**1. Cross Correlation in 1D and 2D (6 pages)**

**1. 1D singal offset**

**Time series**

In the case of time series, cross correlation produces a measure of the temporal similarity of the data sets. A correlogram plot of the cross-correlation values at each lag is useful for extracting noisy signals and synchronizing signals. Cross-correlations between the signals will peak at the lag in which the signals overlap. The lag is the number of time periods that separate the two time series. The delay is determined through cross-correlation of the signals. The peak cross-correlation value occurs at the lag where the signals have the greatest similarity. Figure … shows the correlogram between the two signals with what is likely random noise introduced, possibly from the environment or recording equipment. The noise conceals the signal pattern, however the correlogram shows a distinct peak cross-correlation at a zero lag. This indicates that the two signals are synchronized. For example, if these signals were from a radar, the peak occurs at a particular lag that is transformed into a transmission time through multiplying by the sampling period. The distance can be determined through multiplying the transmission time by the transmission velocity.

**1D signal application/ data[[1]](#footnote-1)**

The data used in this experiment is discrete time series data. A discrete-time signal x(n) is defined as a series of time instants corresponding to a sequence of quantities. Proakis et al. state that n in discrete signals represents the index of the discrete-time instants as the independent variable. This allows the signal to become a function of an integer variable. Therefore, x(n) refers to the nth-sample, and can be represented mathematically by a sequence of real or complex number.

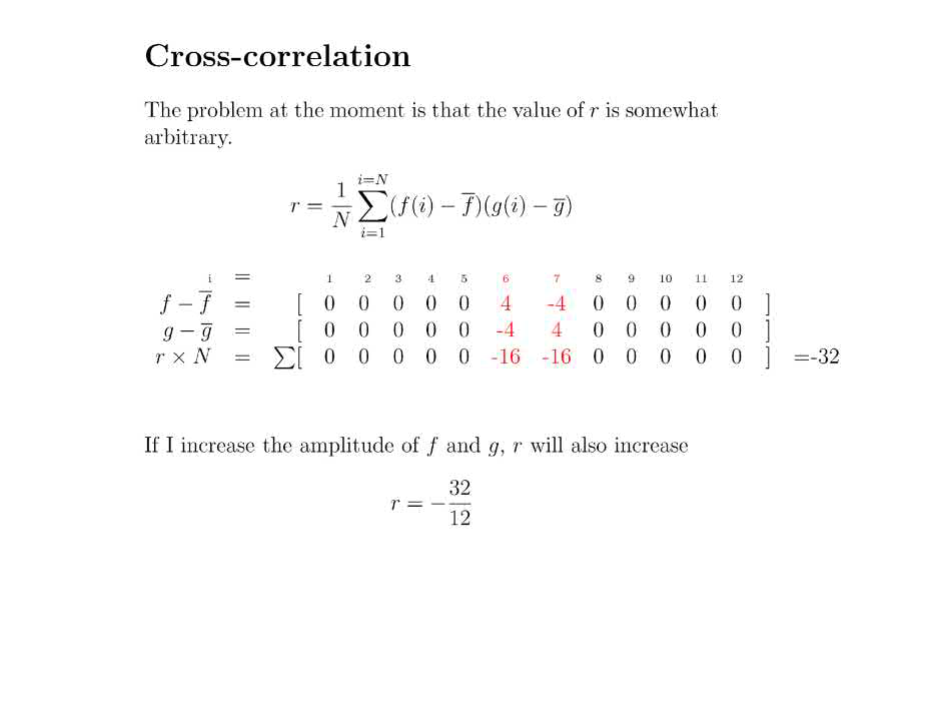
The signals used in the 1D case is are analogue time series, with constant sample period, T, the time between successive samples. The reciprocal of the sampling period is known as the sampling rate Fs, which represents the samples per second.

The signal data is stored in a .txt file, (UTF-8 encoding) with data point separated with a carriage return operator. The function *read\_file* opens the file from the a specified location in the directory, then l=using list comprehension, removes the header using the *islice(data, 1,..)* library function and *.strip()* to separate the numerical values from the carriage return.

**Analysis - temporal**

Applied to temporal 1D cross correlation, the theory outline in Chapter 2 is implemented to find the distance between two speakers given two output signals, respectively.

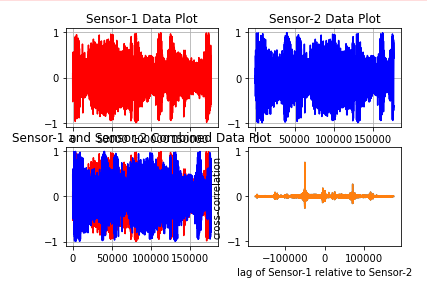
The chosen pattern and template signals are stored separately as lists and passed into the *find\_offset* function. *find\_offset* determines the shift by calculating the index of the maximum value determined from a list of cross correlation coefficients. To determine the cross-correlation coefficients, the template array is zero-padded with the length of the pattern array -1 at either end. Then the product of the overlap between the pattern array and non-padding elements of template array for an iterated offset position of the index. The corresponding cross-correlation function will show a maximum at the index where the overlaps are most congruent. This is returned alongside with the value of the index.



The normalised cross correlation of two padded signal arrays is found from the maximum of the convolution between the two time series signals and the corresponding index. For the two-signal data, the offset of the maximum cross correlation value indicates the number of sample periods between the beginning of each signal file and the maximum overlap.

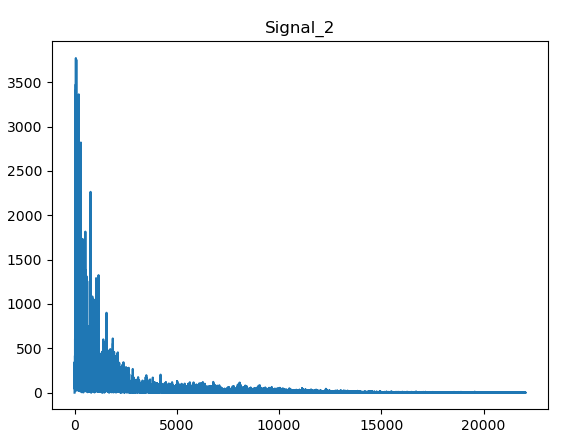
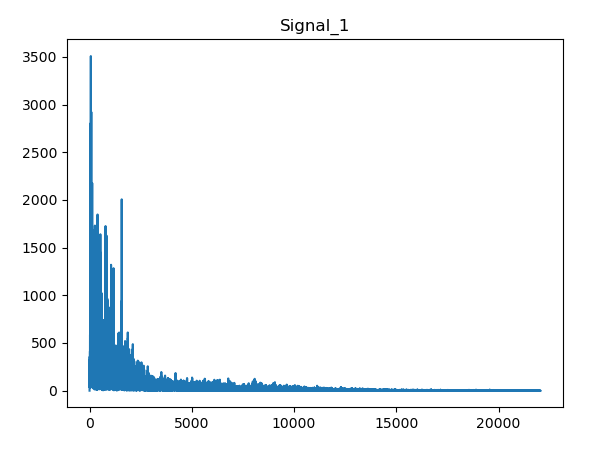
* Off-Set = -50082
* Off-Set Time = -1.136
* Distance between two sensors = 378.17 meters

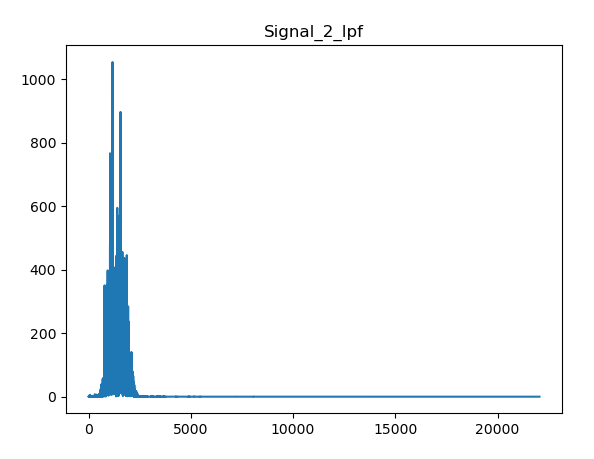
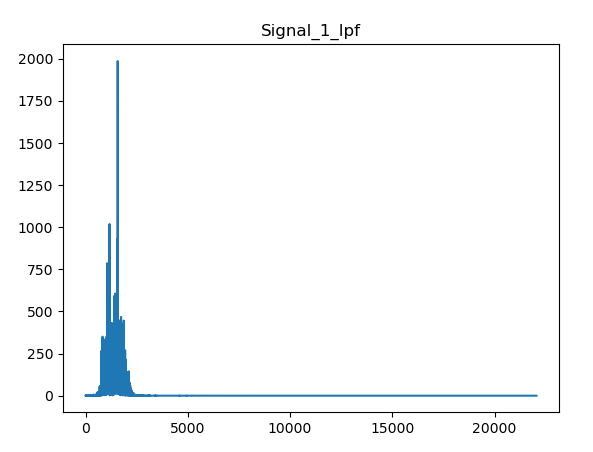
However, the implementation takes two hours to compute.

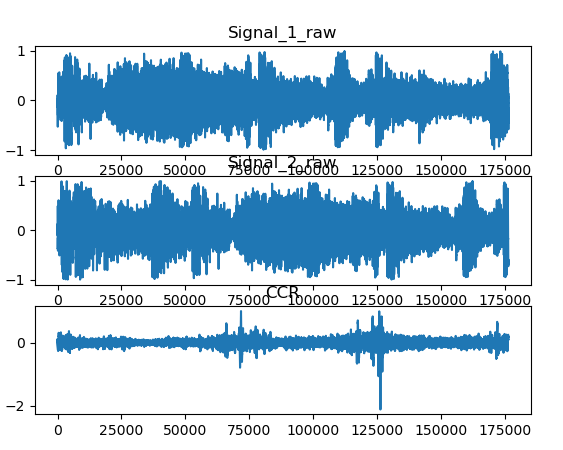


**Analysis - Spectral**

To improve performance, spectral cross-correlation was then employed to reduce the computation time. The code implements the spectral decomposition of the two signal files as outlined above, to determine the cross-correlation coefficient and associated lag. The high outlying high amplitude-low frequency noise and general sensor related noise is reduced when applying a low pass filter. The dominant frequencies are more easily apparent[[2]](#footnote-2) . As all the signal data is real valued, the related matrices are Hermitan. So the frequency spectrum is consequently symmetric, and may be mirrored or removed above the Nyquist frequency of Fsample/2 and filtered using a low pass filter.







The resulting values:

* Freq. = 44100
* Off-Set = -50081
* Off-Set Time = -1.136

Distance between two sensors = 378.16 meters

**Evaluation - Temporal**

For fast implementation, the pattern and template lists are converted into numpy arrays inside the *n\_corr* function. Notably implementation is very slow due to the use of for loops in the *n\_corr* and *calculate\_energy* functions. These enforce a minimum time complexity of N2, and greater. To reduce the calculation time, cached valued are used for the determination of the sum of the squared elements of the *pattern* array. Initially, the calculation was performed for every time the *calculate\_energy* function was called *i* many times[[3]](#footnote-3). Caching the value reduced the loop time for *n\_corr* by half. However, the calculation time was still .5 s per loop in *n\_corr­.* It was found when disabling *calculate\_scores*, the computation time was significantly reduced. This is likely due to the zeros that are multiplied each time the pattern and template are passed. An if statement is used to require only non-zero values are multiplied. This reduced the loop time again by half. The calculation was compared to the a library implementation of the cross correlation using *numpy.correlate*. The run time in comparison was 3.01s. The difference is firstly due to the optimisation of the numpy libraries, but also because the second method does not pad the second array. Thus, the convolution is faster. It may be possible to speed up the calculation time again by using Just in Time (JIT) decorators, like numba or PyPy.

These changes resulted in a 64% decrease in run time of the *n\_corr* loop, but at the cost of proper normalisation techniques. The offset was still calculated to be 389m but given a significant speed decrease, and the loss of proper normalization, this method was abandoned.

**Evaluation - Spectral**

To improve performance, spectral cross-correlation was then employed to decrease the run time. Spectral cross-correlation initially took 2.03 (min:sec) to perform. This technique should be much faster than temporal cross-correlation, so it was clear that something was impacting performance. After researching, it was found that numpy.fft uses the Cooley-Tukey algorithm to calculate the Fourier transform. As it is a divide and-conquer algorithm, it runs most optimally on inputs of a power of two in size, and least with sizes or prime numbers. The signal lengths are prime numbers, and so by padding these inputs with zeros to the closest power of two, the run time was significantly faster. After padding, the run time to find the distance between the two sensors was: 0.15s. The computation time compared to the Temporal cross correlation was noticeably faster. This is due to: Temporal cross correlation must repeat the same operations for each slice. Temporal cross correlation consequently also considers the offset of the neighbours of the slice. So the spectral cross correlation is thus faster, and need only be padded if signals are of different length.

|  |  |
| --- | --- |
| Spectral Cross correlation | Temporal Cross correlation |
| * Off-Set = -50081 * Off-Set Time = -1.136 * Distance between two sensors = 378.16 meters | * Off-Set = -50082 * Off-Set Time = -1.136 * Distance between two sensors = 378.17 meters |

**2. 2D pattern finding**

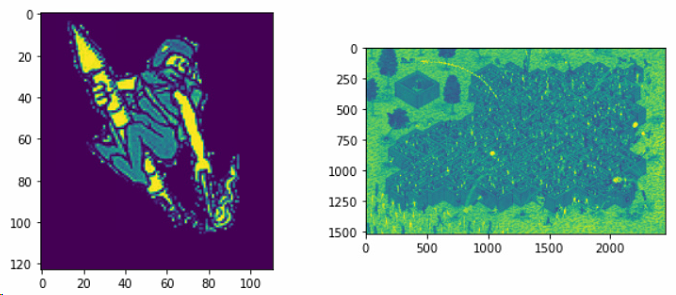
**Intro**

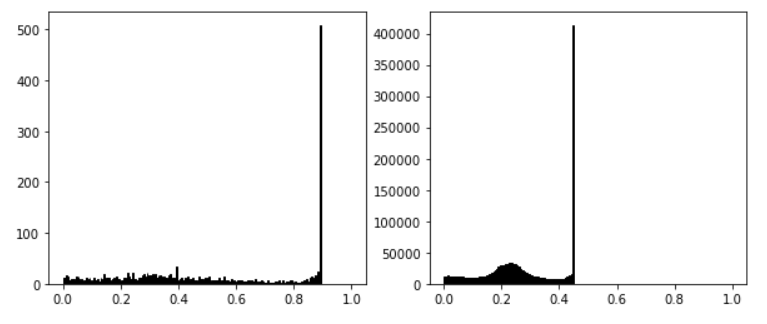
A simple way of determining relationships between images is by comparison of patterns with template data. In this experiment a method of pixel-based template matching via correlation calculation is used.

Test/Mean shift

Shifting the value of the pixel intensity spectrum has the effect of improving the cross-correlation algorithm accuracy and efficiency. In comparison of mean-shifted and non-mean shifted, the mean shifted speed and accuracy was found to perform better. This was apparent in the case of both the spatial cross correlation and spectral convolution approaches. In the case of the spatial correlation, the speed up was 40 seconds, whilst marginally more efficient 0.022s for the spectral approach. (figure). The effect of the zero-mean is to redistribute the intensity over a positive and negative range. In the case of a greyscale image, ranging from black to white, the dark regions of the images will now have a much weaker correlation to brighter regions of equal amplitude, but now opposite sign.

In this experiment, a “rocketman’ was searched for in a larger template image to investigate 2D cross correlation techniques.





**Application/Data**

The images that are used are formed by a matrix of pixels. The pattern and template image information is stored in tensor-like array of intensities over three colour channels, (RGB) and one transparency channel (A). Each channel list represents a pixel with three components of colour, stored at 8 bits[[4]](#footnote-4), ranging from 0 to 255. To determine the scalar cross correlation value, the images are converted into greyscale by finding the relative intensity of grey from a color table(reference). The functions m*atplotlib.image* and conert\_grey*()* rescale the UTF-8 data from each channel to float32 values between 0.0 and 1.0. The weights used in the conversion are calibrated for contemporary CRT phosphors: I = 0.2125 [R] + 0.7154 [G] + 0.0721 [B].

**Spatial - Analysis**

The approach to find the location of a pattern image inside a larger target image can be implemented by extending the method of the 1D spatial correlation. the pattern is moved one pixel at a time over a larger image. The output pixel value is the weighted sum of the input pixels within the window where the weights are the values of the cross-correlation assigned to every pixel of the window itself. The window with its weights is called the convolution kernel. For each shift position, the cross-correlation coefficient is calculated pixel by pixel between the small image and the correspondingly sized region of the larger image.

In a similar way to the 1D spatial convolution, the template image is padded at either end of its coordinated axes with the length of the pattern -1. The cross correlation is then found by *n\_corr2d* by sliding the pattern across the padded template and finding the sum product of every instance of the overlap 2D arrays over the non-padded region of the template image.

The *find\_offset* function will then return the coordinated which corresponds to the largest correlation value given by the position of the top left corner of the pattern image array.

The pattern image is found within the template image with top left corner coordinate of (x,y) = (529,983).

The first technique employed was spatial cross-correlation. The smaller pattern image was passed over the template image and a correlation value determine for each window overlap position.

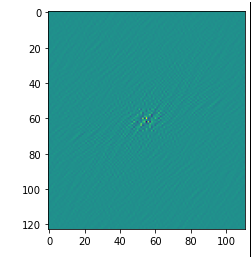
**Spectrum - Analysis**

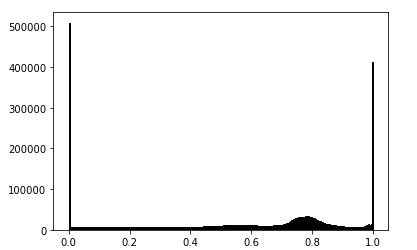
For simple cross correlation, the Fourier transform procedure is as follows. The images are read and converted into greyscale as in the spatial cross correlation script. Within *ccr\_2d* the smaller pattern image is padded with zeros at the bottom and right sides to reshape it to the size of the larger template image. The FFT is then applied to both images, along rows then columns in this way twice to transform information about the neighbour elements as well as the frequency. Thus, a full spatial description in the 2D array of the image is found for any given channel. The phase information is removed by taking the complex conjugate of one of the transformed matrices before evaluating their product. The transformed arrays are multiplied elementwise. Last, the inverse Fourier transform is taken to return the convolution of the two images in the spatial domain. This process is much faster than doing the unnormalized correlation in the spatial domain.

Fourier transform of f(t), Fourier transform of g(t), element-wise multiplication, Inverse Fourier transform of the result. For this reason, using spectral cross correlation should dramatically cut down running time of the python programs that will be created.

Where F is the Fourier transform and \* denotes the complex conjugate. By taking the complex conjugate of one of the inputs we are essentially performing the time reversal of that input, thus the output functions would now be the same as for convolution.

* https://imagemagick.org/docs/AcceleratedTemplateMatchingUsingLocalStatisticsAndFourierTransforms.pdf





Because the image in the Fourier domain is decomposed into its sinusoidal components, it is easy to examine or process certain frequencies of the image, thus influencing the geometric structure in the spatial domain. • In most implementations the Fourier image is shifted in such a way that the DC-value (i.e. the image mean) F(0,0) is displayed in the center of the image. The further away from the center an image point is, the higher is its corresponding frequency.

**Spatial - Evaluation**

* Typically, normalised cross correlation of for image searching is computational slow. The calculation time for this implementation was 40.2s.
* Normalized cross correlation, as described by equation (1), is computationally intensive and slow. Part of the complexity has to do with evaluating the numerator correlation in the spatial domain when the template image is large. The other aspect that adds to the complexity is the computation of the mean and standard deviation of each subsection of the larger image.
* Improvements may be made by using more robust algorithms which take statistical samples of the template image region to reduce the size of the search region. [[5]](#footnote-5)

**Spectral - Evaluation**

The method is much faster than spatial correlation since the number of operations is N^2logN not N^4. This is because of the implementation of the FFT. If the DFT were used, it would require a double sum over the indices for each offset position. The nested for loops would be as below:

def explicit\_correlation(image, kernel): This is O(n^4)

    hi, wi= image.shape

    hk, wk = kernel.shape

    image\_padded = np.zeros(shape=(hi + hk - 1, wi + wk - 1))

    image\_padded[hk//2:-hk//2, wk//2:-wk//2] = image

    out = np.zeros(shape=image.shape)

    for row in range(hi):

        for col in range(wi):

            for i in range(hk):

                for j in range(wk):

                    out[row, col] += image\_padded[row + i, col + j]\*kernel[i, j]

    return out

* This is for R G B colour images, so will take FFT three times. Above algorithm is for a single channel.
  + - * Frequency spectrum features

3. Conclusions

A 'rocketman' was located within a complex image to demonstrate the accuracy of these techniques and the speed of spectral cross-correlation. This can be seen in figure 5. . Both spatial and spectral cross-correlation correctly located the rocketman within the search area, but there was a drastic difference in runtime. The spectral cross correlation took less than one second. As both techniques produced identical results, spectral crose-correlation was chosen for all remaining alignment tasks in this document.

1. http://dspace.calstate.edu/bitstream/handle/10211.3/190191/GonzalezAdrian\_Thesis2017.pdf?sequence=1 [↑](#footnote-ref-1)
2. R. N. Bracewell and R. N. Bracewell, The Fourier transform and its applications [↑](#footnote-ref-2)
3. i many times to iterate over the size of the correlation array, = template\_padded – pattern [↑](#footnote-ref-3)
4. Human eye purportedly can only see 8 bits of colour. Reference [↑](#footnote-ref-4)
5. <https://imagemagick.org/docs/AcceleratedTemplateMatchingUsingLocalStatisticsAndFourierTransforms.pdf> [↑](#footnote-ref-5)