MA201 Project Report

Edge Detection In Images Using Fourier Transform

Anuj Buch	Haikoo Khandor	Ksheer Agrawal	Madhav Kanda	Sanskar Sharma
20110019	20110071	20110098	20110104	20110185

Group “4ier”| 14 November, 2021

Problem Statement

Edge detection is a technique of image processing used to identify points in a digital image with discontinuities, or say, sharp changes in the image brightness. These particular points where the image brightness ranges sharply are called the edges (or boundaries) of the image. It is one of the elementary steps in image processing, pattern recognition in images and computer vision.

The process of edge detection significantly decreases the amount of data and filters out unnecessary information, while conserving the important structural characteristics of an image. In order to achieve this, a fourier transform is used here. Though the latter is used on signals, it has the same effect on images. It converts the input image from spatial (space) domain to the frequency domain. Hence after applying the fourier transform, we get a series of low and high frequencies after plotting the image on a graph. Detecting edges is possible because the edges are made up of high frequencies. On taking the fourier transform of an image and applying a fourier inverse transform, we see that now the image consists only of the object boundaries. Also, here we will use Fast Fourier Transform i.e. FFT. As a traditional approach to this problem, we use DFT (Discrete Fourier Transform) whose time complexity is O(n^2). FFT is better than DFT as it has a time complexity of O(Nlog(N)). Our approach to this problem statement is to use python and employ numpy, matplotlib and other libraries in order to implement a FFT function. We also employ filters here. Filters applied on signals help in allowing specific bands of frequencies to pass through which can be later utilized as per need. Diving deep into the wide applications of the resulting image, we shall detect the edges using a high pass band filter which allows only high frequencies to pass through it. This helps us to detect the edges in the transformed image.

Edge detection is a comprehensive application that finds use in almost everything around us. The documents that one scans using scanner apps utilize this concept. From capturing and verifying fingerprints on digital screens to vision used in robots and the field of medicine, edge detection finds a wide range of use. This project aims to develop an FFT function and high pass filter to implement images and show the final results. The project work may be helpful in fingerprint detection and number plate detection and finds huge applications in fields requiring outline detection like architecture, surgery, etc.

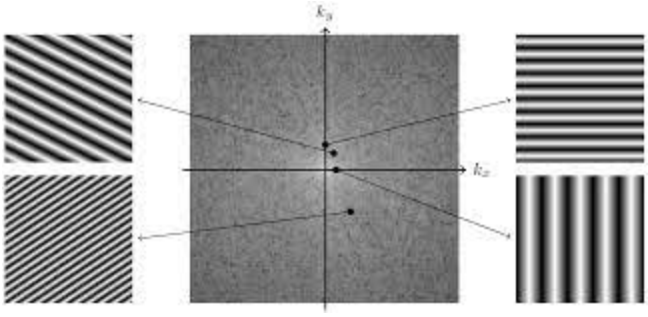
Contents

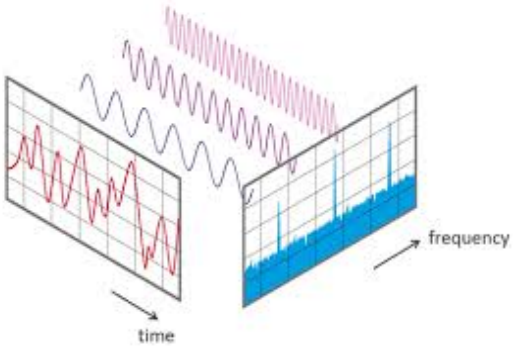
1. Physical Model

First, let us look at what transform means. Transform performs a transformation in the form of an operation such that the user can use it to solve several kinds of equations and tackle various types of functions. These can be of several kinds, depending on their application. Laplace transform, Fourier transform, etc., are some examples. Several of these have become of utmost importance due to their extensive usage in solving real-world scenarios. One such is the Fourier transform. It can be used to solve differential equations, design and analyze electrical circuits, signal analysis, signal processes, filtering, etc.

Fourier transform is a mathematical transform used to dismantle composite functions in space or time domain to functions depending on frequency, be it spatial or temporal. Signals, generally, build up several disturbances and changes as they travel. This causes them to take such complex waveforms, which are extremely difficult to analyze. Figuring out the constituents of the original wave is a next to impossible task even after noise reduction. The Fourier transform plays a key role here. In a layman's language, once a signal passes through the Fourier transformer, it separates it into its constituent frequency components. Hence, one can quickly identify the constituent frequencies of any signal by such a method. It converts signals from the time domain to the frequency domain.

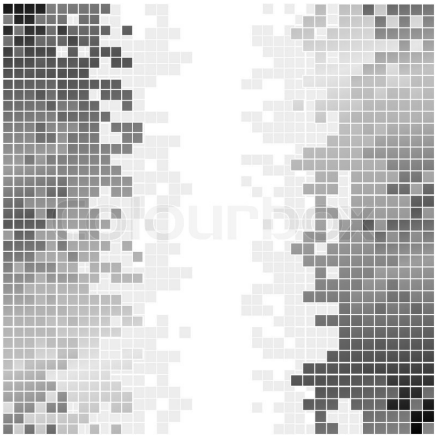
Analogous to signals, Fourier transform does the same thing to an image. It is better to compare and evaluate frequencies rather than the entire image. All said and done, every image has a unique Fourier transform, just as humans have their unique fingerprints.





2. Assumptions

1. For a digital image, we consider *x,y* (spatial coordinates) and the intensity values of *f* as finite discrete quantities.
2. We may see some artifacts - ripple-like structures called ringing effects if we use a rectangular window for masking. Rectangular masks are converted into sinc shape that causes this problem.
3. We visually analyze a Fourier transform by computing a Fourier spectrum (the magnitude of F(u,v)) and display it as an image.
4. Size of the input array is the same as the output array..
5. For a non-periodic function *f(x)* representation by Fourier integral, we assume *L,* as arbitrarily large, but finite.
6. Since the computer has limited space to compute the Fourier integral, we replace the infinity by number a.
7. We assume N to be sufficiently large to avoid aliasing.



3. Governing Equations

This section discusses the various concepts and methods used to perform this process, which acts as some pre-requisite for the same.

3.1 Fourier Series

The Fourier series represents a periodic function, *f(x)*, in terms of its sine and cosine constituents by an infinite series.

$$f(x) = a_0 + \sum_{n=1}^{\infty} (a_n \cos nx + b_n \sin nx). \tag{1}$$

$$a_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\, dx$$

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos nx\, dx \qquad n = 1, 2, \cdots$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin nx\, dx \qquad n = 1, 2, \cdots. \tag{2}$$

3.2 Fourier Integral

Fourier integral is a particular form of Fourier series for piecewise continuous functions. It evaluates integral as against series in the earlier case. The function, *f(x)*, required the input to have a right-hand as well as a left-hand derivative at each point. Moreover, for those points where it does not, it is the average of the two limits at that point. This form of solution further enhances the ease of the solution.

$$f(x) = \int_0^{\infty} [A(w) \cos wx + B(w) \sin wx]\, dw.$$

$$A(w) = \frac{1}{\pi} \int_{-\infty}^{\infty} f(v) \cos wv\, dv, \qquad B(w) = \frac{1}{\pi} \int_{-\infty}^{\infty} f(v) \sin wv\, dv$$

3.3 Fourier Transform

Integral transform produces new functions based on different variables. These act as the most accessible tool to evaluate the solutions to a variety of problems. This is easy to solve in the complex form, but can also be solved using sine and cosine integrals. Fourier transform of a function *f(x)* is given by $\hat{f}(w)$ and is evaluated as follows:

$$\hat{f}(w) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-iwx}\, dx.$$

3.4 Convolution

The convolution $f * g$ of functions f and g is defined by

$$g(x) * f(x) = \int_{-\infty}^{\infty} f(t) g(x-t) dt$$

Suppose that *f(x)* and *g(x)* are piecewise continuous, bounded, and absolutely integrable on the x-axis. Then

$$\mathcal{F}(f * g) = \sqrt{2\pi}\, \mathcal{F}(f)\, \mathcal{F}(g). \tag{5}$$

3.5 Discrete Fourier Transform

The general idea is that the image (f(x,y) of size M x N) will be represented in the frequency domain (F(u,v)). The equation for the two-dimensional discrete Fourier transform (DFT) is:

$$F(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)e^{-j2\pi(x\frac{m}{M}+y\frac{n}{N})} \tag{6}$$

The concept behind the Fourier transform is that any waveform can be constructed using a sum of sine and cosine waves of different frequencies. The exponential in the above formula can be expanded into sines and cosines with the variables u and v determining these frequencies.

3.6 Fast Fourier Transform

A Fast fourier transform is preferred over the Discrete Fourier Transform method because it's computational method for the DFT needs only $O(Nlog_2N)$ operations instead of O(N 2).It makes the DFT a practical tool for large N.

3.7 Inverse Fourier Transform

The inverse of the above Discrete Fourier transform is given by the following equation:

$$f(m,n) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(x,y)e^{j2\pi(x\frac{m}{M}+y\frac{n}{N})} \tag{7}$$

Thus, if we have $F(u,v)$, we can obtain the corresponding image $(f(x,y))$ using the inverse, discrete Fourier transform.

4. Parameters

4.1 DFT and FFT Parameters

- $N = number\ of\ samples\ that\ in\ the\ given\ signal$
- $n = current\ sample$
- $k = the\ index\ of\ the\ term\ in\ the\ summation$
- $x_n = the\ sine\ value\ at\ the\ sample\ n$
- $X_k = The\ final\ summation\ value$
- $W_N^{ux} = A\ method\ of\ representation\ through\ the\ variable\ W\ whose\ numerical\ value\ is\ e^{-j\frac{2\pi}{N}xu} i.e. W_N^{ux}. It\ is\ just\ a\ way\ of\ representation\ in\ place\ of\ having\ to\ write\ W_N^{ux} = e^{-j\frac{2\pi}{N}xu} the\ exponential\ function\ every\ now\ and\ then.$
- $f(x) = The\ function\ on\ which\ the\ Fourier\ transform\ is\ applied.$
- $F(x) = The\ Fourier\ transform\ of\ the\ function\ f(x).$

4.2 IDFT Parameters

- $x = Discrete\ signal$
- $X_k = The\ final\ summation\ value\ of\ the\ DFT$
- $n = current\ sample$
- $k = the\ index\ of\ the\ term\ in\ the\ summation$

4.3 Other Parameters

Other parameters used in edge detection of the image are given below.

1. A1 = 20
2. A2 = 2000
3. mask = 12, where mask represents the unit length of transformed image masked with 0.
4. img is read as a numpy array of intensity of input image in grayscale.

5. Solution Methodology

To produce a line “drawing” of a scene from an image of that scene, we use the following methodology -

We define the image as a 2-d function *f(x,y)* where *x* and *y* are spatial coordinates and the amplitude of *f* at any pair of coordinates *(x,y)* and the intensity values of the image at that point. For a sinusoidal signal, if the amplitude varies so fast in a short time, you can say it is a high frequency signal. Edges are significant local changes of intensity in an image. More intuitively, at edges the amplitude varies drastically in images. So we can say, edges are high frequency content in images.

Once we perform the fourier transform of an image, we would see a plot of high and low frequencies. Interestingly, all the low frequency components are saturated at the center, and the high frequency components are scattered around. We then create a high pass filter which would typically be a mask array of the same size as the image with a miniature square of zeroes at the center and rest all ones. Now when the mask is applied to the original image, the resultant would only have high frequencies. This becomes quite useful as low frequencies correspond to non-edges in the spatial domain. The result shows High Pass Filtering is an edge detection operation.

6. Important Concepts:

6.1 DFT

The Discrete Fourier Transform(DFT for short) is a transform that can be used to transform a finite-length, discrete signals of equally spaced signals into a complex-valued function of frequency. It tells us information about the frequency of different sine waves that are summed up to get that particular signal.

We use the Euler formula to relate the exponential function with the trigonometric function.

Euler Formula: $e^{i\phi} = \cos\phi + i\sin\phi$

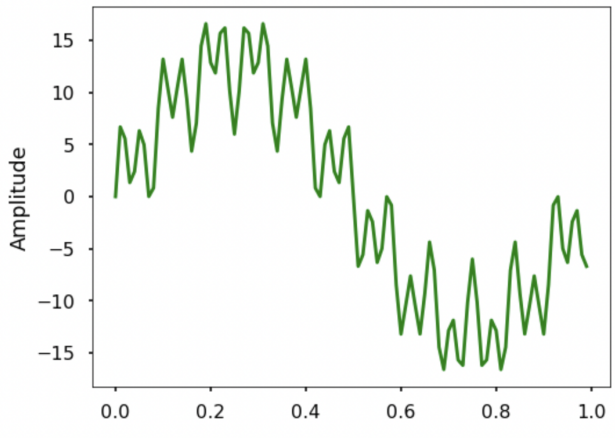
Formula for Discrete Fourier Transform:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N} = \sum_{n=0}^{N-1} x_n[\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)]$$

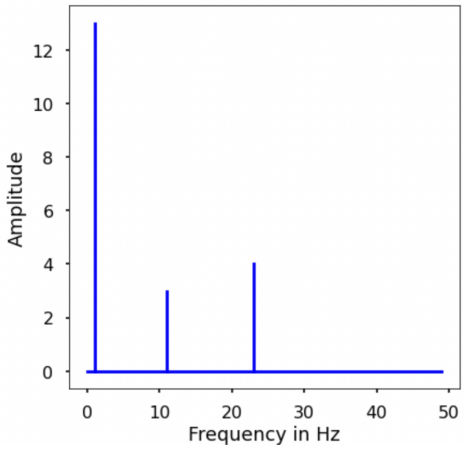
Parameters:

- $N = number\ of\ samples\ that\ in\ the\ given\ signal$
- $n = current\ sample$
- $k = the\ frequency\ of\ the\ term\ in\ the\ summation$
- $x_n = the\ sine\ value\ at\ the\ sample\ n$
- $X_k = The\ final\ summation\ value$

Problem statement: To make a function that calculates the DFT of a signal given as a 1-D input to the function. This function shall return the computer DFT values back as the output.



Signal constituting of 3 different waveforms



After applying DFT

Limitations of DFT: The main issue with the above DFT implementation is that it is not efficient when operated on dense signals with a large number of data points. The algorithm used to implement DFT is of order N^2. To speed up computation, a faster version of DFT, "Fast Fourier Transform" is used.

Checking the time taken by FFT and DFT:

```
starttime_fft=time.time()
F_fft=FFT(np.array(f))
endtime_fft=time.time()
elapsed_fft=endtime_fft-starttime_fft
print("Time taken to run the FFT Algorithm: "+str(elapsed_fft)+" sec.")

starttime_dft=time.time()
F_dft=DFT1D(np.array(f))
endtime_dft=time.time()
elapsed_dft=endtime_dft-starttime_dft
print("Time taken to run the DFT Algorithm: "+str(elapsed_dft)+" sec.")
```

Result:

```
Time taken to run the FFT Algorithm: 0.000568389892578125 sec.
Time taken to run the DFT Algorithm: 0.0013747215270996094 sec.
```

6.2 FFT

Motivation: DFT (Discrete Fourier Transform) is a method through which a finite sequence of equally spaced samples is converted into a complex valued function of frequency. However, in case of large data samples, it is not an efficient algorithm in terms of time complexity. Since it is

of O (n^2), we cannot use it for practical purposes. Hence there is a need to define another function. Here comes Fast Fourier Transform. It effectively solves the problem by using recursion and does it in O(Nlog(N)). (N Is the data size)

About FFT: Fast Fourier Theorem uses DFT and recursively iterates through two halves of the array which it takes as an input. The array is split into two one consisting of even indices and the other of odd indices. This process keeps on repeating till we reach the base case. Here, it simply returns the value and keeps on returning till we get the whole array.

We also use **Euler's famous** complex formula

$e^{ix} = cos(x) + isin(x)$

Formula used in computing DFT:

$$X_k = \sum_{n=0}^{N-1} x_n . e^{\frac{-i2\pi kn}{N}} = \sum_{n=0}^{N-1} x_n . [\cos\left(\frac{2\pi}{N} kn\right) - i . \sin\left(\frac{2\pi}{N} kn\right)]$$

Formula used in computing FFT:

$$F(u) = \sum_{x=0}^{N/2-1} f(2x) . W_{N/2}^{ux} + W_N^u \sum_{x=0}^{N/2-1} f(2x+1) W_{N/2}^{ux}$$

The first term is the DFT of the N/2 elements corresponding to the even indices, the second term is the DFT of the N/2 elements related to the odd indices.
This way, we can split the DFT of N elements, in a recursive way, into two N/2 DFTs, and later combine the results:

Parameters:

- $N = number\ of\ samples\ that\ in\ the\ given\ signal$
- $n = current\ sample$
- $k = the\ frequency\ of\ the\ term\ in\ the\ summation$
- $x_n = the\ sine\ value\ at\ the\ sample\ n$
- $X_k = The\ final\ summation\ value$
- $W_N^{ux} = A\ method\ of\ representation\ through\ the\ variable\ W\ whose\ numerical\ value\ is\ e^{-j\frac{2\pi}{N}xu} i.e.\ W_N^{ux}.\ It\ is\ just\ a\ way\ of\ representation\ in\ place\ of\ having\ to\ write\ W_N^{ux} = e^{-j\frac{2\pi}{N}xu} the\ exponential\ function\ every\ now\ and\ then.$
- $f(x) = The\ function\ on\ which\ the\ Fourier\ transform\ is\ applied.$
- $F(x) = The\ Fourier\ transform\ of\ the\ function\ f(x).$

6.3 Inverse Discrete Fourier transform (IDFT)

Given a discrete Fourier transform X: $Z \to C$ of a discrete signal. The inverse Discrete Fourier Transform of X is defined as the signal X : $[0,\ N\ -\ 1] \to C$ with components x(n) given by the expression

$$x(n) := \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(k) e^{j2\pi kn/N} = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(k) \exp(j2\pi kn/N)$$

Theorem

Given a discrete signal x:[0, N − 1] → C, let X = F(x): Z → C stand in for the DFT of x and $\bar{x}$ = F^{-1} (X) : [0, N − 1] → C be the Inverse Discrete Fourier transform of X. We then have that x ≡ $\bar{x}$, or, equivalently,

$$F^{-1}\ (F\ (x)) = x$$

Proof

Given a discrete signal x: $[0,\ N\ -\ 1]\rightarrow C$, let $X\ =\ F(x)$: $Z\rightarrow C$ be the DFT of x and $\bar{x}\ =\ F^{-1}\ (X)\ :\ [0,\ N\ -\ 1]\rightarrow C$ be the Inverse Discrete Fourier transform of X. From the definition of the Inverse Discrete Fourier transform, we have

$$\tilde{x}(\tilde{n}) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(k) e^{j2\pi k\tilde{n}/N}$$

Substituting the definition of the DFT for X(k) in the above equation

$$\tilde{x}(\tilde{n}) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \left(\frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \right) e^{j2\pi k\tilde{n}/N}$$

We may exchange the order of the summation, so that we first sum over k , and then pull out $x(n)$ since it is independent of k, i.e.

(A)---------- $$\tilde{x}(\tilde{n}) = \sum_{n=0}^{N-1} x(n) \left(\sum_{k=0}^{N-1} \frac{1}{\sqrt{N}} e^{-j2\pi kn/N} \frac{1}{\sqrt{N}} e^{j2\pi k\tilde{n}/N} \right)$$

Due to the orthonormality we obtain:

$$\sum_{k=0}^{N-1} \frac{1}{\sqrt{N}} e^{-j2\pi kn/N} \frac{1}{\sqrt{N}} e^{j2\pi k\tilde{n}/N} = \delta(n - \tilde{n})$$

Therefore by the above result **(A)** reduces to:

$$\tilde{x}(\tilde{n}) = \sum_{n=0}^{N-1} x(n)\delta(n - \tilde{n}) = x(n)$$

With $\tilde{n} = n$ since the only nonnegative term in the sum is when $\tilde{n} = n$

Code:

```
def idft(y_real, y_imaginary):
  N, x = len(y_real), []
  for n in range(N):
    real = 0
    for k in range(N):
      theta = k * (2 * math.pi) * (float(n) / N)
      real += (y_real[k] * math.cos(theta)) - (y_imaginary[k] *
math.sin(theta))
    x.append(real)
  return x
```

Parameters:

1. y_real = Array containing real part of the function on which we have to apply Inverse Discrete Fourier Transform
2. y_imaginary = Array containing imaginary part of the function on which we have to apply Inverse Discrete Fourier Transform
3. N = length of array of y_real
4. x = Array containing real part of IDFT implemented function
5. theta = angle of exponential function in the Inverse Discrete Fourier Transform

Explanation of Code

- Above we have defined a function which takes in the input y_real and y_imaginary.
- We define N as the length of the y_real array and x as an empty array.
- We initiate a for loop in which we initialize a variable named real, which we initialize as 0 and run it N number of times so as to calculate for each of the functions in y_real.
- We then initialize a for loop which runs for N times, for calculating the value of theta and real part of the inverse discrete fourier transform.
- After each iteration of the loop we append to x the real part of the inverse fourier transform of the function to store the value.

7. Algorithms Implemented

A Python Code was written to solve the problem statement. The algorithm followed can be stated in the following steps:

Writing a DFT function implementation in Python:

1. We receive a 1-D array as input for the DFT function. This 1-D array corresponds to the discrete, complex signal that we want to apply the DFT to.
2. For each frequency term, we apply the formula:
$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i 2\pi kn/N} = \sum_{n=0}^{N-1} x_n [\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)]$$
3. For each frequency we choose, we must multiply each signal value by a complex number and add together the results.

Code:

```
def DFT(f):
    N = len(f)
    # we are creating a empty array of 0s of size N
    F = np.zeros(N).astype(np.complex64)

    # creating indices for x, allowing to compute the multiplication using
numpy (f*exp)
    x = np.arange(N)


    # for each frequency 'u', perform vectorial multiplication and sum
    # doing 4N - 2 operations N times → O(n^2) complexity
    for freq in np.arange(N):
      F[freq] = np.sum(f*np.exp((-1j*2*np.pi*freq*x)/N))

    return dft_F
```

Time complexity analysis:
For each real-complex multiplication, we need to compute two real multiplications, thus needing $2N$ multiplications. Total number of operations = $2N$(for multiplication) + $2(N-1)$ (for addition of N-1 terms) = $4N-2$. To perform this for all of the N frequencies, total number of computations reach $N(4N-2)$ thus making it a N^2algorithm.

Writing a FFT function implementation in Python:

The Fast Fourier Transform (FFT) is a divide and conquer algorithm that recursively splits the input array in two parts; one for the odd indices while another for the even indices, until the trivial case is achieved. The trivial case is it returns the integer back.

It is important to note that complex exponentials (that can be decomposed into a sum of sine and cosine by Euler's equation) are periodic and symmetric, and from those properties the FFT is defined.

In particular, from $e^{-j\frac{2\pi}{N}xu}$, we isolate the constant term, and define it as a variable: W = $e^{-j\frac{2\pi}{N}}$.

Note W is constant because it does not depend on the time sampling (controlled by x), nor depends on the frequencies (u). $W_N^{ux} = e^{-j\frac{2\pi}{N}xu}$

For example, for a signal with 4 observations, i.e., N = 4: W = (6.123233995736766e-17-1j).

This value does not depend on u nor x. The two properties we are going to use to implement the FFT are:

1. Periodicity in u,x: $W_N^{ux} = W_N^{u(N+x)} = W_N^{(u+N)x}$.
2. Symmetry of the complex conjugates: $W_N^{u(N-x)} = W_N^{-ux} = (W_N^{ux})^*$ for example this is easy to see for x=N, $W_N^{uN} = e^{-j2\Pi u} = 1$.

Now we define the **division** step of the algorithm. This is done by decomposing the transform into even and odd indices of x. To avoid a cluttered notation, let us express the transform in terms of the variable W :

$$F(u) = \Sigma_{x=0}^{N-1} f(x) W_N^{ux}$$

Now we write the function of evaluating the even indices 2x and the odd indices 2x+1:

$$F(u) = \Sigma_{x=0}^{N/2-1} f(2x) W_N^{u(2x)} + \Sigma_{x=0}^{N/2-1} f(2x+1) W_N^{u(2x+1)}$$

Note 2x forms the sequence 0, 2, 4, 6, while 2x+1 the sequence 1, 3, 5, 7 as we wanted, therefore:

$$F(u) = \Sigma_{x=0}^{N/2-1} f(2x) (W_N^2)^{ux} + \Sigma_{x=0}^{N/2-1} f(2x+1) (W_N^2)^{(x+\frac{1}{2})u}$$

Let us manipulate this sum, isolating the terms that are independent of x, which is the W_N^u :

$$F(u) = \Sigma_{x=0}^{N/2-1} f(2x) (W_N^2)^{ux} + W_N^u \Sigma_{x=0}^{N/2-1} f(2x+1) (W_N^2)^{ux}$$

But we know that $W_N^2 = e^{-j\frac{2\Pi}{N}2} = e^{-j\frac{2\Pi}{N/2}} = W_{N/2}$

and this defines the 'trick', since it allows to write the transform as:

$$F(u) = \Sigma_{x=0}^{N/2-1} f(2x) W_{N/2}^{ux} + W_N^u \Sigma_{x=0}^{N/2-1} f(2x+1) W_{N/2}^{ux}$$

The first term is the DFT of the N/2 elements corresponding to the even indices, the second term is the DFT of the N/2 elements related to the odd indices.

This way, we can split the DFT of N elements, in a recursive way, into two N/2 DFTs, and later combine the results:

$$F(u) = F_{even}(u) + W_N^u . F_{odd}(u)$$

Recall the property of symmetry of the complex conjugate:

$$F(u + N/2) = F_{even}(u) - W_N^u . F_{odd}(u)$$

In this simple example, we partition the elements until we reach the base case, that is when there is only 1 even and 1 odd element, allowing us to compute.

Now this result is stored and we execute the other 'side' of the recursion, relative to the first odd indices

Now, combining the individual results (recurseeven and recurseodd):

```
recurseodd0 = f_odd_even[0] + exp(-2j*pi*0/N) * f_odd_odd[0]
recurseodd1 = f_odd_even[0] - exp(-2j*pi*0/N) * f_odd_odd[0]
recurseodd = [recurseodd0, recurseodd1]
recurseeven0 = f_even_even[0] + exp(-2j*pi*0/N) * f_even_odd[0]
recurseeven1 = f_even_even[0] - exp(-2j*pi*0/N) * f_even_odd[0]


recurseeven = [recurseeven0, recurseeven1]
```

Let us code a function for this algorithm

```
def FFT(f):
    N = len(f)
if (N == 0 or N == 1):
        return f
    # taking the elements at odd and even position separately
    evenarr=FFT(f[0::2])
    oddarr=FFT(f[1::2])
    # stores the final array containing the array after FFT is used
    ans = np.zeros(N).astype(np.complex64)
    # only required to compute for half the frequencies
    # since u+N/2 can be obtained from the symmetry property
    for u in range(N//2):
        ans[u]=evenarr[u]+exp(-2j*pi*u/N)*oddarr[u] # conquer
        ans[u+N//2]=evenarr[u]-exp(-2j*pi*u/N)*oddarr[u]  # conquer
    return ans
```

Time complexity analysis:

The time complexity of FFT is $N * log(N)$. This improves upon the DFT's running time of N^2.

8. Results and Discussion

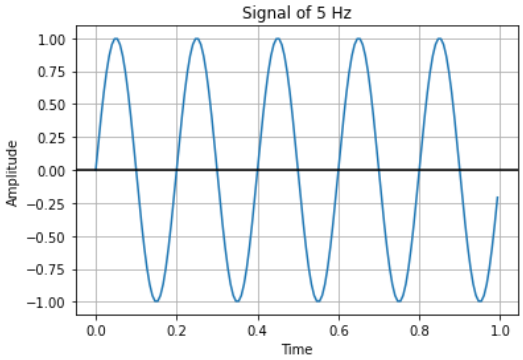


Figure 1: Amplitude Vs Time

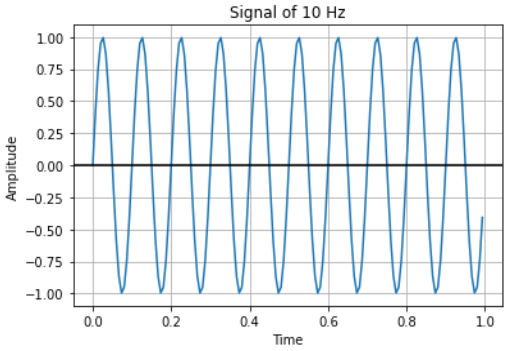


Figure 2: Amplitude Vs Time

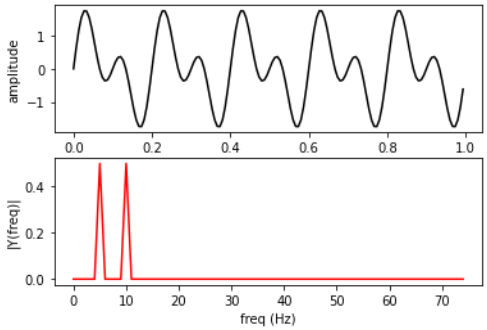


Figure 3: Amplitude & Frequency Plot

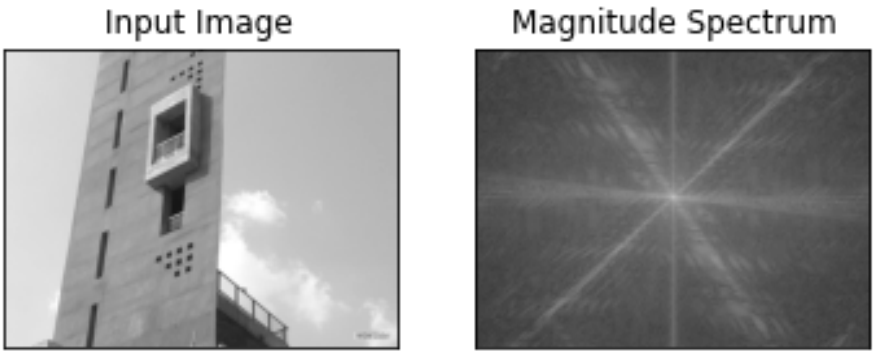


Figure 3: Fourier Transform of Input Image in GrayScale

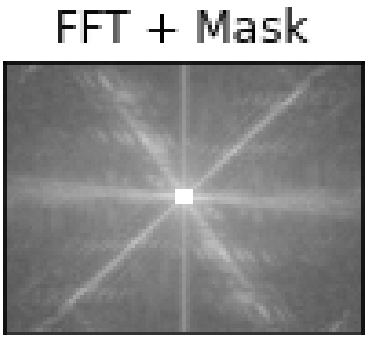


Figure 4: After Applying HPF to FFT Transformed Image

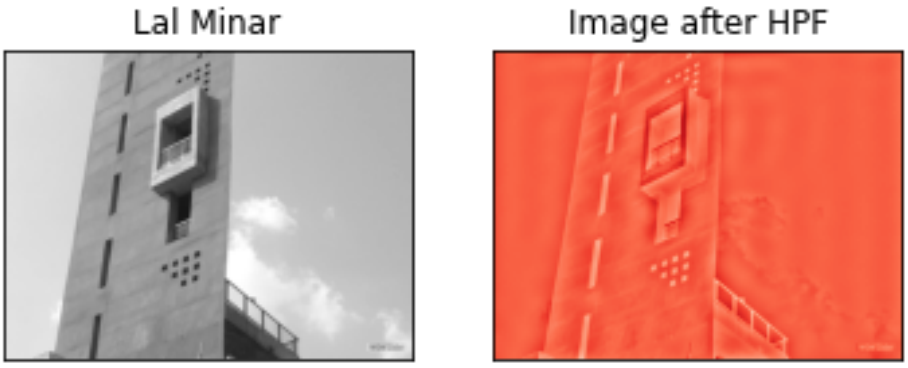


Figure 5: Resultant of HPF - Edge Detection

Results

- Here as we can see, a high pass filter sharpens(or shows the edge of) an image. It attenuates the low frequencies and leaves the high frequencies of the Fourier Transform relatively unchanged.
- A Fast fourier transform is preferred over the Discrete Fourier Transform method because it's computational method for the DFT needs only $O(N)log_2N$ operations instead of O(N 2) . It makes the DFT a practical tool for large N.
- $F(0,0)$ is equal to MN times the average value of $f(x,y)$.
- The fourier spectrum is symmetric about the origin.
- Adding zero padding to the image helps to create a finer sampling of the fourier transform faster computation of FFT.
- The zero frequency components are displayed in the top left and therefore need to be shifted in the center using the fftshift function.
- In the Fourier transform, the high peaks are so high that they hide the details. We reduce the contrast with the log function.
- Fourier transform leads to complex parts. Both the magnitude and the phase functions are necessary for the complete restoration of an image from its Fourier transform.
- Multiplication of two Fourier transforms corresponds to convolution of the associated functions in spatial domains. Therefore simple multiplication of spatial filter and frequency domain speeds up the filtering process.

9. Future Scope

- Optimization of data for continuous feed into the program to further decrease the time required is what one can work upon. It shall require a parallel flow of data and detection.
- Automation of various processes requires excellent surveillance and observational abilities. Since machine technology mandates automation, image processes are of extreme use here. However, here, we need a combination of smooth as well as sharp images, which a blend of these filters shall achieve.
- The edge detection technology can be used to detect edges in fingerprints which can enhance the rate of detection of fingerprints as it would take less memory than a photo of a fingerprint.
- It is challenging to analyze the boundaries of an image viewed through a satellite as the color difference between two territories is not much. Thus, edge detection can help us identify territories quickly through satellite images as it produces sharper images.
- Edge detection is critical when doing some serious image processing, such as that for medical reasons. These require high precision and hence sharper features. Therefore, such processes can use high pass filters.

10. Code for Edge Detection

11.1 Fourier Transform of images using OpenCV

```
 #Importing numpy,cv2 & matplotlib
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt


# Reading image in grayscale
img = cv.imread('lal_minar.jpg',0)

#Frequency transform is a complex array
f = np.fft.fft2(img)

#Shifting the zero frequency component to the center
fshift = np.fft.fftshift(f)

#Finding the magnitude spectrum
A1 = 20;
magnitude_spectrum = A1*np.log(np.abs(fshift))

#Plotting
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

```
#Center

rows, cols = img.shape

crow,ccol = rows//2 , cols//2

#  HPF masking, center 12X12 grid masked 0, remaining all ones

mask = 12;

fshift[crow-mask:crow+mask, ccol-mask:ccol+mask] = 0

# Finding the magnitude spectrum of masked Fourier Transform

A2 = 2000;

fshift_mask_mag = A2 * np.log(np.abs(fshift))

#Restoring the original indexing

f_ishift = np.fft.ifftshift(fshift)

#Inverse FFT

img_back = np.fft.ifft2(f_ishift)

#Finding the magnitude spectrum

img_back = np.real(img_back)

#Plotting

plt.subplot(131),plt.imshow(img, cmap = 'gray')

plt.title('Lal Minar'), plt.xticks([]), plt.yticks([])

plt.subplot(132),plt.imshow(fshift_mask_mag , cmap = 'gray')

plt.title('FFT + Mask'), plt.xticks([]), plt.yticks([])

plt.subplot(133),plt.imshow(img_back, cmap = 'Reds')

plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])

plt.show()
```

11. References

[1] https://docs.opencv.org/3.4/de/dbc/tutorial_py_fourier_transform.html

[2] http://apps.usd.edu/coglab/schieber/psyc707/pdf/2D-FFT.pdf

[3] Kreyszig_Advanced Engineering Mathematics Book

[4] https://mathworld.wolfram.com/FastFourierTransform.html

12. Contribution

MA201 Maths project was a delightful experience for all of us as we learned to work together as a team. Understanding about the Fourier series and its application was very interesting and engaging as we were not aware of the extent of the application that Fourier series and its transform plays in our life. It is due to this concept that image and signal processing is possible today without which almost all electronics and digital media would not have been possible.

In the project, Anuj Buch handled and researched the Physical model, Governing Equations, Further Scopes and documented the final draft. Haikoo Khandor took the responsibility of FFT defining it in Parameters, explained it in detail in Important Concepts, coded and commented in Algorithm Implementation. Ksheer Sagar Agrawal assessed in Solution Methodology, Results and Discussion and implemented the Code for Edge Detection. Madhav Kanda explained IDFT in Parameters, Important Concepts, coded it and did Future Scopes. Sanskar Sharma played a crucial role in DFT in Parameters and Important Concepts and coded the DFT part in Algorithm Implementation.