IMAGE REGISTRATION USING THE FFT

Amery Cong

https://github.com/amerycong/image_registration

ABSTRACT

For the image registration, we attempt to identify the pixel shifts in the rows and columns between any 2 images in a set of 5 MR scans of grapefruits. Using the Fourier Shift Property, we aim to obtain the phase correlation in order to determine the translational offset through analysis of the cross power spectrum. By taking the Fast Fourier Transform (FFT) along the two dimensions of any two images in the set, we can compute the phase correlation between the rows and columns of each image which is used to calculate the displacement from one image to the other. Due to the increased performance of phase correlation over geometric spatial correlation in terms of computation speed, we also observe the accuracy of this method compared to the actual pixel shift values.

1. INTRODUCTION

In the field of image processing, phase correlation is a method of image registration which computes the translations between two similar images. In the industry, the optimization of this process is essential in government fields such as healthcare where MRI scans are thoroughly compared against each other, as well as defense and security applications which require the identification of camouflaged units on a similar background. In university research and academia, there is a large emphasis on computer vision and pattern recognition as well.

For this particular project, we will use the phase correlation technique to identify the pixel shifts between any 2 of the 5 provided grapefruit images (Figure 1).

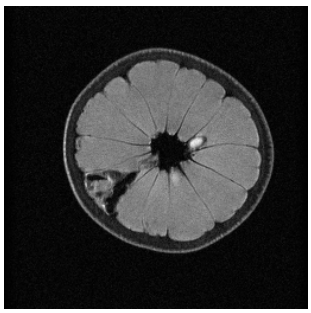


Fig. 1. The first of 5 grapefruit images

To do this, we make use of the Fourier Shift Property which states if:

$$y(n) = x(n-k) \tag{1}$$

then:

$$Y(k) = e^{-j2\pi/Nk} X(k) \tag{2}$$

If a given signal $y(n)$ is equal to a signal $x(n)$ time-shifted by k, then the resulting Fourier coefficients of the signals are related by the term $e^{-j2\pi/Nk}$.

For the application to 2D images, we can treat the row and column vectors as two 1-dimensional entities, and analyze them separately.

2. METHODS

Our implementation for the pixel shift analysis was written in MATLAB due to the ease of access of Digital Signal Processing and Image Processing toolboxes. Its status as an interpreted language also made it easy to debug and observe how our data changed as we performed operations on them.

First, we read in two images from the set and obtain their respective summed row and column vectors with the following code:

```
I1 = imread(image1);
r1 = sum(I1,1);
c1 = sum(I1,2);
I2 = imread(image2);
r2 = sum(I2,1);
c2 = sum(I2,2);
```

We then proceed to generate a Hamming Window to be applied to the signals in order to reduce side lobe patterns in the frequency domain. This improves our analysis as our algorithm will focus more on the center of the signals, which corresponds to the central area of the images where the grapefruit is located. MATLAB has a `hamming` function which creates the appropriate Hamming window based on the size of the input vector. These windows are generated using the code:

```
hamming_r1 = hamming(length(r1));
hamming_c1 = hamming(length(c1));
```

```
hamming_r2 = hamming(length(r2));
hamming_c2 = hamming(length(c2));

hamming_r1 = hamming_r1';
hamming_r2 = hamming_r2';

window_r1 = r1.*hamming_r1;
window_r2 = r2.*hamming_r2;
window_c1 = c1.*hamming_c1;
window_c2 = c2.*hamming_c2;
```

In this particular example, we are working with square images so that the length and width of all images are exactly the same. This means that all of our Hamming windows will be identical, although we kept them separate in our code for debugging purposes and robustness in the case of different-sized input images. A plot of the Hamming window for our vectors is shown in Figure 2.

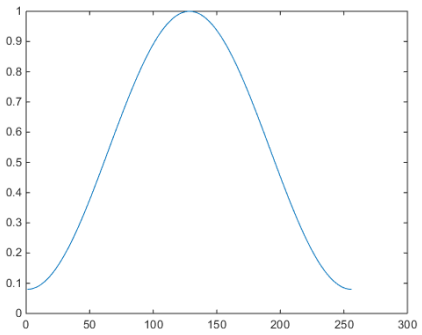


Fig. 2. Hamming Window

The Hamming window vectors for the summed rows are transposed so that the dimensions of the window stay consistent. For reference, our obtained Hamming window of the row vector of the first grapefruit image is shown below in Figure 3. Its appearance is similar to that of all the other windows (both row and column) we compute in this project.

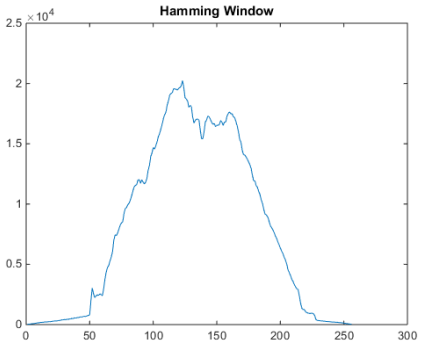


Fig. 3. Hamming Window for row vector of `img1.png`

Next, we proceed to obtain the Fast Fourier Transform (FFT) of our windowed vectors. The FFT is an algorithm (Equation 3) which computes the discrete Fourier transform of each vector, establishing a relationship between the time-domain (space-domain the case of 2-D images) and the frequency-domain.

$$X(k) = \sum_{i=0}^{N-1} x(i) e^{-j2\pi kn/N} \tag{3}$$

In the above equation, $X(k)$ refers to the summed row and column vectors for each processed image.

The MATLAB implementation simply required a call of the `fft` function on our 4 dimension vectors:

```
fft_r1 = fft(window_r1);
fft_r2 = fft(window_r2);
fft_c1 = fft(window_c1);
fft_c2 = fft(window_c2);
```

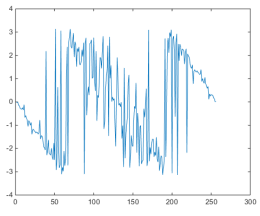
After computing the necessary FFT vectors, we proceed to calculate the phase difference using the cross power spectrum $C(k)$, as shown in Equation 4.

$$C(k) = \frac{X(k)Y(k)^*}{|X(k)Y(k)^*|} \tag{4}$$

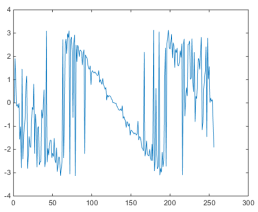
Here, $X(k)$ is the FFT row (or column) vector of Image 1, and $Y(k)$ is the FFT row (or column) vector of Image 2, with the asterisks denoting the conjugate counterparts.

```
row_correlation = ...
    (fft_r1.*conj(fft_r2))./abs(fft_r1.*conj(fft_r2));
col_correlation = ...
    (fft_c1.*conj(fft_c2))./abs(fft_c1.*conj(fft_c2));
```

Our final step in the our project is to calculate the slope of the phase signal in $C(k)$ and convert it to pixel units so we can compare our shift measurements. After computing the cross power spectrum for the row vectors, our plot of the angle of the data looked like a sawtooth wave with a noticeably noisier segment in the middle. Because the slope we need to determine is that of the outer segments with less noise, we use the `fftshift` function to swap the left and right halves of the signal so that the window is centered on the less noisy portion, which leads to easier slope calculations (Figure 4).



Original



Shifted

Fig. 4. Comparison of original and shifted phase signals

Looking at the zero-frequency shifted signal, we notice that there are still noisy peaks and troughs within the middle portion, which can affect our obtained slope. As a result, we need to extract a cleaner signal from within this middle portion. To do this, we start from the center of the signal and iterate forwards and backwards through the signal until we reach a point that deviates too much from the previous point, setting this point as the start/end point as appropriate. To calculate a value for what we deem to deviate "too much", we take the average of the absolute change in value (derivative) across the signal. Although there may be more robust ways to find the threshold, we have found that this method generates values that work without problems and seem reasonable when visually inspected. Figure 5 shows the obtained boundary points of the row correlation vector of `img1.png` and `img2.png` along with the extracted and windowed version subject to our threshold algorithm.

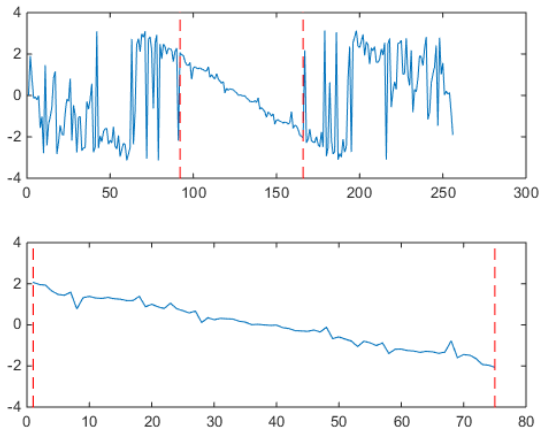


Fig. 5. Calculated boundaries of phase signal for slope measurement

To find the slope of this section of data, we use the first order `polyfit` function to fit the row and column signals to a line. The function will output a 1x2 vector, where the first element corresponds to the slope (m) and the second element corresponds to the y-intercept value (b) in the linear slope equation $y = mx + b$. This linear equation is plotted against the signal to show the closeness of fit in Figure 6.

Now that we have a slope value, we need to multiply it by a scale factor in order to obtain an answer in units.

```
scale_factor = -256/(2*pi);
y_shift = p_row(1)*scale_factor;
z_shift = p_col(1)*scale_factor;
```

With our pixel values obtained, we can proceed to calculate the error and analyze our results.

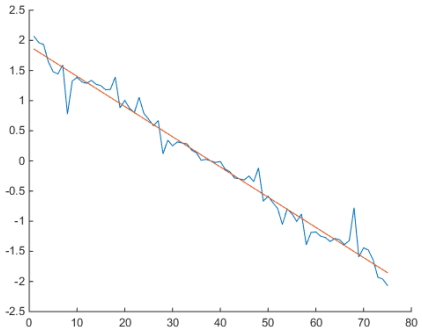


Fig. 6. Linear fit of extracted signal

3. RESULTS AND ANALYSIS

Our project consists of 5 image files (`img1.png`, `img2.png`, `img3.png`, `img4.png`, and `img5.png`) and 4 script files. The primary script `main.m` assigns the image set and initial parameters and calls `compute_shift.m` to carry out the actual pixel shift calculation. The `compute_shift.m` script calls on `find_boundaries.m` and `find_error.m` throughout its run.

Running our code on the set of 5 images produces the results as tabulated below (Table 1)

Image 1	Image 2	Z Shift	Y Shift
`img1.png`	`img2.png`	-3.904	2.042
`img1.png`	`img3.png`	-7.885	3.703
`img1.png`	`img4.png`	-4.212	4.820
`img1.png`	`img5.png`	-12.176	5.882
`img2.png`	`img3.png`	-3.955	2.238
`img2.png`	`img4.png`	-0.223	3.462
`img2.png`	`img5.png`	-7.661	4.213
`img3.png`	`img4.png`	3.656	2.379
`img3.png`	`img5.png`	-4.012	2.179
`img4.png`	`img5.png`	-7.612	-0.252

Table 1. Computed pixel shifts between pairs of images

To help better understand the significance of our results, the actual pixel shifts are provided in Table 2 and the absolute difference between the actual and computed values is shown in Table 3.

From the table of offsets, we can easily see that our computations for the Z shifts (columns) are fairly accurate, with an average offset of 0.108 pixels. On the other hand, our Y shifts (rows) are much more volatile, with an average offset of 0.774 pixels. This difference in accuracy can be attributed to the quality of our linear fit when determining the slope of our phase signal. To attempt to find out where this error comes from, we compare `img1.png` with `img4.png` because this pair of images resulted in a low Z offset (0.108 pixels) and a

Image 1	Image 2	Actual ΔZ	Actual ΔY
img1.png	img2.png	-4.000	2.400
img1.png	img3.png	-8.000	4.800
img1.png	img4.png	-4.320	7.200
img1.png	img5.png	-12.000	7.200
img2.png	img3.png	-4.000	2.400
img2.png	img4.png	-0.320	4.800
img2.png	img5.png	-8.000	4.800
img3.png	img4.png	3.680	2.400
img3.png	img5.png	-4.000	2.400
img4.png	img5.png	-7.680	0.000

Table 2. Actual pixel shifts between pairs of images

Image 1	Image 2	Z Offset	Y Offset
img1.png	img2.png	0.096	0.358
img1.png	img3.png	0.115	1.097
img1.png	img4.png	0.108	2.380
img1.png	img5.png	0.176	1.318
img2.png	img3.png	0.045	0.162
img2.png	img4.png	0.097	1.338
img2.png	img5.png	0.339	0.587
img3.png	img4.png	0.024	0.021
img3.png	img5.png	0.012	0.221
img4.png	img5.png	0.068	0.252

Table 3. Absolute difference between computed and actual shifts

high Y offset (2.380 pixels). Figure 7 shows the phase signal of the row vector prior to finding a linear fit and Figure 8 shows the phase signal of the column vector.

As evidenced from the two plots, we can see that the phase signal of the row vector is much noisier compared to that of the column vector. This would imply that in order to obtain more accurate results, we would have to add stricter constraints on finding the threshold in the case of the row vector. As a result, this would aid in finding a slope more representative of the extracted section and lead to a more accurate pixel shift. In terms of the input data, the noisy row vector may imply that the signal quality in the Y-direction is simply more degraded than the quality of the Z-direction signal.

4. CONCLUSION

This exploratory project has helped demonstrate how the Fast Fourier Transform and Discrete Time Fourier Transform can be applied to 2D images to detect pixel shifts. By converting an image from the space domain to the frequency domain, we are able to easily determine the phase shifts between two similar images. Despite the noise we observed in the image set, especially along the Z-direction, we were still able to obtain pixel shift values close to the actual shifts. With an overall

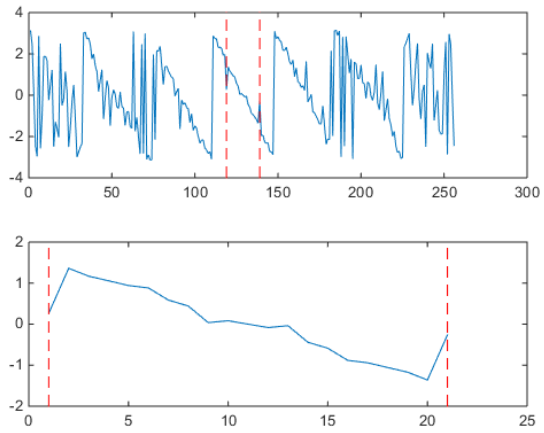


Fig. 7. Phase signal of row vector of `img1.png` vs `img4.png`

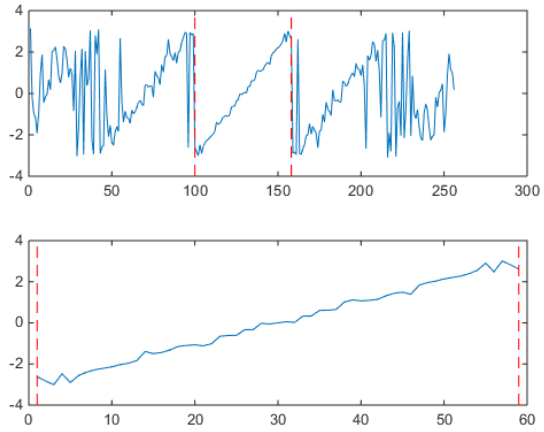


Fig. 8. Phase signal of column vector of `img1.png` vs `img4.png`

average error of approximately 0.44 pixels, we can conclude that the phase correlation method of determining translational offsets works satisfactorily.

5. REFERENCES

[1] C. D. Kuglin and D. C. Hines, "The phase correlation image alignment method," in *Proc. Int. Conf. on Cybernetics and Society*. IEEE, Sep. 1975, pp. 163–165.

[2] W. S. Hoge, "Subspace identification extension to the phase correlation method," *IEEE Trans. Medical Imaging*, vol. 22, no. 2, pp. 277–280, Feb. 2003.