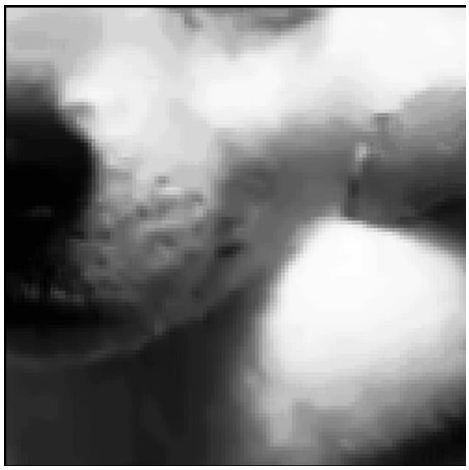


Zoom in to see the eyes

Nearest-neighbor (scaling down matrix) Bi-linear (scaling down matrix)



Nearest-neighbor (scaling up matrix) Bi-linear (scaling up matrix)



Discussion:

1.Differences between nearest neighbor and bi-linear interpolation: the NN method would have very shape edges between the different intensities ,but the Bi-linear method would have very smooth effects.

·When you use a scaling down matrix, NN seems to have a better effect

·When you use a scaling up matrix, bilinear seems to have a better effect

That's because in the bilinear case, you map the in-between intensities between intensity 'cliffs' to the new enlarged matrix, which leads to smooth results. While in the NN case, there is no such a transition.

2.Both would lose the very thin lines when applying a scaling down matrix.

3.The coordinate in the matrix is like This: with x stretches down, and y stretches to the right. And since we define the backward transformation as the matrix applies, the scaling factor (which is 1.3 in the case) would have a scaling down effect.

Algorithm:

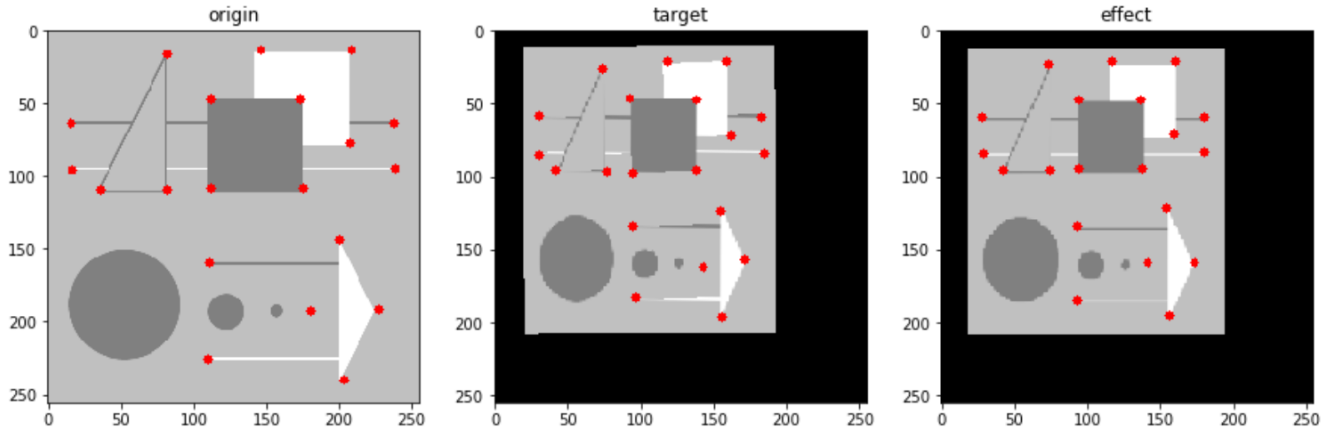
Step1:Scan the output image in order and map the pixel coordinate to the original image.

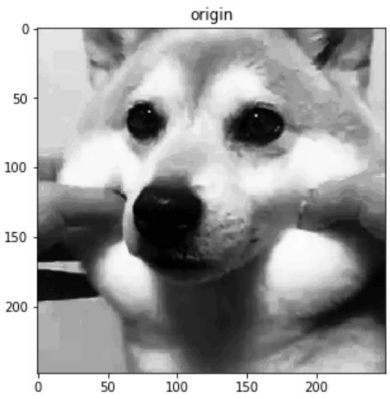
Step2:Fetch the intensity value in the original image and apply it to the new image.

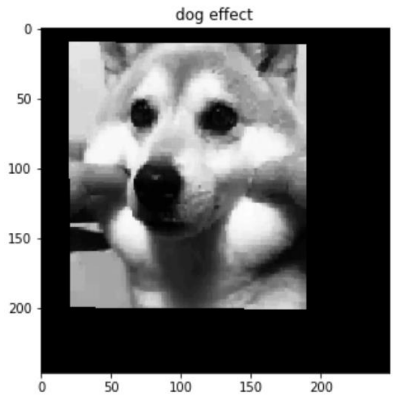
Step3:Whenever the transformation put the query out of the range, it will return 0 as the value.

Step4:Whenever the transformation put the query not at a pixel location, then round it to the suitable location to fetch the value(using Nearest Neighbor or Bi-linear interpolation)

Q1(b) Template matching







Algorithm:
Step1: we use an opencv library to select the landmarks in pairs correspondingly
Step2: we stack all the pairs of landmarks together and solve the linear system with python numpy.linalg.lstsq package and get the transformation matrix T
Step3: Use the T to transform the original image to a new image and compare it with the original target image.

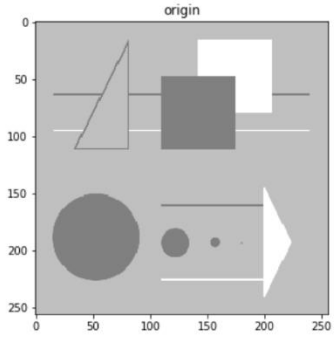
Discussion:
1. The more landmarks you have chosen, the more accuracy of results you would achieve ,by saying accuracy, I mean how similar is the new generated image similar to the referenced target image.

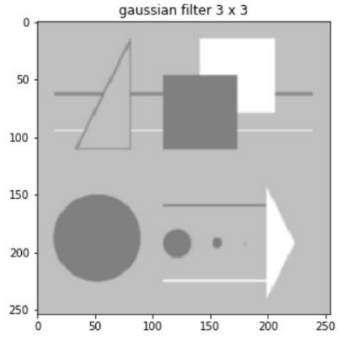
Q2(a) Hough Transformation for straight lines without edge orientation

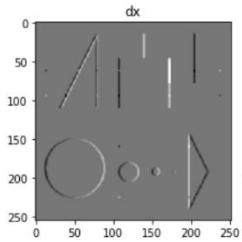
Algorithm:
Step1: use the "edge_laplacian" function to get a clear boundary with threthoding
Step2: prepare Up-sampling and Down-sampling function due to the axis scaling and size of polling cells ,and decide the mapping function between different and different true values.
choose $\triangle\rho$ =1 and $\triangle\theta$ = 1 as the cell size. Choose the
X_axis_scale=theta _resolution/cell_size=720/1=720 to be the number of intervals within $[0,\pi)$
y_axis_scale=max/y_compress_scale/cell_size=1200/1/1=1200 to be the number of intervals within [0,1200)
We need to tune the resolution very carefully, (it's better to have a high resolution), so that the result line would be precise and avoid lines which intersect with one point but have very slightly different slopes.
Step3: prepare axis shift function to make up for the inconvenience ,which is you can never plot an image which has negative x,y coordinates. We definitely need the image matrix to do peak detection!
Step4: After we get ρ and θ in the Hough space based on the function $\rho=x\cos\theta+y\sin\theta$, we do the voting step, here we also applies the importance of voting based on the intensity. we coud use Gaussian kernel and peak detection to select the key points.
Step5: Return the key points from Hough space and draw the lines. Here is an issue. We need to avoid gap between edges by drawing Y based on X whenever the slope is less than one, and X based on Y when more than one. Sometimes we need to use up-sampling(nearest neighbor / bilinear interpolation) and down-sampling whenever we consider the x axis resolution is too small and want to stretch it. Otherwise the stretching would cause lines to be 'cut off' !

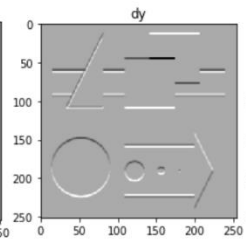
Results:

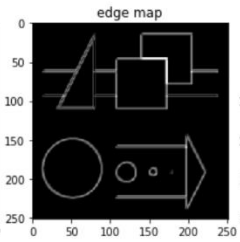
edge threthod=0, peak threthod=60% , peak kernel size= 21, running time=3 min,

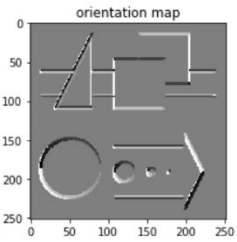


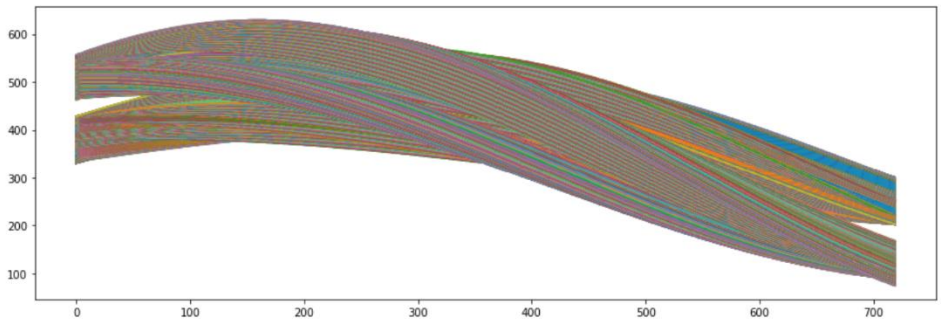


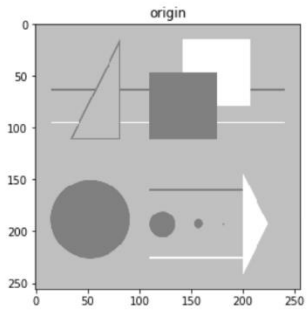


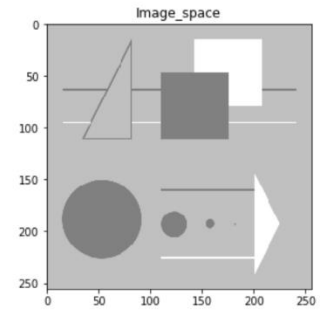


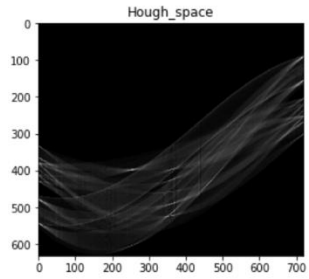


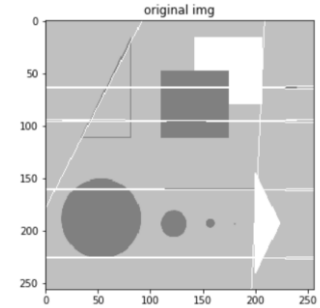




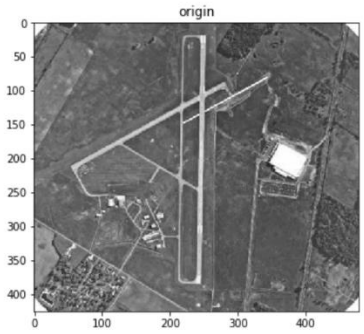


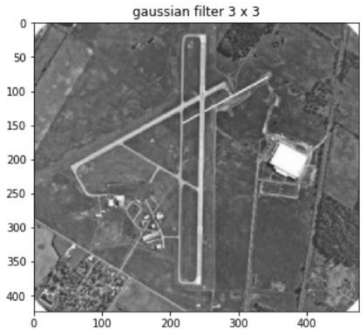


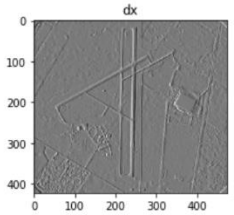


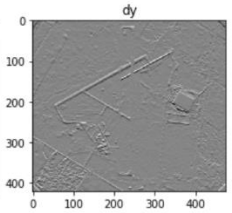


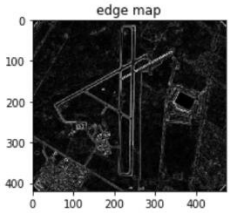
edge threthod=0, peak threthod=50% , peak kernel size= 11, running time=15 min,

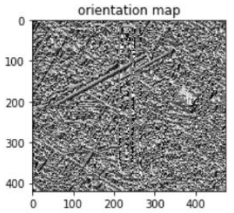


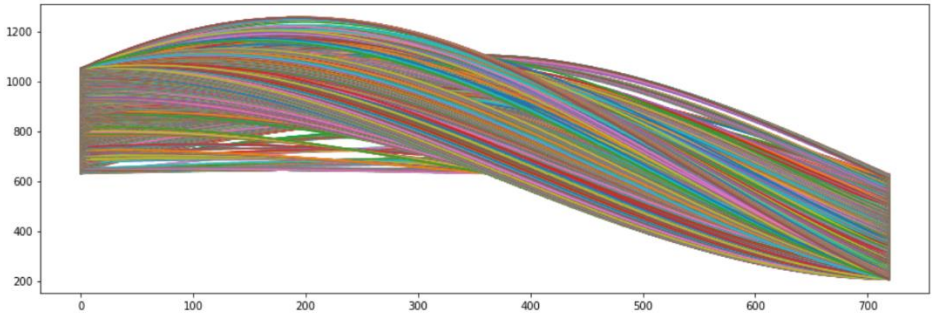


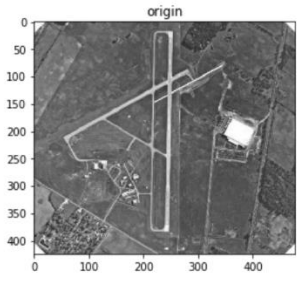


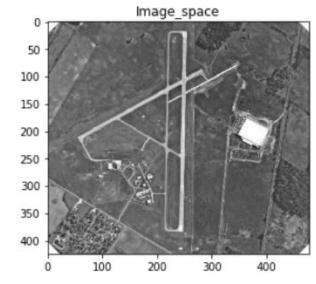


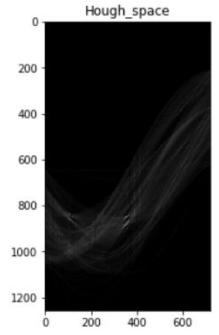


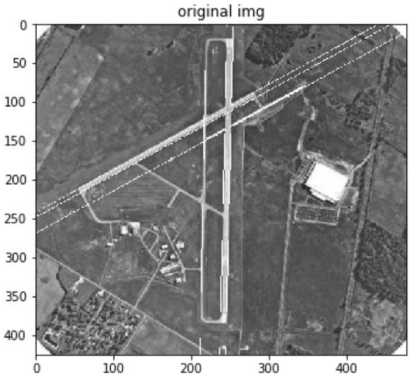








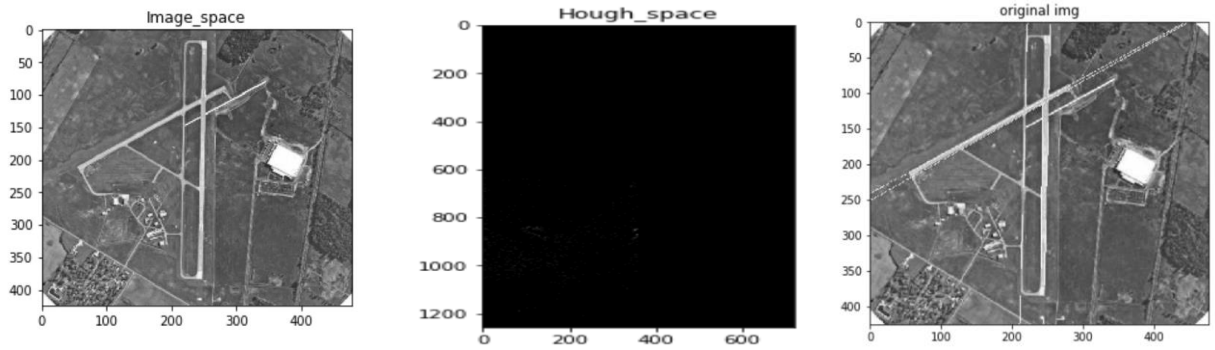




Discussion:
1.For the parameters listed above, peak_threthod controls how many lines at most to be detected. If you allow the threthod to be too small, then there would be too much details. For example, if there is a circle in the image, it will produce several lines tangent to the circle
Peak_kernal_size controls to which degree would two close lines consider as one line and thus not to be counted twice. If you choose the kernel size to be too large, it will function as "gathering two highest points together as one high point." So some details would be lost. If you choose too small a size, you will see many lines with similar slopes intersect in one point, all of which stand for the same edge redundantly.
2. When produce the edge map from the original image , we will see the former is 4 pixels smaller than the latter one.(Gaussian 3x3 convolution and derivative filter 3x3 convolution), so when considering the key points, we need to add this offset.
3. When consider the importance of votes based on the image intensities, you would see some small house like in the airport map in the second example, the white small house would have a higher priority than a long line with weak intensity. This would also cause a problem.
4. When performing the peak detection, if you apply the gaussian filter but do not use the padding=reflective mode, you will see the boundary would more easily be smoothed down and be easily neglected. So that's why many students do not retrieve boundary points in Hough space, which is horizontal lines in the Image space.

Q2(a) Bonus : Hough Transformation for straight lines with edge orientation
Algorithm:
The process is generally the same as the previous one ,except for that ,when doing Image-to-Hough space transformation, each edge(pixel) in image space only vote for one point in Hough space. Since each edge has its own ρ and θ, we no longer need to draw a whole cosine wave for ever pixel in image space. That saves a lot of computing time.

Results:
edge threthod=0, peak threthod=30% , peak kernel size= 11, running time=3 min,



Discussion:
1. Since each edge has its own ρ and θ, we no longer need to draw a whole cosine wave for ever pixel in image space. That saves a lot of computing time.
2.Hough space would be really dark because we only select a limited number of cells to vote, so the contrast is not so obvious. But we can still see the airport lanes being detected through the process.
3.When we do the query for every edge in the image space, you would see the query range is [0,360),while the $\theta \in [0,180)$. That's the orientation problem. Solution is also easy. Whenever you query a orientation which is 210, you simply mod 180 to make it the same as 30 degree. This process is easy to forget but important.

4. It saves computing time! As is shown in previous example, that costs 15 min to finish. But in this case ,it only cost 3 min to finish, which is way more faster.