TIE Jython Program

Ruixin Guo
Email: rguo5@kent.edu

February 17, 2025

1 Introduction

ImageJ is an image analysis software written in Java. Java is a programming language run on Java Virtual Machine (JVM). That is, if you want to run Java on a machine, you must install JVM on the machine first. Jython [1] is one JVM implementation of the Python programming language. It is designed to run Python code on JVM. One can think that Jython and Java are two different high-level languages that run on the same virtual machine and share the same low-level programming language. Jython and Python are the same high-level language but their underlying implementations are different. They run on different virtual machines and use different low-level languages. Jython is supported by ImageJ. It allows ImageJ to run a Python program.

The original TIE program is written in Matlab, which cannot be executed by ImageJ. To solve this problem, we developed the **TIE Jython Program** as an ImageJ plugin. It is a Jython implementation of the TIE Matlab Program, which can be executed by ImageJ.

2 Backgrounds

The most important function in the TIE program is Discrete Fourier Transformation (DFT).

DFT: Let the input vector be $x = [x_0, x_1, ..., x_{n-1}]$ and the output vector be $X = [X_0, X_1, ..., X_{n-1}]$. The DFT formula is

$$X_k = \sum_{j=0}^{n-1} x_j e^{-\frac{2\pi i}{n}kj} \qquad \text{for } k = 0, 1, ..., n-1$$

and the inverse DFT formula is

$$x_k = \frac{1}{n}\sum_{j=0}^{n-1} X_j e^{\frac{2\pi i}{n}kj} \qquad \text{for } k = 0, 1, ..., n-1$$

FFT: Fast Fourier transform (FFT) is an optimized algorithm of DFT. Suppose the input vector length is n, the time complexity of DFT is $O(n^2)$, while the time complexity of FFT is $O(n \log n)$. So using FFT instead of DFT will make the program run significantly faster when n is large. The most common FFT algorithm is Cooley-Tukey [2].

Inverse FFT: The inverse FFT is similar to inverse DFT. One can get inverse FFT by making the following modifications on the FFT: (1) Replacing the weights $e^{-\frac{2\pi i}{n}k}$ with $e^{\frac{2\pi i}{n}k}$ for $k = 0, 1, ..., n-1$; (2) Add a scale $\frac{1}{n}$ at the end of the output.

2D FFT: Suppose the input is a 2D matrix. The 2D FFT is to first apply 1D FFT to each row of the matrix, then apply 1D FFT to each column of the matrix. In Matlab, `fft2(X)` is equivalent to `fft(fft(X).').'`[3]. It is OK if applying FFT first to columns then to rows. The order does not matter.

Non-power-of-2 FFT: Cooley-Tukey FFT only supports power-of-2 input – it requires $n = 2^k, k \in \mathbb{N}$. Some FFT algorithms such as Rader's FFT [4] and Bluestein's FFT [5] support non-power-of-2 input.

3 Program Design Details

3.1 FFT

The main problem of Jython is that it does not support popular python packages such as `numpy`. This means we cannot use the FFT algorithm in `numpy`. However, ImageJ supports OpenCV library [6], and we can use the FFT algorithm in OpenCV. The function that performs FFT in OpenCV is `dft`.

The OpenCV library operates the data with a specific data class: `Mat`. The Jython program operates the data with Python `List`. In order to use the `dft` function in OpenCV, we need to convert the data to `Mat` class first. However, there is no direct way to convert a `List` to a `Mat`. ImageJ provides another data class `ImageProcessor` as an intermediary. One can convert `ImageProcessor` to either `List` or `Mat`, and vise versa.

So the whole process becomes:

1. Convert `List` to `ImageProcessor`.
2. Convert `ImageProcessor` to `Mat`.
3. Run `dft`: Input a `Mat`, and output a `Mat`.
4. Convert `Mat` to `ImageProcessor`.
5. Convert `ImageProcessor` to `List`.

The codes implementing the above process are:

```
from ij import ImagePlus
from ij.process import FloatProcessor
from org.bytedeco.javacpp.opencv_core import dft, DFT_COMPLEX_OUTPUT
from org.bytedeco.javacpp.opencv_core import Mat
from ijopencv.ij    import ImagePlusMatConverter
from ijopencv.opencv import MatImagePlusConverter

def fast_dst(x):
    n = len(x)
    y = [0] * n
    input0 = [0] + x + [0] * (n + 1)

    imp2mat = ImagePlusMatConverter()
    mat2ip = MatImagePlusConverter()

    pixel_matrix = split_list(input0, wanted_parts = 1)
    pixel_matrix = [list(x) for x in zip(*pixel_matrix)]
    img = ImagePlus("FFT", FloatProcessor(pixel_matrix))

    ImMat = imp2mat.toMat(img.getProcessor())
    ImMat_out = Mat()
    dft(ImMat, ImMat_out, DFT_COMPLEX_OUTPUT, 0)

    NewIP  = mat2ip.toImageProcessor(ImMat_out.reshape(1))
    ret = NewIP.getPixels()

    for i in range(n):
        y[i] = - ret[2 * i + 3]

    return y
```

I will explain the code step by step:

```
from org.bytedeco.javacpp.opencv_core import dft, DFT_COMPLEX_OUTPUT
from org.bytedeco.javacpp.opencv_core import Mat
```

`org.bytedeco.javacpp.opencv_core` is the OpenCV library I use, suggested by [6]. The API document of this library is on [7]. I imported the `dft` function, the `DFT_COMPLEX_OUTPUT` flag and the `Mat` class.

```
from ijopencv.ij     import ImagePlusMatConverter
from ijopencv.opencv import MatImagePlusConverter
```

`ijopencv` is an library which can convert data between ImageJ and OpenCV. Its source code is on [8].

```
pixel_matrix = split_list(input0, wanted_parts = 1)
pixel_matrix = [list(x) for x in zip(*pixel_matrix)]
img = ImagePlus("FFT", FloatProcessor(pixel_matrix))
```

`input0` is a Python `List`. It is one dimensional. These codes convert `input0` to a `ImagePlus` class `img`, which is a two dimensional matrix. One can convert `ImagePlus` to `ImageProcessor` by calling `getProcessor()` method. `wanted_parts` determines the number of rows of the matrix. Here `img` is a matrix with only one row – a row vector.

```
ImMat = imp2mat.toMat(img.getProcessor())
ImMat_out = Mat()
dft(ImMat, ImMat_out, DFT_COMPLEX_OUTPUT, 0)
```

The first line converts `img` to a `Mat` class named `ImMat`. The second line creates an empty `Mat` named `ImMat_out` for receiving the output. The third line is calling the `dft` function. The usage of this function is `static void dft(opencv_core.Mat src, opencv_core.Mat dst, int flags, int nonzeroRows)`. Here `src` is the input, `dst` is the output, both of them are `Mat` class. I set the `flag` to be `DFT_COMPLEX_OUTPUT` to make the output complex, otherwise the output will be real by default. I didn't use the fourth argument so just set it 0.

```
NewIP  = mat2ip.toImageProcessor(ImMat_out.reshape(1))
ret = NewIP.getPixels()
```

The first line converts `ImMat_out` to the `ImageProcessor` class. Note that the data type in `ImMat_out` is `CV_32FC2` (32-bit complex) by default, and `CV_32FC2` type cannot be converted to the `ImageProcessor` class. See the following source code of `MatImagePlusConverter` from [8] in `ijopencv.opencv`:

```
public static ImageProcessor toImageProcessor(Mat mat) {
    final int type = mat.type();
    ImageProcessor result = null;

    if (type == opencv_core.CV_8UC1) { // type = BufferedImage.TYPE_BYTE_GRAY;
        result = makeByteProcessor(mat);
    } else if (type == opencv_core.CV_8UC3) { // type = BufferedImage.TYPE_3BYTE_BGR;
        result = makeColorProcessor(mat); // faulty
    } else if (type == opencv_core.CV_16UC1) { // signed short image
        result = makeShortProcessor(mat);
    } else if (type == opencv_core.CV_32FC1) { // float image
        result = makeFloatProcessor(mat);
    } else {
        throw new IllegalArgumentException("cannot convert Mat of type " + type);
    }
    return result;
}
```

This shows `toImageProcessor` method only accepts `CV_32FC1` (32-bit real) type. So we need to convert `CV_32FC2` to `CV_32FC1`. One way to make this conversion is to use `reshape(1)` method [9]. For example, if the `CV_32FC2` type array is $[(0.5, -0.3), (-0.4, 0.6)]$, which contains two complex numbers $0.5 - 0.3i$ and $-0.4 + 0.6i$. After reshaping, the `CV_32FC1` type array would be $[0.5, -0.3, -0.4, 0.6]$, which contains four real numbers. The `getPixels()` method converts a `ImageProcessor` class to a Python `List`.

```
for i in range(n):
    y[i] = - ret[2 * i + 3]
```

If the size of `input0` is n, the size of `ret` would be $2n$. Remember that the index of a Python `List` starts from 0. So the $2 * i$-th element in `ret` represent the real part of the i-th complex number, and the $2 * i + 1$-th element in `ret` represent the imaginary part of the i-th complex number.

3.2 Adjusting Brightness

This part of code is newly added in the Jython program. The original Matlab program does not have this.

The brightness of an image depends on its intensity. Here the intensity of an image is defined as the average pixel values, i.e., the sum of pixel values divided by the number of pixels [10].

The key idea of adjusting the brightness is to make the overfocused image and the underfocused image having the same intensity. The code is as follows.

```
sz = Dim * Dim   # Get the size of the Image
avg0 = 0
avg1 = 0
for i in range(sz):
    avg0 = avg0 + Ia1[i]/sz # avg0 is the intensity of the first Image
for i in range(sz):
    avg1 = avg1 + Ib1[i]/sz # avg1 is the intensity of the second Image
avg2 = (avg0 + avg1)/2      # avg2 is the average intensity
for i in range(sz):
    Ia1[i] = Ia1[i] + (avg2 - avg0) # Replace the intensity of the first image with the
        average intensity
for i in range(sz):
    Ib1[i] = Ib1[i] + (avg2 - avg1) # Replace the intensity of the second image with the
        average intensity
```

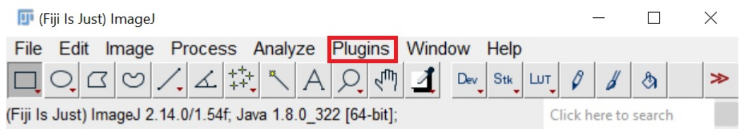
This code is executed after loading the images and before running TIE.

4 Manual

4.1 Install the Program

We use Fiji (an extended version of ImageJ) as an example to show how to install and run the TIE Jython program. Fiji can be downloaded from [12]. Using the Windows 64-bit version for example, the executable file of Fiji is located at Fiji.app/ImageJ-win64.exe, and you can start Fiji by double clicking it.

To install the TIE Jython program to Fiji, copy the file _TIEJython.py to the folder Fiji.app/plugins and restart Fiji. Then in the Fiji window, the program can be found at Plugins > TIEJython.



(Fiji Is Just) ImageJ 2.14.0/1.54f; Java 1.8.0_322 [64-bit];

Before running the program, you need to install OpenCV library. OpenCV is officially supported by Fiji. The installation steps are: Help > Update > Manage update sites > Select "IJ-OpenCV-plugins" > Close > Restart ImageJ [11].

4.2 Run the Program

1. Open the overfocused and underfocused images in Fiji. The images can be opened by `<<Drag and Drop>>` to the Fiji window. After finishing this, you should see the overfocused image and the

underfocused image as shown in Figure 1. The overfocused and underfocused images must have only one channel (grayscale) and the same height and width. The height and width do not need to be equal.

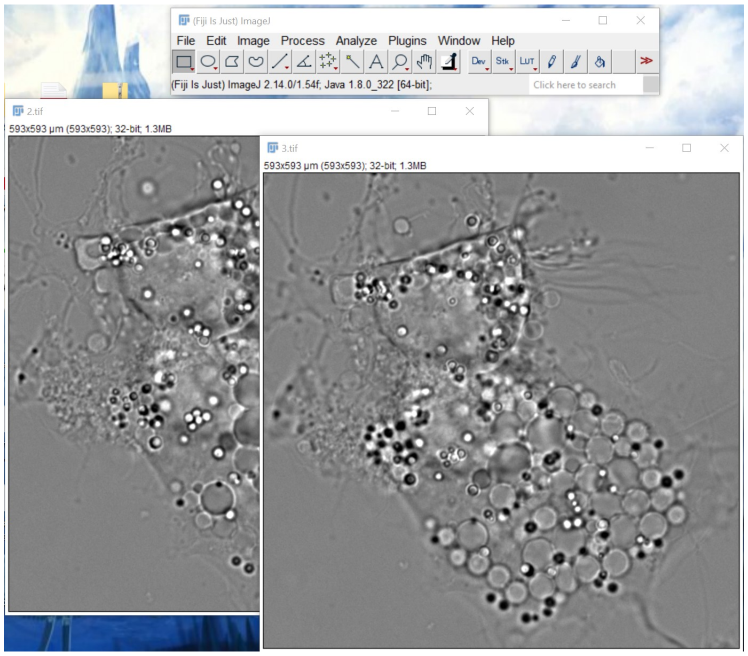
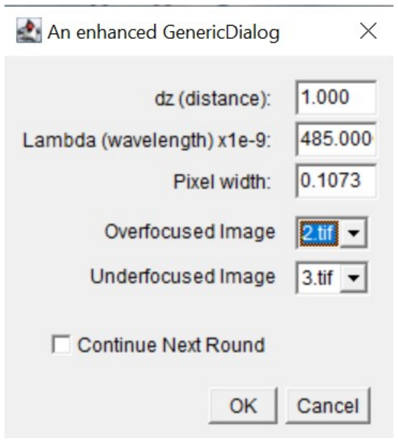

(Fiji Is Just) ImageJ
File Edit Image Process Analyze Plugins Window Help
(Fiji Is Just) ImageJ 2.14.0/1.54f; Java 1.8.0_322 [64-bit];
Click here to search
2.tif
593x593 µm (593x593); 32-bit; 1.3MB
3.tif
593x593 µm (593x593); 32-bit; 1.3MB

Figure 1: Open the overfocused and underfocused images

2. Run the TIEJython program, and you will see a window as shown in Figure 2. You need to set the overfocused and underfocused images. The default dz, Lambda and Pixel width (hp) are the same as those in TIE Matlab Program: $\mathrm{dz} = 1, \mathrm{Lambda} = 485 \times 10^{-9}, \mathrm{hp} = 0.1073$. Note that the program will scale the input of Lambda by 10^{-9}. After finishing the parameter settings, click OK, and the program will be executed. If you select "Continue Next Round", the program will not quit after the current round of calculation is finished, and the Dialog will reappear to enable next round.


An enhanced GenericDialog

dz (distance): 1.000

Lambda (wavelength) x1e-9: 485.000

Pixel width: 0.1073

Overfocused Image: 2.tif

Underfocused Image: 3.tif

☐ Continue Next Round

Figure 2: Set Parameters

3. After the program completes, you will get a Raw TIE Phase Image. The original Matlab Program will generate both Raw and Filtered Images, while TIEJython only generates the Raw Image.

5 Experimental Results

Due to the high performance of the `dft` function in OpenCV library, the program takes around 8 seconds to complete on my computer (Windows 10 OS, 11th Gen Intel Core i7-11800H @ 2.30GHz

CPU, 16GB Memory). The difference between the TIE Matlab Program and the TIEJython Program without adjusting brightness is between 10^{-7} and 10^{-6}.

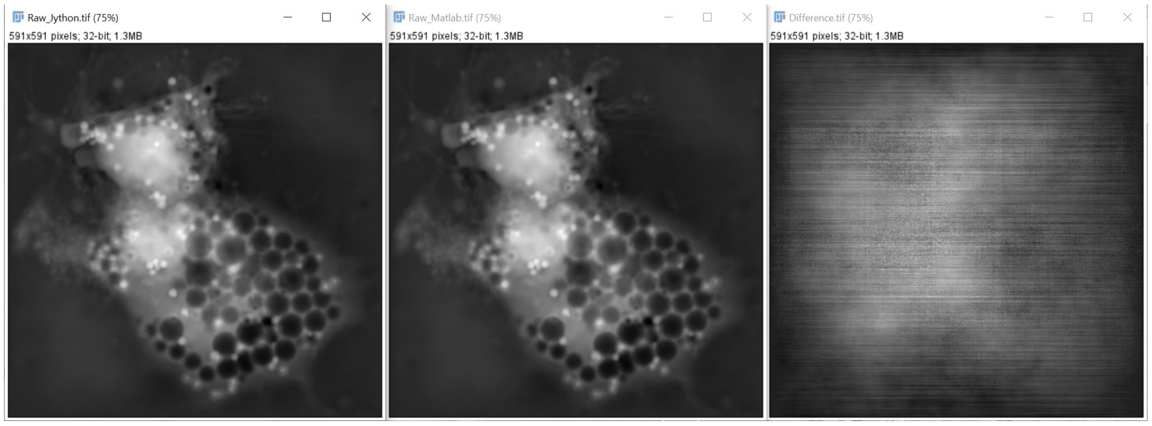

Raw_Jython.tif (75%)
591x591 pixels; 32-bit; 1.3MB
Raw_Matlab.tif (75%)
591x591 pixels; 32-bit; 1.3MB
Difference.tif (75%)
591x591 pixels; 32-bit; 1.3MB

Figure 3: The Result of TIEJython (Left), the Result of TIE Matlab Program (Middle) and their Difference (Right). The Difference is obtained by pixel-wise subtraction.

References

[1] https://www.tutorialspoint.com/jython/index.htm

[2] https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

[3] https://www.mathworks.com/help/matlab/ref/fft2.html

[4] https://en.wikipedia.org/wiki/Rader%27s_FFT_algorithm

[5] https://ccrma.stanford.edu/~jos/st/Bluestein_s_FFT_Algorithm.html

[6] https://imagej.net/scripting/jython/

[7] https://javadoc.io/static/org.bytedeco.javacpp-presets/opencv/3.4.0-1.4/org/bytedeco/javacpp/opencv_core.html

[8] https://github.com/joheras/IJ-OpenCV/tree/master/src/main/java/ijopencv

[9] https://stackoverflow.com/questions/34454178/convert-a-matrix-of-type-cv-32fc2-to-type-cv-32fc1

[10] https://cellprofiler-manual.s3.amazonaws.com/CPmanual/MeasureImageIntensity.html

[11] https://github.com/joheras/IJ-OpenCV

[12] https://imagej.net/software/fiji/