Part 1

1.1

First, we padded the image if the image was no a square. We padded it so that its rows and columns become a multiple of some power of 2.
Then, we took the fast Fourier transform to find the real and the imaginary part of the image for all red, green and blue pixel. We then found the magnitude of all the red, green and blue using the formula: E(i; j) = log(R(i; j)2 + J(i; j)2). Then we convert all the three magnitudes to get a png image.
This represents the spectrogram of the original image.

Use the following to run the code
./watermark 1.1 bike.png bike_output.png

1.2

First, we padded the image if the image was no a square. We padded it so that its rows and columns become a multiple of some power of 2.
Then, we took the fast Fourier transform to find the real and the imaginary part of the image for all red, green and blue pixel.
Then we removed the noise from the real and the imaginary part of the image. We did this by changing the real and the imaginary values of all the pixels
whose magnitude as calculated by the formula: E(i; j) = log(R(i; j)2 + J(i; j)2) to zero except for the centre pixels whose value is kept intact.
After taking the inverse Fourier transform of the image we get the image values in the spatial domain which are then plotted onto a png image.

Use the following to run the code
./watermark 1.2 noise1.png correctednoise1.png

1.3

For this part as per the instructions we first found out the binary vector. Then added the watermark on the image. For doing this we used the given formula for making the changes in the bins. To find out the circular bins, firstly we calculated the center of the given image that is by dividing the row and column size by 2 and getting the center and stating it as h and v. Then we calculated the x and y values as x = r*cos(theta) +h and y = r*sin(theta) + v and similarly we calculated the symmetric values x1 and y1 by making an addition of PI to the given theta. Here we used 128 bits that is our L is 128 bits.

However, while doing this we observed that for different sizes of the image we had to modify the value of alpha. For instance, for an image of size 128*128 we used alpha value as 0.008, while if we used the same value for an image of size 256*256 them the image was distorted so we had to reduce the value of alpha. But doing that and keeping the value of radius as 30cwe got respectable values.

For Checking whether there is watermark present or not we used pearson's coefficient correlation formula. We found out that if we set the threshold above 0.01 then we are getting a watermarked image while any non-watermarked image has a negative coefficient correlation.
We used the following formula

Pearson Correlation Coefficient Formula:
r = (Summation of Xi * Yi - #ofElements * Mean of X * Mean of Y) / (SquareRoot(Summation of Square of Xi - #ofElements * Square of the Mean of X) + (SquareRoot(Summation of Square of Yi - #ofElements * Square of the Mean of Y))

Use the following to run the code
./waretmark 1.3 [pen.png] [output.png] add 123
./waretmark 1.3 [pen.png] [output.png] check 123

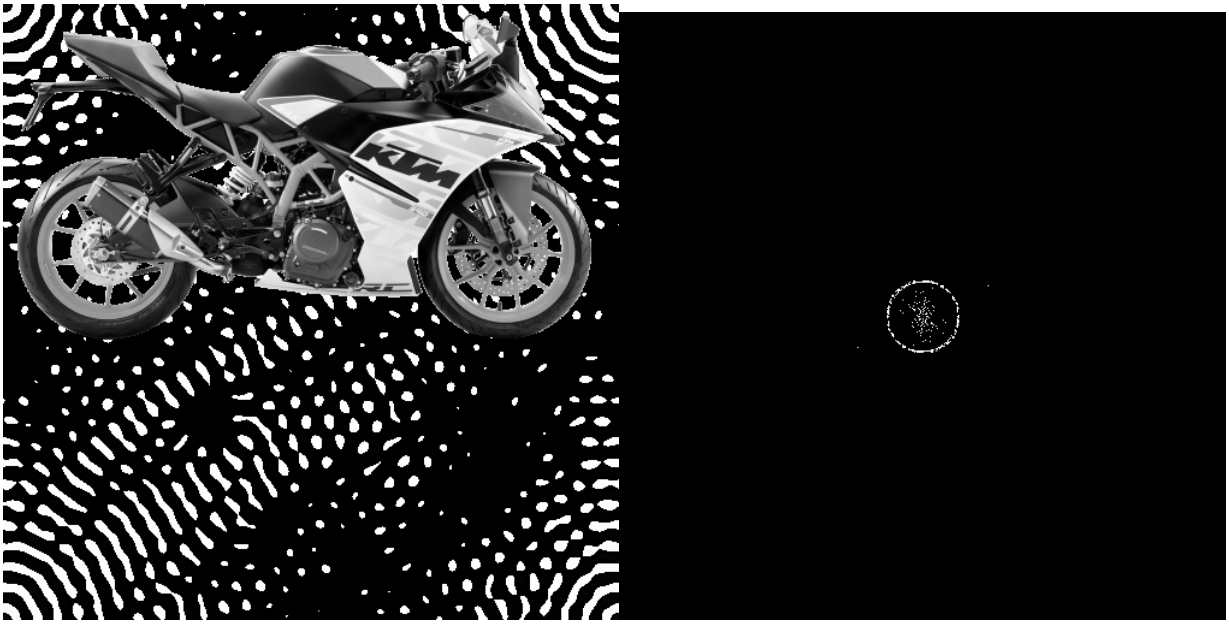
Ref:
https://wikimedia.org/api/rest_v1/media/math/render/svg/01d103c10e6d4f477953a9b48c69d19a954d978a

Sample Images and their spectrograms


Alpha 0.002

Alpha 0.055

Alpha 0.07



Alpha 0.04

Part 2.1

To implement convolution using a 2 dimensional kernel.

Function convolve_general in Detect.cpp imeplements the 2-d convolution and returns the convoluted matrix

Part 2.2

To implement separable convolution

Function convolve_separable in Detect.cpp implements convolution using 2 1-d kernels

Part 2.3

Edge detection

Function find_edges in Detect.cpp implements edge detection using sobel gradient algorithm and non-maximum suppression technique.

Part 2.4

For detecting the ICS in PCP

Function detect_ics in Detect.cpp implements detecting ics in an image.

(Detection of ICS)

Approach 1:

Using hough transform :-

Function Hough Transform in Detect.cpp implements hough transform and returns the transformed matrix

Function get_lines in Detect.cpp returns the lines in the image.

After this, I got struck om how to filter out rectangles using these lines

Approach 2

Using super vised learning :-

Additional script training.cpp scans the tagged ICS and extract features(Feature 1 :- 4 possible difference in intensities for the rectangles corresponding to ICS) ,2^{nd} feature. Is sum of all the gray scale pixel values in gray scale image

G++ training.cpp - o ./train

./train

The features are stored in a text file.

Using cosine similarity (for feature 1) , different rectangles are compared the trained ones and one with highest similarity score is chosen.

Using difference in summation of pixel values. In this approach difference in summation of values are calculated and the one with the lowest difference is chosen

The rest of code description can be found in inline comments of the script.