Università
della
Svizzera
italiana

Faculty
of
Informatics

Bachelor Thesis

June 18, 2023

Spectrally Accurate Resampling of High Quality Rotated Images

Dylan Reid Ramelli

Abstract

The idea of rotating an image might seem a trivial thing and until the work on this project began it was thought so as well. It became clear that rotation is a multi faceted operation that involves different steps. First we must acquire the image and to do that we must be able to sense it. Just like a robot arm in an assembly line needs to know which parts of the car go where, we must be able to take a discrete sample from a continuous signal. Once the image is acquired we need tools that are as accurate as possible and immune to disturbances to be able to perform accurate rotations on the image. This project, which is based on a research paper on rotating images using a convolution based interpolation [2] will try to explore a similar way to rotate an image while maintaining a high degree of accuracy and performance. To do this the Fourier Transform's property of shift and of convolution of a signal with a filter kernel will be taken advantage of. What will also be shown is how the filter kernel was created in the frequency domain and the problems that the choice of filter might pose to the result of the shift.

Advisor
Prof Rolf Krause
Co-Advisors
Dr Diego Rossinelli, Dr Patrick Zulian

Advisor's approval (Prof Rolf Krause): Date:

Contents

1 Introduction, Motivation

For many years the amount of digital imaging data has been increasing exponentially (citation) and as such the need for acquiring this large data and process it with care has become a central point of focus. The goal of this project is to process high quality signals, in this case images, by rotating them using techniques such as the Fourier Transform, Shift Theorem, FIR filtering and CUDA parallel processing.

The accurate rotation of images is very useful in many applications such as Data Augmentation for example. For Data augmentation we want to increase the coverage of a certain dataset. In Convoluted Neural Networks we can augment the dataset by rotating the input images by some random rotation and then feeding them to the model to improve it without having to collect more data externally. Another application of this technique can be seen in Data Visualization for multi-modal imaging. Images are taken at different times or in different ways, such as in a PET-CT scan and rotation can enable us to re-align the elements present in these images. In this case we can either acquire images at different times and then combine them together so that we can manipulate the images by rotation for example to align them correctly. Go into more detail.. ask Diego.

2 State of the art

For certain applications such as... it is required to be able to rotate images while maintaining the highest quality possible. To achieve satisfying results the general approach is to use bi-linear and nearest neighbor interpolation.

Include images from Diego of blood vessels in the heart iI think

3 Methodology

In this project we will use the same method to rotate an image as in the original paper [2] except we will be creating our own convolution kernel in the frequency domain by using a cosine function as base. Defining the function in the Fourier space allows us to fine tune the filter to have a better control over the smoothing effect that will occur when we shift by fractional amounts. Since the kernel will be small we will also just use a normal Inverse Fourier Transform instead of a IFFT.
In the article that this project takes inspiration from it is shown that the traditional rotation matrix

$$R(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

can be factorized as three matrices each of which represents a shearing of the image in a cardinal direction.

$$R(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} = \begin{pmatrix} 1 & -tan\theta/2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ sin\theta & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & -tan\theta/2 \\ 0 & 1 \end{pmatrix}$$

The first matrix shears the image in the x direction by $\Delta_x = -y \cdot tan(\theta/2)$, the second matrix shears the image in the y direction by $\Delta_y = x \cdot sin(\theta)$ and the last matrix shears the image again in the x direction by Δ_x. In our case we can represent the image as a one dimensional array ordered row-wise.

3.1 Fourier Transform

The Fourier transform of a function $g(x)$ is defined as follows:

$$G_m = \sum_{k=0}^{N-1} g(k) e^{-i\frac{2\pi}{N}km} \tag{1}$$

To shift a signal by a fractional amount it is imperative to use the correct frequency when doing so.
Given that for n samples we have $\lfloor \frac{n}{2} \rfloor + 1$ distinct frequencies

3.2 Fourier Transform with Shift Theorem

Here we derive the shift theorem for a discrete signal starting from the normal Fourier Transform with g_k being our 1D input array, N the number of samples and m the frequency:

$$G_m = \sum_{k=0}^{N-1} g_k[k] e^{-i\frac{2\pi}{N}km}$$

Now instead of $g_k[k]$, we want $g_k[k-\delta]$ where δ is the amount we want to shift. So the above equation can be rewritten as:

$$Z_m = \sum_{k=0}^{N-1} g_k[k-\delta]e^{-i\frac{2\pi}{N}km}$$

$$Z_m = \sum_{r=0-\delta}^{N-1-\delta} g_k[r]e^{-i\frac{2\pi}{N}km}, r = k-\delta$$

Since $r = k-\delta$ then $k = r+\delta$, and as such:

$$Z_m = \sum_{r=-\delta}^{N-1-\delta} g_k[r]e^{-i\frac{2\pi}{N}(r+\delta)m}$$

We can then separate the exponential:

$$Z_m = \sum_{r=-\delta}^{N-1-\delta} g_k[r]e^{-i\frac{2\pi}{N}rm}e^{-i\frac{2\pi}{N}\delta m}$$

And factor it out of the sum:

$$Z_m = e^{-i\frac{2\pi}{N}\delta m}\sum_{r=-\delta}^{N-1-\delta} g_k[r]e^{-i\frac{2\pi}{N}rm}$$

The sum now has exactly the same range as before:

$$Z_m = e^{-i\frac{2\pi}{N}\delta m}\sum_{k=0}^{N-1} g_k[k]e^{-i\frac{2\pi}{N}km}$$

$$Z_m = e^{-i\frac{2\pi}{N}\delta m}G_m = H_m \cdot G_m \tag{2}$$

The above equation 2 works well for any kind of shift that we want to perform on our signal but for any fractional amount we encounter some problems with the use of the correct frequency indexes. Here is a graphical example that was found to be useful: With an $N = 3$ number of samples of a signal we have exactly 2 distinct frequencies **??** but the total frequencies are actually 3 and they are $[0+i0, \frac{1}{2}+i\frac{\sqrt{3}}{2}, \frac{1}{2}-i\frac{\sqrt{3}}{2}]$. The last two frequencies are one the complex conjugate of the other.

"Explain why".

To solve this we need to take into consideration the negative frequencies and in particular if the number of samples is odd or even. Normally we multiply each sample by its corresponding phase, which is based on the frequency number m but since we need to take into consideration the negative frequencies we can define a function called wavenum that returns the correct frequency index to use in the phase shift:

$$wave\_n(m) = (m + \lfloor N/2 \rfloor) \bmod N - \lfloor N/2 \rfloor$$

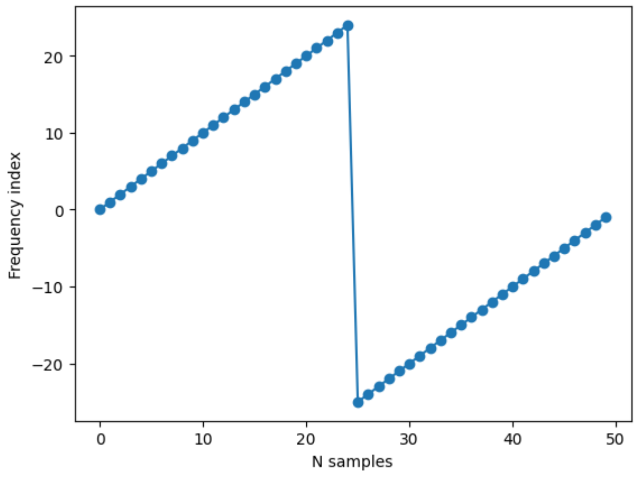


Figure 1. Wavenum function with $N = 50$

$$H_m = e^{-i\frac{2\pi}{N}\delta \text{wave\_}n(m)}$$

$$Z_m = H_{\text{wave\_}n(m)} \cdot G_m \tag{3}$$

3.3 Filter

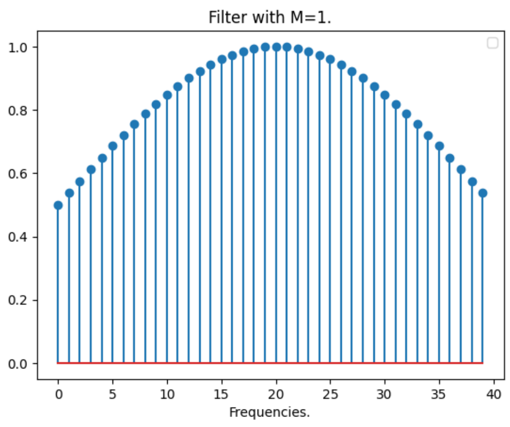
To start of we create the kernel by defining it in the Frequency domain with the use of the Continuous Fourier transform on a sinusoid function.

$$F(\omega) = \int_{-M}^{M} \cos(\frac{\pi}{M}t)\mathrm{e}^{-\mathrm{i}\omega t}, \mathrm{d}t \tag{4}$$

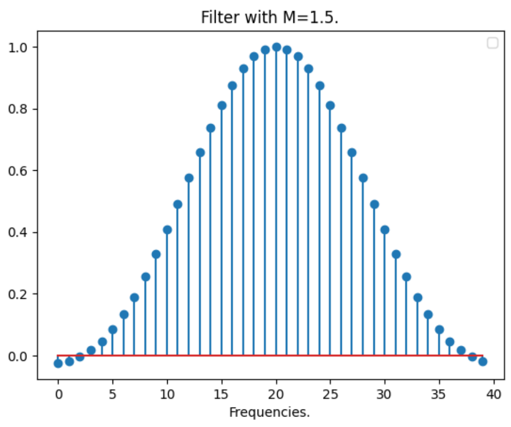
This equation allows us to define our CFT in an interval M, that we choose based on the fractional amount we want to shift, as to alleviate the oscillations and smoothing that occur with different sizes of M.

$$F(\omega) = \frac{2M^2\omega\sin(M\omega)}{M^2\omega^2 - \pi^2} \tag{5}$$

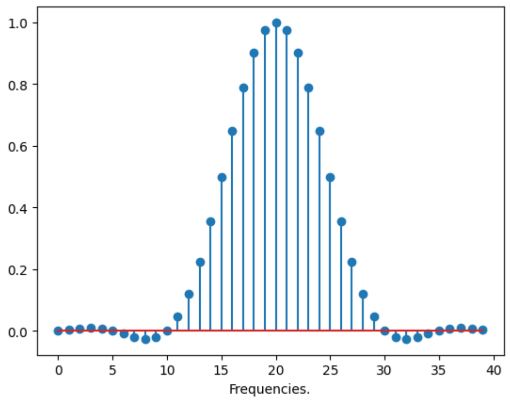
Using this function we can build the following filters:



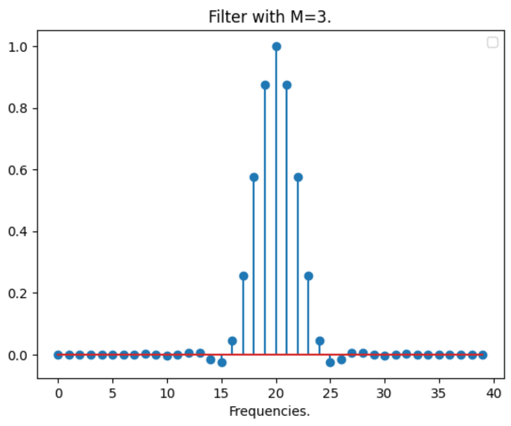
Filter with M = 1.



Filter with M = 2.



Filter with M = 2.



Filter with M = 3.

Figure 2. Filter

For better accuracy we generally choose our interval M to be small when we have a small fractional shift and big when we have a fractional shift that gets closer to 0.5. We could write a simplex expression that defines M as:

$$M = \begin{cases} 1.5 & if\, 0.1 < x < 0.3 \\ 2 & if\, 0.3 < x < 0.5 \end{cases}$$

With some experimenting we found that the filter with $M = 3/2$ is a good compromise between smoothing the values and keeping them as accurate as possible. for any fractional shift.

Now that we have our filters we can shift them by the fractional amount in the same way that we defined in equation 2. If for example we want to shift our original signal by 10.45 we will start by creating our filter with $M = \frac{3}{2}$. Then we will apply a shift of .45 to the filter:

4 Implementation and Project Design

Since the premise of this project is based on the fact that a rotation can be performed with three 1D operations, the bottom-up approach was a no-brainer. We started with the fundamentals of signal processing by exploring the Fourier Transform and Shift Theorem to build our shift kernel. We used Jupyter Notebook extensively to test our method and quickly plot the results, while also documenting everything so that it would be easy to write this report. Since we used the bottom-up approach we were able to re-use parts of the code to test some other aspect of the shift. We started with simple shifts of a 1D signal by an integer value which proved to be quite simple and then we tried to shift the same signal by a fractional amount. This is where some problems arose because some care is needed while shifting each frequency by the correct phase shift. The complete the project the tools that were used were mainly: MakeFile, CMake, Jupyter Notebook and some built-in libraries of C and Python like numpy and matplotlib.

4.1 Libraries

Numpy and Matplotlib were a huge help in plotting the results of our initial tests of the Fourier Transform and of the final results. These libraries were also useful for when we started writing code in C. Where we would output the binary result to a file and then read it and plot the result in python. There are a few plotting libraries for C but we felt that this was just the faster/easier solution. The whole project started out as a CMake and this was chosen mainly because I was already familiar with this tool and was able to quickly setup an environment that would build the executable. However towards the end we decided to just use a plain Makefile to keep everything nice and simple.

4.2 Creating a Filter in the Frequency domain

So first off we need to build our kernel. As an example we will create one of size 40 with $M = 2$ since we will be shifting by a fractional amount. To create the kernel then we follow equation 5, while also paying attention to evaluating the limit when $\omega = \frac{\pi}{M}$ or $\omega = -\frac{\pi}{M}$. This gives us:

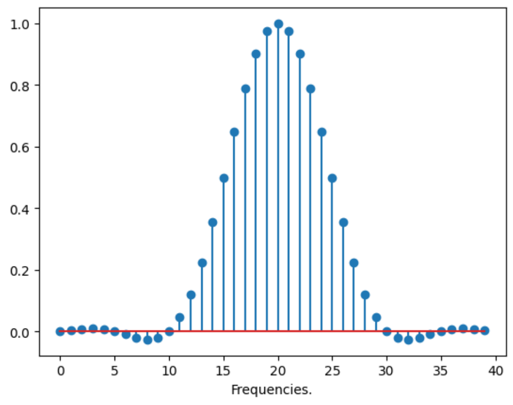


Figure 3. Filter with $M = 2$

Now let us say that the fractional amount to shift is 0.25. To implement this we can use a simple for loop that multiplies each frequency value of the filter by it's corresponding shift 3.

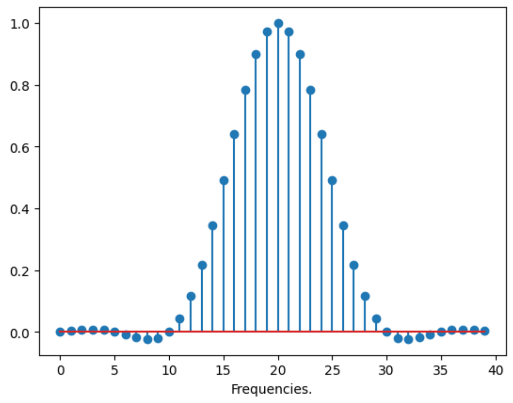


Figure 4. Filter with $M = 2$ after shift of 0.25

Afterwards we implemented a simple version of the inverse Fourier transform, which give us our signal in the time domain. Before returning the result however we need to perform the equivalent of numpy's fftshift function to have the filter values centered. This whole operation can be done in serial and with no real optimization of the code since the kernel size will always be much smaller than the input signal and as such will no be computationally expensive.

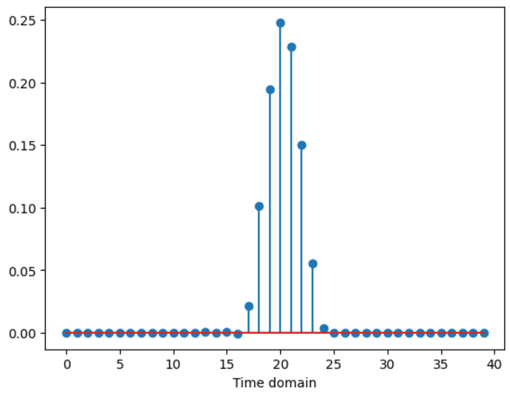


Figure 5. Filter with $M = 2$ in spatial domain.

Here is some pseudo-code that explain what is being done to our original filter:

Algorithm 1 Filter shift

4.3 Convolution Implementation

To keep our one dimensional convolution as simple and as efficient as possible we analyzed it using a Web tool called "Compiler Explorer"[1]. This tool allows us to see which assembly code is being used to perform the convolution operation and choose which implementation should run faster on certain compilers and CPU's.

4.4 Integer Shift

The end goal has always been to shift by any real value so now we need to handle the integer part of the shift. First of all we need to clarify that if the shift does not have a fractional part then we skip all of the filter creation and convolution and just follow the simple operation that is defined here. To shift a signal by an integer amount we will be using the C function *memcpy* to copy the values of the original array to a resulting array but at a different position, which is defined by the integer shift.

5 Results

5.1 Results with 1D signals.

Let us take as starting signal:

These are the results of shifting the signal by 20.25 for $M = \{1, \frac{3}{2}, 2, 3\}$:

At first glance we can see that with $M = 1$ we have too many oscillations present in the resulting signal and with $M = 3$ the signal is smoothed out too much. We can see though that if we set $M = 1$ and the fractional value is small, for example 0.01 we have less oscillations and the signal is more similar to the original.

This is because the there are less oscillations in the kernel after the phase shift for small fractional shifts. In this case:

We have maximum oscillations exactly at 0.5 and with $M = 1$:

This is why for a fractional shift that get closer to 0.5 it is necessary to use a bigger value for M such as $\frac{3}{2}$ or 2 to smooth out the oscillations:

Here we could add a part where we show a simple implementation of choosing the right M for the corresponding amount of fractional shift by looking at which one preserves best the filter

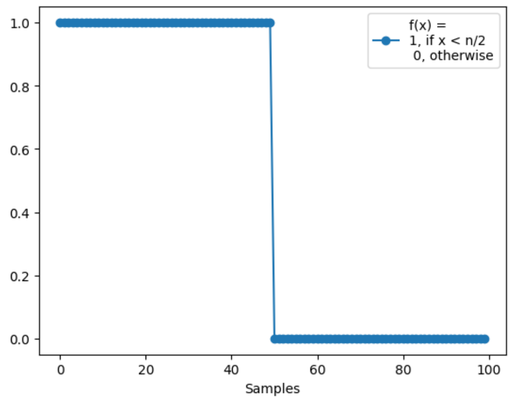
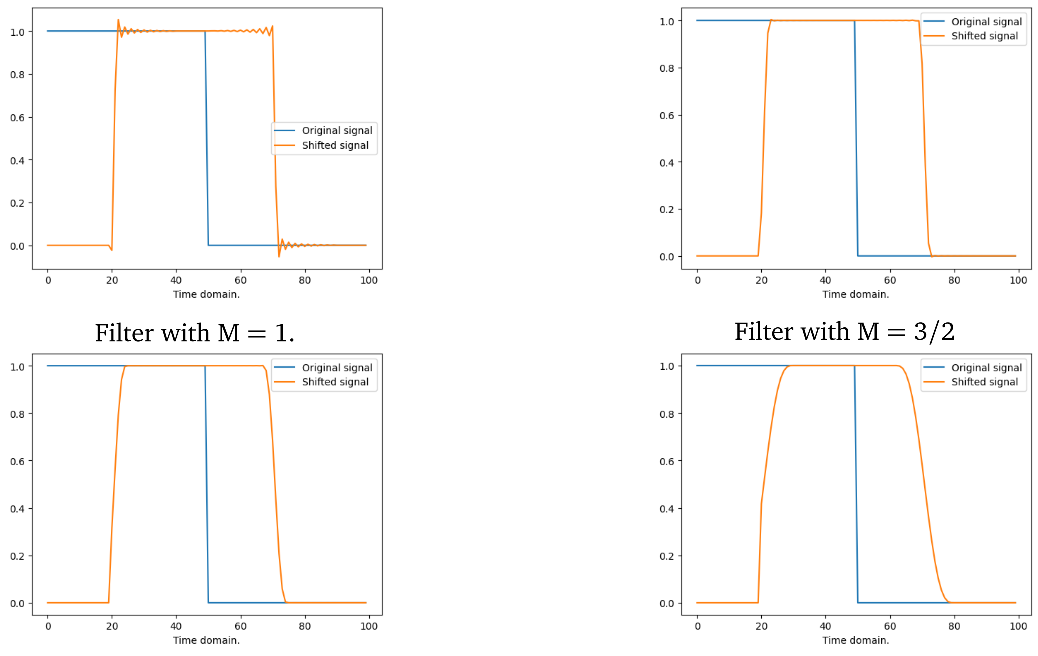


Figure 6. Original signal of size $n = 100$ samples.



Filter with M = 1. Filter with M = 3/2

Filter with M = 2. Filter with M = 3.

Figure 7. Shift by 20.25

5.2 Results on 2D signals, i.e images

6 Conclusion

7 Future works

One way to improve upon this work would be to implement the code with CUDA

References

[1] Matt Godbolt.

[2] IEEE Philippe Thévenaz Michel Unser, Senior Member and Leonid Yaroslavsky. Convolution-based interpolation for fast, high-quality rotation of images.

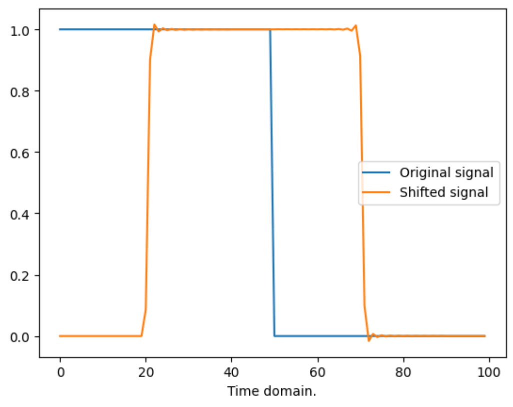


Figure 8. Shift by 20.01 with $M = 1$

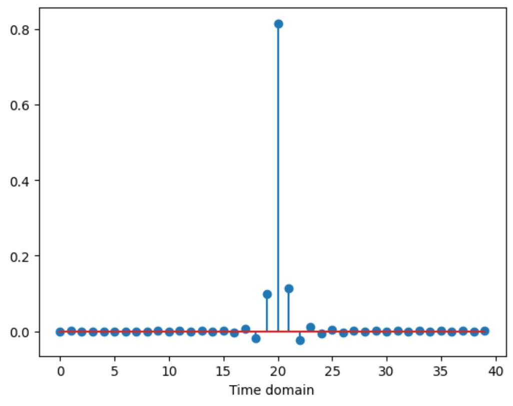


Figure 9. Filter shifted by 0.01 with $M = 1$

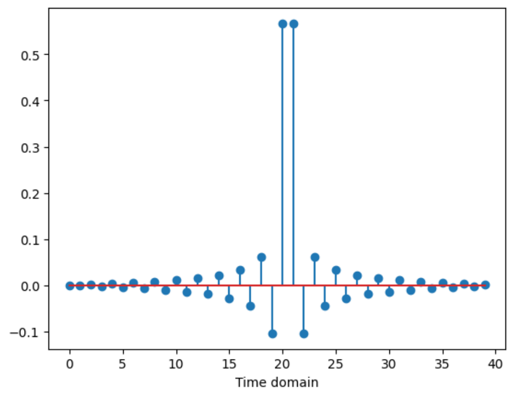


Figure 10. Filter shifted by 0.5 with $M = 1$

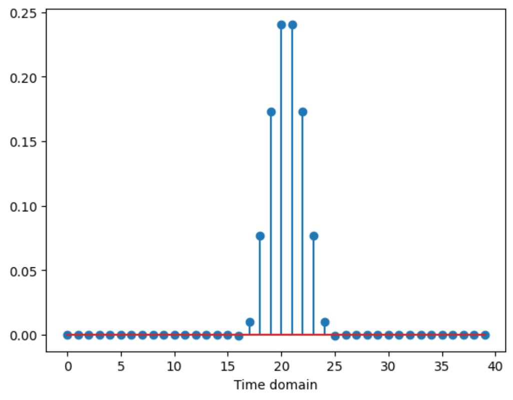


Figure 11. Filter shifted by 0.5 with $M = 2$