PROBLEM 1

METHODS & ANALYSIS

Preparing Correlation Array

Correlation array was created using matlab function `corr2`, for each pixel of the car image traversing through it. The correlation value of each pixel was calculated by finding correlation with template image (size PxQ) comparing it with a matrix of PxQ size of the original image starting from that pixel. Matlab function `corr2` returns 1 for exact same match and lower if it doesn't match the images.

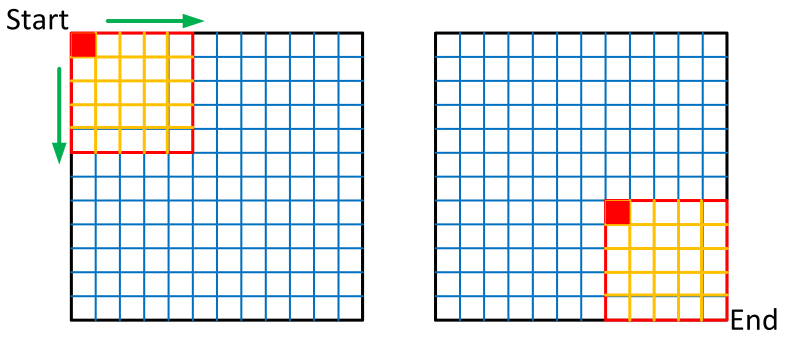

Figure 1.1: Scanning Method followed by the Program

For example, a correlation value of (x,y) pixel of original image (car.png) was calculated from correlation of sub matrix (x:P+x-1,y:Q+y-1) and template (1:P, 1:Q) (shown in figure 1.1). Assumption was that there was no object partially available in the main image and all the objects to be detected are completely available in full shape within the frame/boundary of the main image.

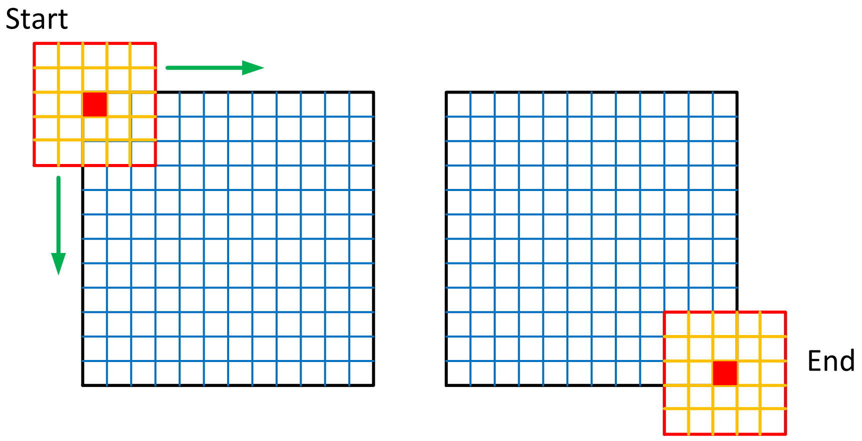

Figure 1.2: Standard Scanning Method

This is different from matlab `normxcorr2` which calculates correlation with sub matrix (x-P/2:x+P/2, y-Q/2:y+Q/2) and increases the correlation array size to (M+P x N+Q), where original image size is MxN and template size is PxQ (shown in figure 1.2).

Finding Peaks – Detecting Object in the Given Image

The program takes argument as correlation threshold (`corrthresh`) to select the peaks of the correlation array. Selection of correlation threshold was important:

`corrthresh = 1.00`: Only one peak was identified for the third wheel from the left.
`corrthresh = 0.75`: Peaks were identified in three zones for first three wheels from the left.
`corrthresh = 0.40`: Peaks were identified in four zones for all four wheels.

Now it was required to find a 'single' peak from each zone, especially when `corrthresh = 0.4` was chosen to detect the fourth wheel from the car image. In this case, more peaks were selected near the first three wheels. The correlation values were higher (>0.4) in nearby locations of the highest correlation value (which were 0.7 or higher). Local maxima were required to be selected for these zones. For this, all the nearby peaks [co-located by less than half of the height (P/2) and half of the width (Q/2) of the template image] are grouped and checked for highest correlation value. Only the peak with highest correlation value was chosen as final peak and rest are discarded for each region.

Making the Peaks

The final peaks are marked with red cross in the original image. Marking for (x,y) peak was drawn on (x+P/2, y+Q/2) pixel of the original image (car.png).

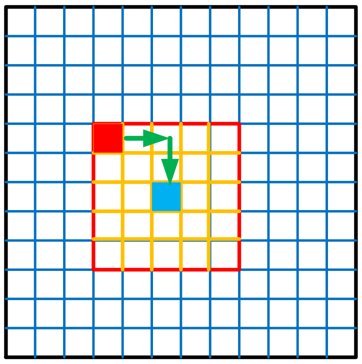
Figure 1.3: Shift for marking

OUTPUT OF PROBLEM 1

```
Maximum Peaks (at pixel location):
0.79 at (122,65)
0.93 at (120,108)
1.00 at (118,151)
0.44 at (119,196)
```

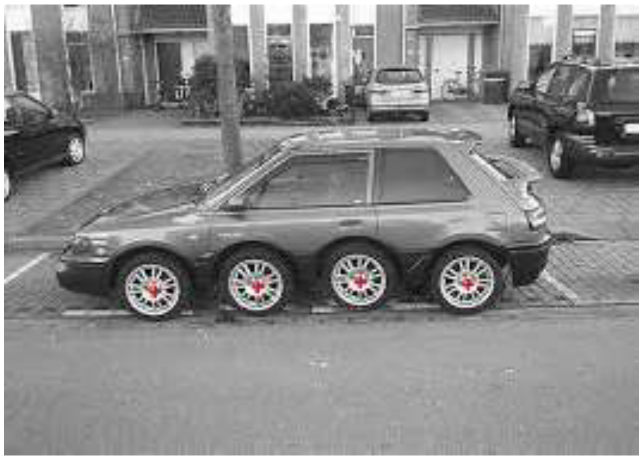

Figure 1.4: Output

PROBLEM 2

METHODS & ANALYSIS

Creating the Image

After creating a frame of 0s & 1s, matlab function `toeplitz` was used to get a striped image. Then matlab function `insertShape` was used to insert a white square centered in the striped image.

Calculating DFT

Matlab function `fft2` was used to create the DFT of the image and then it was centered using function `fftshift`. For displaying the shifted DFT found at this stage was done during display of the DFTs.

Creating a Mask

Afterwards, a mask was created to get rid of the stripes from the image. The stripes are the high-frequencies part of the image and they lie at the bordering around the center point of the shifted DFT. Hence the mask we need is nothing but a Low Pass Filter to remove the high frequencies (stripes). At first the built-in Gaussian filter (with sigma value 5) was used as fol:

```
mask = fspecial('gaussian', [IMAGESIZE, IMAGESIZE], 5);
```

However, the mask and final filtered image was like fol:

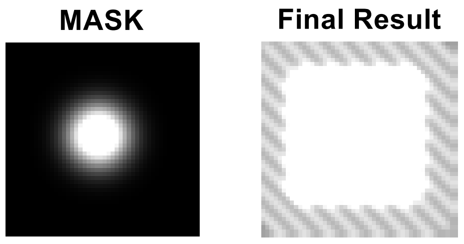

Figure 2.1

Then a separate low pass filter (binary) was used with a cut-off frequency (`freqcutoff`), which is the circular cutoff frequency normalized to [0 1]. The lower frequencies below cut-off frequency are mapped to '1'. All others are made '0'. Hence it becomes a binary circular mask. The code for the mask is as fol:

```
hr = (IMAGESIZE-1)/2;
hc = (IMAGESIZE-1)/2;
[x, y] = meshgrid(-hc:hc, -hr:hr);
grid = sqrt((x/hc).^2 + (y/hr).^2);
mask = double(grid <= freqcutoff);
```

The mask and final output (with frequency cut off 0.3) was like:

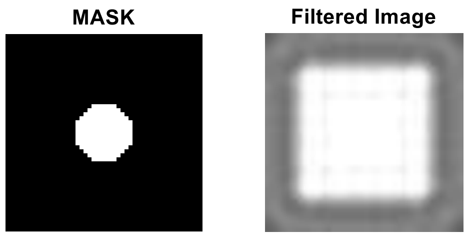

Figure 2.2

<u>Analysis:</u>

1. The built-in matlab Gaussian function uses a gradual change in the mask (grayed transition from white to black). The frequencies in the transition area are also calculated with low in the filtered image. That's why **faded stripes were visible** in the output (with sigma value 5). If we lower the value of sigma, we had a complete white box. If we increased the value of sigma, we were including higher frequency and darker stripes were re-appearing in the image and white box was getting grayed.

2. With custom binary mask, there's no transition area. All the frequencies below cut-off frequency were taken and above cut-off frequency were removed. As such the **stripes were also removed completely**. However, if cut-off frequency above 0.3, the stripes became visible again.

Applying Filter and getting Filtered Image

Shifted DFT from the image was multiplied with the mask to get rid of the stripes (high frequency components in the DFT). Then the resultant DFT was shifted back and inverse DFT was taken. The result from inverse DFT had some imaginary component, which was not significant. As such absolute value of the inverse DFT was taken to get the final filtered image.

Binarize & OPEN-CLOS on Filtered Image

After getting the filtered image, we can run binarize (`mybinarize`) and OPEN-CLOS (`myopen-myclose`) operations from HW1 and we can get a clean white square with black background.

OUTPUT OF PROBLEM 2

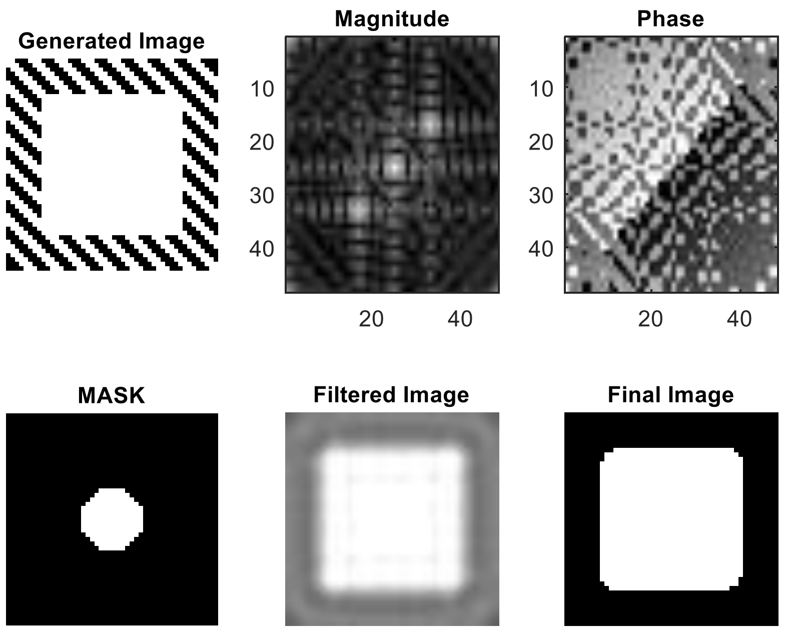

Figure 2.3: Output