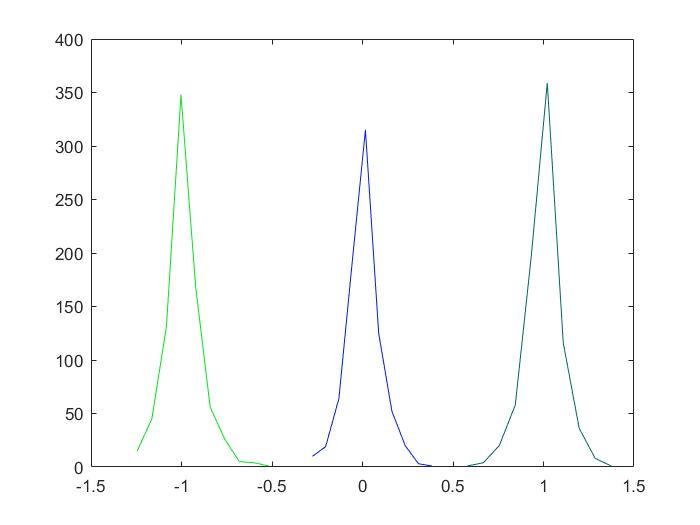
Media Security – Assessed Practical Three – Jordan McDonald

Part 1

In order to analyse this graph it would be useful to review the experiment context, three sets of 800 images are either embedded with a 1 bit message, 0 bit message or no watermark message. Therefore we would expect an even distribution of peaks in the chart above, however that is not the case, it is clear that the central peak is lower than the other two. This is due to the linear correlation method not being 100% effective, the graph shows it is possible for values in each array to be wrongly categorised. In addition to this the experiment does not perform an attack which shows the flaws of this method. The left peak represents a message bit of 0, the right peak represents the message bit of value 1 and the central peak represents no watermark. It is a caveat that the currently set threshold value used to classify the values in each array is currently 0.5, which may affect the results. In summary linear correlation can be used for watermark detection of 0 or 1 bit messages and has reasonable success, however it is not 100% effective as discussed above.

Part 2

|  |  |  |  |
| --- | --- | --- | --- |
| Threshold | Original image | Image (0 bit watermark) | Image( 1 bit watermark) |
| 0 | 0 | 1 | 1 |
| 0.2 | 0.916250000000000 | 1 | 1 |
| 0.5 | 1 | 1 | 1 |
| 0.7 | 1 | 0.991250000000000 | 0.993750000000000 |
| 1 | 1 | 0.538750000000000 | 0.441250000000000 |

Initially we will analyse the effect of the threshold on the un-watermarked original image. In the table there is a clear pattern where an increased threshold results in the detector giving an input as a 1 bit message more likely, which leads to an incorrect classification for the bottom three results for the original image. This shows the importance of a balanced choice of threshold, in this case 0.2 seems to be a good choice and is the only threshold value that gives the expected result. Next we will analyse the 0 bit watermark results from the table, a pattern has emerged that shows the output value decreases as the threshold increases. From this we can see that the detection rate gets weaker as the threshold increases, therefore it is pertinent that the threshold is carefully chosen. The same pattern is seen when we consider the 1 bit watermark in relation to a threshold, were the detection rate gets more incorrect as the threshold increases. This shows that an increased threshold increases the overlap of the peaks in the graph before, which leads to more images having a greater potential for misclassification. Therefore a reasonable threshold should be chosen, in this case 0.2 seems ideal.

Part 3

1. *OrIm = imread( strcat(num2str( imNo ), '.bmp' ) );* - Initially the imread function is called, the contents of which are read from a file and stored in the variable Orlm. As a parameter the ‘strcat’ function is called which performs a concatenation of two strings, to cast the value ‘imNo’ to a string the ‘num2str’ function is called. The result of this is then concatenated with the string ‘.bmp’ which allows the image to be read from file and allows a large amount of images to be read on each loop iteration.
2. *[Nc, Xc] = hist( cac, 10 );* - In this line of code ‘10’ represents the number of bins that will be used for the histogram. Bins are defined as the entire range of values into a series of intervals—and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable. ‘Nx’ represents the position of the bin centres in X and the values of these centres. ‘Nc’ is the number of elements in each bin container.
3. *x = find( (cac <= Thr ) & ( cac >= -Thr) );* - the find function finds the indices of non-zero elements that satisfy a certain requirement, in this case the inner set of brackets which establish the parameter by applying a logical expression. In this context it gets the indices inside the ‘cac’ array that are less then or equal to the threshold and greater than or equal to the threshold. The total indices that satisfy this are then stored in the variable x