

Understanding FFT- based algorithm to calculate image displacements with IDL programming language

Cynthia Rodriguez
University of Texas at San Antonio

ABSTRACT

There are many ways automatic image registration can be solved. However, not many have been implemented into a software package like ENVI. The following descriptions of the `shift_idl.pro` and `srs_idl.pro` programs will prove to be a difference. These two programs aim to be a solution to automatic image registration for the state-of-the-art software ENVI (Environment for Visualizing Images) through an FFT based algorithm. FFT an acronym for Fast Fourier Transform is an algorithm known for successfully performing automatic image registration. Using IDL (Interactive Data Language) we are able to implement this algorithm with IDL's built-in FFT function. With the aid of the programming code from `shift_idl.pro` and `srs_idl.pro`, I hope to get a better understanding of how FFT based algorithm is used to calculate image displacements.

INTRODUCTION

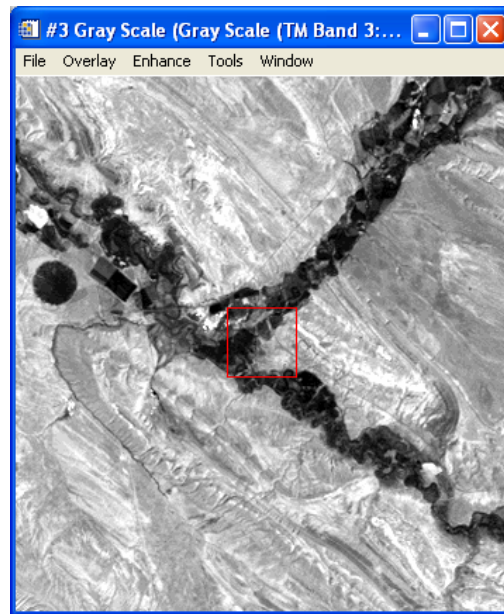
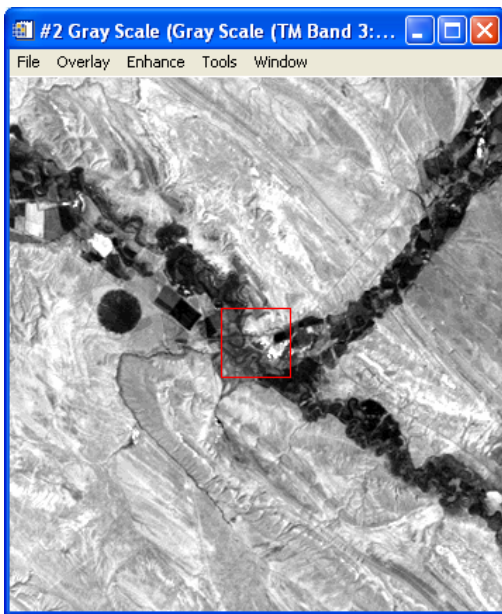
There is a saying that states "Don't reinvent the wheel" which basically means, if it's already been programmed, don't program it again. Instead it is strongly suggested to reuse the existing code. Doing this provides time saved. This is a highly accepted and encouraged practice. However to reuse someone

else's code demands patience, coding expertise, and notes, lots of notes in some cases. In general, programming languages are like human languages in that the person/programmer using the language can contribute their own style to the language even though the language has defined syntactic and semantic rules. For example, there is more than one way to execute a loop. There are for loops, while loops, and do loops. In addition, to considering a programmer's style, there is also the fact of understanding the goal of the program. This is something that must be taken into context. What is the programmer trying to accomplish? Furthermore, if the program is a highly advanced mathematical program, than without a doubt comments and notes will be needed. Therefore, in order to accurately understand the two programs `shift_idl.pro` and `srs_idl.pro`, I felt that a thorough walk through of the code and notes/comments was the most effective way to learn how automatic image registration could be accomplished.

STUDY AREA and DATA

To understand how IDL programming can be used to solve automatic image displacement, I used `shift_idl.pro` and `srs_idl.pro` IDL programs. These two programs were created by Hongjie Xie, Nigel Hicks, G. Randy Keller, Haitao Huang, and Vladik Kreinovich. The images used to test the programs were an original image, and the other a simulated image taken from using the original image and transforming it by a scaling of 1.3, rotation of 19 degrees to the east, and a shifting to the east (right) by 19 rows (pixels). These two images were used to test the procedures `shift`, `rotation` and `scaling`. The algorithm coded requires

that both images have the same size of resolution. However, the images can be in different coordinate systems. In addition, to the provided code, I also used an article called “An IDL/ENVI implementation of the FFT-based algorithm for automatic image registration” that helped guide me through the steps of the code.



METHODS

In order to understand how the two programs solve automatic image registration, I first decided to read the article and get a little foundation on what image registration was. The definition of automatic image registration is that it is the process that aligns one image to a second image of the same scene through a program. This is done so that the properties of an object being imaged can be addressable by the same coordinate pair from either one of the images. I then

needed to find out what the FFT based algorithm was. From the article, I learned that FFT stands for Fast Fourier Transform, and that it is a proven successful factor in solving image registration. The main formula behind FFT is getting the Ratio of FFT on image1 times conjugate of FFT on image2 and then with all of this, divide it by the absolute of FFT on image 1 and FFT on image 2.

$$\text{Ratio} = F1 \text{ conj}(F2) / |F1F2|$$

Once the ratio is calculated, then apply an inverse fourier transform. Based on these calculations you will find the displacement between two images.

$$\text{Ex: Ratio} = (x0,y0), [I2(x,y) = I1(x-x0, y-y0)]$$

When this algorithm is implemented in coding, you get a result of a single array with values close to zero everywhere except for a small area around a single point. To locate this point call the function Max (build-in IDL function) to find the position of Max value. This position is the exact shift displacement between the images (x0, y0). Using this displacement you can optimally register the images. With this gained knowledge, I now felt confident to understand the coding for shift_idl. Doing a step by step walkthrough of the program, I noticed the beginning of the program was pretty standard IDL code of initiating variables and opening the two image files and reading the data into a declared double-dimension arrays. Once the data was received, the code then calls FFT on both images. With the forward fast fourier transform calculations, the code then computes the Ratio and inverse fourier transform on the ratio. Next the code calls MAX function to locate the max value position. Using Max position, the programmer then gets the coordinates of the displacement by performing MOD

and DIVIDE operators to get X – shift in columns and Y- shift in rows. Last the code runs a conditional statement to check for shifts to the west and north and makes them negative numbers while the shifts to the east and south stay positive numbers.

After understanding shift_idl.pro, I proceeded to analyze the code for srs_idl.pro which includes a rotation and scaling as well as shifts. The main idea behind representing rotation and scaling as shifts was to convert rectangular coordinates (x, y) to log-polar coordinates ($\log(p)$, θ). However by doing this, it was discovered that certain points newly created didn't match with the points on the original rectangular grid. Due to the unmatched coordinates, the referenced article concluded to use bilinear interpolation to solve the problem. Bilinear interpolation basically gets the surrounding intensities around the current point and interpolates it. Accordingly, it can then get the correct value on the new grid. With this knowledge, I next started a step-by step walkthrough of the srs_idl.pro code. I noticed it begins like shift_idl.pro, but the difference is that since srs_idl.pro contains more programming code than shift_idl.pro, the srs_idl.pro program makes a point to provide easily modulated code of functions. This provides easier reading and change and reuse if necessary. The first functions in this program LoadImage, Normalize, ShiftArr, main task is to get the data ready in a predefined format so that calculations can be made more easily. LoadImage opens and reads image files into Im1 and Im2 variable. Normalize gets the grid values in an array and makes the array a floating array and divides values by 255.0 to make values between 0 and 1. ShiftArr makes values in the array

negative values, if MOD 2 equals 1. From then it next calls FFT on both images and then abs function to get absolute values of image1 and image2. Afterwards, it calls HighPass function which removes low frequency noise to the absolute values. The code next calls LogPolar function on both images to transform the rectangular coordinates to log polar coordinates in the process also calculating bilinear interpolation to get accurate coordinates on the new grid. In order for this function to run properly, the polar plane must have the exact same number of rows as the rectangular plane. To achieve this a nested for loop was created to go through each index in the array and individually get the original coordinate and transfer it to the log polar coordinate. The code now is able to use the polar coordinates and find out the scale and rotation shifts by calling FFT function to both image1 and image2. Then it finds the ratio by calling function Ritio, and next calling FFT inverse function on the ratio. Afterwards, the code calls the absolute function to the values. Now, the code can call Max function to find the max position in the array of absolute values. With the max position it can then calculate and find the values for scale and angle using the Max location.

$$\text{scale}=\text{b}^{\text{(position MOD rows)}}$$

$$\text{angle}=180.0 * (\text{position}/\text{rows}) / \text{cols}$$

Once the angle is found the programmer makes a conditional statement to check to see if the angle is greater than or less than 90 degrees. If an angle is greater than 90 or equal to 90, the angle is positive and the image is rotated to the east relative to the base image. The actual angle rotation is 180 minus the computed angle. If the angle is less than 90 degrees, than the angle is negative and the

image is rotated to the west. The article referenced says that the angle is always between 0 and 180, because it assumes that the image received will never be completely upside down. The image, if upside down, will be manually corrected to right side up. After the angle and scale have been computed, the code creates a new image3 with the function ROT which constructs a new image by doing reverse rotation and reverse scaling. Now that we have the data for the new image3, the programmer calls FFT function for both image1 and image3. The rest of the code is exactly like shift_idl.pro. Basically once again the programmer uses the FFT values to compute the Ratio and call inverse FFT on the Ratio. Then the code uses Max function to locate the max value in the array, and with the max position found, it gets the X and Y coordinates. Last the program checks to see if coordinates are out of bounds and adjust accordingly for north and west shifts which are computed as negative.

RESULTS and DISCUSSION

The results of both programs are identical in that each program's goal is to get the calculation of the shift displacement between two images. The difference is that srs_idl.pro has to go through more steps of calculating since not only does the code compute shift, but also rotation and scaling displacements. In summary, the shift only program contains only two forward FFTs and one inverse FFT. While on the other hand, the scale, rotation, and shift program contains six forward FFTs and two inverse FFTs. This involves a lot more computation, therefore this program requires more time and memory. Once more over, FFT-

Cynthia Rodriguez

Remote Sensing

2/7/2007

based algorithm is a proven success in solving image registration. Understanding its implementation in two programs `shift_idl.pro` and `srs_idl.pro` was accomplished through the aid of the referenced article. Conclusion, I was able to understand how FFT-base algorithm is used with IDL programming language to solve automatic image registration.

REFERNCES

Hongjie Xie, Nigel Hicks, G. Randy Keller, Haitao Huang, Vladik Kreinovich. "An DL/ENVI implementation of the FFT-based algorithm for automatic image registration." Department of Geological Sciences, Pan American Center for Earth and Environmental Studies (*PACES*), University of Texas at El Paso, Texas 79968, USA. March 26, 2003.