Real-time image fusion processing for astronomical images

Mohamed AbouRayan
University of Toledo

Follow this and additional works at: http://utdr.utoledo.edu/theses-dissertations

Recommended Citation
http://utdr.utoledo.edu/theses-dissertations/1984

This Thesis is brought to you for free and open access by The University of Toledo Digital Repository. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of The University of Toledo Digital Repository. For more information, please see the repository's About page.
A Thesis

entitled

Real-time Image Fusion Processing for Astronomical Images

by

Mohamed AbouRayan

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the

Master of Science Degree in

Electrical Engineering

_______________________________
Dr. Mohsin Jamali, Committee Chair

_______________________________
Dr. Jackson Carvalho, Committee Member

_______________________________
Dr. Ezzatollah Salari, Committee Member

_______________________________
Dr. Vincent Schmidt, Committee Member

_______________________________
Dr. Patricia R. Komuniecki, Dean
College of Graduate Studies

The University of Toledo

May 2016
An Abstract of
Real-time Image Fusion processing for astronomical Images

by

Mohamed AbouRayan

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the
Master of Science Degree in
Electrical Engineering

The University of Toledo

May 2016

This thesis provides a detailed study of ten different methods of Image fusion
techniques and their theoretical background and their step-by-step detailed algorithmic
implementation. Implementations of these techniques were applied to astronomical
images to further study the results using MATLAB. The MATLAB code created for
Astronomical Image fusion is provided for quick implementation and validation of the
results.

Then this paper compares the results of the ten different methods of image fusion
done over astronomical images. The comparison is made using execution time of each
algorithm and thirteen different ways of image quality and image fusions effectiveness
with regards to astronomical images. The thirteen measures of effectiveness are studied,
and their theoretical background and their step-by-step detailed algorithmic
implementation are provided. The MATLAB code created for the thirteen measures of
effectiveness is provided for quick implementation and validation of the results.

After figuring out the best algorithms in MATLAB that gives either best image
quality results or best execution time, the paper implement four algorithms using Python.
with the aim to continue further study with parallel and cloud computing. These algorithms are Principal Component Analysis (PCA), Average method, Discrete Wavelet Transform (DWT) using Haar filter and Discrete Wavelet Transform (DWT) using Daubechies filter.

The work then provides an introductory study of parallel computing, major concepts in their implementation and discusses issues facing the parallel implementation. Then, a parallel computing implementation of the Python code for Astronomical Image fusion is applied using two methods. A default Python library and an external Python library named JobLib and discussion, and comparison of the results versus the sequential computing is provided. The Python code for sequential and parallel computing is provided for quick implementation and validation of the results.

The paper then explores the emerging field of cloud computing and its advantages over regular processing. Then, an implementation of the Python code for Astronomical Image fusion is applied using Amazon Web service’s (AWS) cloud computing service Amazon Elastic Compute Cloud (EC2) with two different systems a Windows-based system and a Linux based system and then discussion and comparison of the results versus the sequential computing is provided. The Python code for the cloud computing is provided for quick implementation and validation of the results.

Finally in this thesis, a method is provided that compares two astronomical images and returns a probability if these images came from the same source by extracting the fixed error signal produced by the Charge-coupled device (CCD) camera which would be unique to each camera and then compare it. MATLAB code has been provided for quick implementation and validation of the proposed algorithm.
Acknowledgements

First, I would like to thank my thesis advisor Dr. Mohsin Jamali of the Electrical Engineering department at the University of Toledo. The door to Dr. Jamali’s office was always open whenever I ran into a trouble spot or had a question about my research or writing. This work could not be completed without his guidance, encouragement, and advice. I would also like to thank Dr. Vincent Schmidt for providing astronomical images and encouragement to work on this topic. Finally, I would like to thank my committee members Drs. Carvalho, Salari and Schmidt in evaluating this thesis.

Secondly, I would like to thank my parents (Magdy and Eman), my brother (Sherif), my sister-in-law (Elma) and my sweetheart (Meaghan) for providing me with unfailing support and continuous encouragement throughout my studies and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Finally, I would like to thank all my family and friends in Louisville and Toledo for always supporting and encouraging me.
# Table of Contents

Acknowledgements........................................................................................................ v

Table of Contents........................................................................................................... vi

List of Tables .................................................................................................................. x

List of Figures ................................................................................................................. xi

List of Abbreviations ..................................................................................................... xiv

List of Symbols .............................................................................................................. xv

1. Introduction .................................................................................................................. 1

   1.1. Astronomical images ............................................................................................. 3

   1.2. Thesis outline ....................................................................................................... 5

2. Image Fusion ................................................................................................................. 6

   2.1. Image Fusion Techniques ..................................................................................... 6

      2.1.1. Select Minimum ............................................................................................ 7

      2.1.2. Select Maximum .......................................................................................... 8

      2.1.3. Average Method .......................................................................................... 10

      2.1.4. Principal component analysis ...................................................................... 11

      2.1.5. Discrete wavelet transformation using Daubechies ..................................... 14

      2.1.6. Discrete wavelet transformation using Haar .................................................. 17

      2.1.7. Discrete stationary wavelet transform [1 Level] .......................................... 20

      2.1.8. Discrete stationary wavelet transform [2 Levels] .......................................... 23

      2.1.9. Dual-tree complex wavelet transform ............................................................ 26

      2.1.10. Discrete cosine transform based Laplacian pyramid .................................... 29
2.2. Implementation of Fusion techniques using MATLAB ........................................ 33

2.3. Image Measures Of Effectiveness (MOE) ................................................................. 34

2.3.1. Root Mean Square Error (RMSE) ........................................................................... 35
2.3.2. Percentage Fit Error (PFE) .................................................................................. 36
2.3.3. Mean-Squared Error (MSE) ................................................................................ 36
2.3.4. Mean Absolute Error (MAE) ................................................................................ 37
2.3.5. Difference Entropy ............................................................................................... 37
2.3.6. Normalized Cross Correlation (CORR) .............................................................. 39
2.3.7. Signal to Noise Ratio (SNR) ................................................................................ 39
2.3.8. Peak Signal to Noise Ratio (PSNR) ................................................................. 40
2.3.9. Structural Content (SC) ....................................................................................... 41
2.3.10. Laplacian Mean Squared Error (LMSE) ............................................................ 42
2.3.11. Structural Similarity Index Measurement (SSIM) ............................................. 43
2.3.12. Mutual Information (MI) ................................................................................... 44
2.3.13. Universal Image Quality Index (QI) ............................................................... 45

2.4. Implementation of Image measures of effectiveness .............................................. 46

2.5. Results using MATLAB .......................................................................................... 46

2.5.1. Image Quality Results ......................................................................................... 46
2.5.2. Image Fusion Results ......................................................................................... 54
2.5.3. Time Results ....................................................................................................... 64

2.6. Implementation of Fusion techniques using Python ............................................. 65

2.7. Results using Python ............................................................................................... 66

2.8. Conclusion ............................................................................................................... 67
3. Parallel Computing ......................................................................................... 69
   3.1. Introduction to parallel computing ............................................................... 69
   3.2. Implementation of parallel computing using Python .................................. 74
   3.3. Results ....................................................................................................... 76
   3.4. Conclusion ................................................................................................. 76

4. Cloud Computing ............................................................................................ 81
   4.1. Introduction to Cloud Computing ............................................................... 81
   4.2. Implementation of cloud computing .......................................................... 86
   4.3. Results ....................................................................................................... 90
   4.4. Conclusion ................................................................................................. 90

5. Further Image Fusion implementation ............................................................. 95
   5.1. Space Situational Awareness (SSA) ............................................................ 95
   5.2. Image Characterization ............................................................................... 98
   5.3. Big-Data Image Fusion ............................................................................... 100

6. Conclusion and Future work ........................................................................... 104

References ........................................................................................................... 109

Appendix A .......................................................................................................... 114
Appendix B .......................................................................................................... 118
Appendix C .......................................................................................................... 126
Appendix D .......................................................................................................... 139
Appendix E .......................................................................................................... 144
Appendix F ................................................................................................................................. 150

Appendix G ................................................................................................................................. 156
List of Tables

Table 2-1. MATLAB Time Results ................................................................................................................. 64
Table 2-2. Python Sequential Time Results ..................................................................................................... 66
Table 3-1. Parallel Method One Time Results ............................................................................................... 76
Table 3-2. Parallel Method Two Time Results ............................................................................................... 76
Table 3-3. Parallel processing comparison of Average Method ................................................................. 77
Table 3-4. Parallel processing comparison of Principal Component Analysis ........................................... 78
Table 3-5. Parallel processing comparison of discrete wavelet transformation using Daub ........... 79
Table 3-6. Parallel processing comparison of discrete wavelet transformation using Harr ........... 80
Table 4-1. Microsoft Windows Server Cloud Time Results ........................................................................... 90
Table 4-2. Ubuntu server Cloud Time Results ............................................................................................. 90
Table 4-3. Cloud processing comparison of Average Method ................................................................. 91
Table 4-4. Cloud processing comparison of Principal Component Analysis ........................................... 92
Table 4-5. Cloud processing comparison of discrete wavelet transformation using Daub ........... 93
Table 4-6. Cloud processing comparison of discrete wavelet transformation using Harr ........... 94
Table 6-1. Processing time comparison of Average Method ....................................................................... 106
Table 6-2. Processing time comparison of Principal Component Analysis ............................................. 106
Table 6-3. Processing time comparison of discrete wavelet transformation using Daub ........... 106
Table 6-4. Processing time comparison of discrete wavelet transformation using Harr ........... 106
Table 6-5. Speed-up comparison of Average Method ................................................................................. 107
Table 6-6. Speed-up comparison of Principal Component Analysis ......................................................... 107
Table 6-7. Speed-up comparison of discrete wavelet transformation using Daub ................................. 107
Table 6-8. Speed-up comparison of discrete wavelet transformation using Harr ............................. 107
List of Figures

Figure 1-1. Astronomical FITS image ......................................................................................... 3
Figure 1-2. FITS header ................................................................................................................ 4
Figure 2-1. Select Minimum ........................................................................................................... 8
Figure 2-2. Select Maximum .......................................................................................................... 9
Figure 2-3. Average Method ......................................................................................................... 10
Figure 2-4. PCA algorithm .......................................................................................................... 13
Figure 2-5. Daubechies wavelet transform ................................................................................... 14
Figure 2-6. Daubechies filter bank .............................................................................................. 15
Figure 2-7. DWT using Daubechies ........................................................................................... 16
Figure 2-8. Haar wavelet transform ............................................................................................ 17
Figure 2-9. Haar filter bank ........................................................................................................ 18
Figure 2-10. DWT using Haar .................................................................................................... 19
Figure 2-11. Discrete stationary wavelet transform ...................................................................... 20
Figure 2-12. DSWT filter bank .................................................................................................. 21
Figure 2-13. DSWT (1 Level) .................................................................................................... 22
Figure 2-14. Discrete stationary wavelet transform (2 levels) ..................................................... 23
Figure 2-15. DSWT filter bank (2 levels) ................................................................................... 24
Figure 2-16. DSWT (2 Levels) .................................................................................................. 25
Figure 2-17. Dual-Tree complex wavelet transforms ................................................................... 27
Figure 2-18. DTCWT filter bank ............................................................................................... 27
Figure 2-19. DTCWT .................................................................................................................. 28
Figure 2-20. Laplacian Pyramid ................................................................................................. 31
Figure 2-21. DCTLP .......................................................... 32
Figure 2-22. Root Mean Square Error ...................................................... 47
Figure 2-23. Percentage Fit Error ............................................................ 47
Figure 2-24. Mean Squared Error ........................................................... 48
Figure 2-25. Mean Absolute Error ......................................................... 48
Figure 2-26. Difference Entropy ............................................................ 49
Figure 2-27. Correlation ........................................................................ 49
Figure 2-28. Signal to Noise Ratio .......................................................... 50
Figure 2-29. Peak Signal to Noise Ratio ................................................... 50
Figure 2-30. Structural Content ............................................................. 51
Figure 2-31. Laplacian Mean Squared Error ............................................ 51
Figure 2-32. Structural Similarity Index ................................................... 52
Figure 2-33. Mutual Information ............................................................. 52
Figure 2-34. Universal Image Quality Index ............................................. 53
Figure 2-35. Average Method Image Fusion Results ................................ 54
Figure 2-36. Discrete cosine transform based Laplacian pyramid Image Fusion Results ........ 55
Figure 2-37. Discrete stationary wavelet transform [1 Level] Image Fusion Results ............. 56
Figure 2-38. Discrete stationary wavelet transform [2 Levels] Image Fusion Results .......... 57
Figure 2-39. Dual-tree complex wavelet transforms Image Fusion Results .................. 58
Figure 2-40. Discrete wavelet transformation using Daubechies Image Fusion Results .......... 59
Figure 2-41. Discrete wavelet transformation using Haar Image Fusion Results .............. 60
Figure 2-42. Select Maximum Image Fusion Results .................................. 61
Figure 2-43. Select Minimum Image Fusion Results .................................... 62
Figure 2-44. Principal component analysis (PCA) Image Fusion Results ..................... 63
Figure 2-45. MATLAB Time Results ...................................................... 64
Figure 3-1. Deadlock [28]................................................................. 73
Figure 3-2. Parallel processing comparison of Average Method............................ 77
Figure 3-3. Parallel processing comparison of Principal Component Analysis.............. 78
Figure 3-4. Parallel processing comparison of discrete wavelet transformation using Daub .... 79
Figure 3-5. Parallel processing comparison of discrete wavelet transformation using Harr ...... 80
Figure 4-1. Cloud classifications based on the location of the cloud [31]......................... 83
Figure 4-2. Cloud classifications based on the Service of the cloud [32].......................... 84
Figure 4-3. Comparison of different cloud services 1 .............................................. 87
Figure 4-4. Comparison of different cloud services 2 .............................................. 87
Figure 4-5. Amazon Elastic Compute Cloud (EC2) instances used ................................ 88
Figure 4-6. Cloud processing comparison of Average Method ..................................... 91
Figure 4-7. Cloud processing comparison of Principal Component Analysis .................. 92
Figure 4-8. Cloud processing comparison of discrete wavelet transformation using Daub........ 93
Figure 4-9. Cloud processing comparison of discrete wavelet transformation using Harr........ 94
Figure 5-1. Ibex SSA data fusion implementation [35].............................................. 97
Figure 5-2. Big Data Image Fusion Results 1 ....................................................... 100
Figure 5-3. Big Data Image Fusion Results 2 ....................................................... 101
Figure 5-4. Big Data Image Fusion Results 3 ....................................................... 102
Figure 5-5. Big Data Image Fusion Results 4 ....................................................... 103
### List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT</td>
<td>Computerized axial tomography</td>
</tr>
<tr>
<td>CCD</td>
<td>Charge-coupled device</td>
</tr>
<tr>
<td>CORR</td>
<td>Normalized cross correlation</td>
</tr>
<tr>
<td>Cov</td>
<td>Covariance</td>
</tr>
<tr>
<td>Daub</td>
<td>Daubechies</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete cosine transform</td>
</tr>
<tr>
<td>Dent</td>
<td>Difference Entropy</td>
</tr>
<tr>
<td>DSWT</td>
<td>Discrete stationary wavelet transform</td>
</tr>
<tr>
<td>DTCWT</td>
<td>Dual-tree complex wavelet transform</td>
</tr>
<tr>
<td>DWT</td>
<td>Discrete wavelet transformation</td>
</tr>
<tr>
<td>KLT</td>
<td>Karhunen-Loeve transform</td>
</tr>
<tr>
<td>LMSE</td>
<td>Laplacian Mean Squared Error</td>
</tr>
<tr>
<td>LP</td>
<td>Laplacian pyramid</td>
</tr>
<tr>
<td>MAE</td>
<td>Mean Absolute Error</td>
</tr>
<tr>
<td>MI</td>
<td>Mutual Information</td>
</tr>
<tr>
<td>MOE</td>
<td>Measures of Effectiveness</td>
</tr>
<tr>
<td>MRI</td>
<td>Magnetic resonance imaging</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean-Squared Error</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>PFF</td>
<td>Percentage Fit Error</td>
</tr>
<tr>
<td>PSNR</td>
<td>Peak signal to noise Ratio</td>
</tr>
<tr>
<td>QI</td>
<td>Universal Image Quality Index</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root Mean Square Error</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal to noise Ratio</td>
</tr>
<tr>
<td>SC</td>
<td>Structural content</td>
</tr>
<tr>
<td>Ssimval</td>
<td>Structural Similarity Index (SSIM) for measuring image quality</td>
</tr>
</tbody>
</table>
List of Symbols

$I_1$ .........................First Image to be fused
$I_2$ ..........................Second Image to be fused
$I_{\text{fused}}$ ..........................Fused Image
1. Introduction

Chapter 1

Image fusion is the process of merging useful information from two or more images to a single image, and it is a form of data fusion where the signal here is represented by the image dataset. There are five main categories for image fusion: [1] [2]

1. Multisensor fusion
2. Multiview fusion
3. Multimodal fusion
4. Multifocus fusion
5. Multitemporal fusion

Multiview fusion is the process of integrating images taken of the same scene with different cameras with different views or same camera with several pictures with different views taken of the same scene to produce a more informative fused single image. An example would be trying to create a panoramic picture of the scene.

Multimodal fusion is the process of integrating different modal of images taken of the same scene in one single fused image; this category is mainly used for medical imaging.
An example would be fusing a Magnetic Resonance Imaging (MRI) with a Computerized Axial Tomography scan (CAT scan) to produce a fused image that contains both of these relevant images. Multi-focus fusion is the process of integrating images taken of the same scene but with different focuses and example would be an image that focuses more on the background of the image while the other focuses more on the foreground, then the resulting fused image would integrating both the background and foreground information into a single final image. Another example would be using image fusion for images restoration such as images that were taken by red light traffic camera and produce a final image with a car plate number. Multitemporal fusion is the process of integrating useful information from different images of the same scene but at different time values. An example would be trying to detect changes in a scene by removing redundant information and fusing only relevant new information. Finally, the multisensor fusion and that will be our main focus in our work, which is integrating images taken of the same scene but with different sensors such as the case with satellite images that are taken with different color sensors and then fused to produce a single final image with complete relevant information of the scene.
1.1. Astronomical images

Astronomical images are taken by space observatories spread across the globe using telescopes equipped with electronic detectors such as Charge-Coupled Device (CCD). The astronomical images are usually saved using the file extension FITS (Flexible Image Transport System). FITS files format saves very useful information regarding the astronomical images in the image metadata called FITS header as shown in Figure 1.2 and a sample astronomical FITS image is shown in Figure 1.1.

The astronomical images used in this work was provided by Dr. Vincent Schmidt from the Air Force Research Laboratory (AFRL) at Wright-Patterson Air Force Base in Dayton.

![Figure 1-1. Astronomical FITS image](image-url)
Figure 1-2. FITS header

```plaintext
SIMPLE = T
BITPIX = 16 / 8 unsigned int, 16 & 32 int, -32 & -64 real
NAXIS = 2 / number of axes
NAXIS1 = 3352 / fastest changing axis
NAXIS2 = 2532 / next to fastest changing axis
XORGSUB = 0 / Subframe X position in binned pixels
EXPTIME = 20.000000000000000 / Exposure time in seconds
OBJCTRA = '19 15 45' / Nominal Right Ascension of center of image
OBJCTHA = '1.2827' / Nominal hour angle of center of image
OBJCTAZ = '205.6181' / Nominal azimuth of center of image
EXPOSURE = 20.000000000000000 / Exposure time in seconds
SET-TEMP = -8.000000000000000 / CCD temperature setpoint in C
XBINNING = 1 / Binning factor in width
BZERO = 32768.000000000000 / physical = BZERO + BSCALE*array_value
DATE-OBS = '2013-06-29T01:57:21' / YYYY-MM-DDThh:mm:ss observation start, UT
BSCALE = 1.000000000000000 / physical = BZERO + BSCALE*array_value
SITELAT = '59 49 23' / Latitude of the imaging location
YORGSUB = 0 / Subframe Y position in binned pixels
FILTER = 'Blue' / Filter used when taking image
IMAGETYP = 'Light Frame' / Type of image
FOCUSPOS = 23731 / Focuser position in steps
FOCUSSZ = 1.000000000000000 / Focuser step size in microns
FOCUSTEM = 22.000000000000000 / Focuser temperature in deg C
OBJECT = 'Autosave Image' / Name of object being imaged
SITELONG = '184 02 57' / Longitude of the imaging location
OBJCTDEC = '59 09 44' / Nominal Declination of center of image
OBJCTALT = '40.7305' / Nominal altitude of center of image
JD = 2450564.5676041565 / Julian Date at start of exposure
JD-HELIO = 2450564.580859118 / Heliocentric Julian Date at exposure midpoint
AIRMASS = 1.9286297942095079 / Relative optical path length through atmosphere
FOCALLEN = 2939.0900000000000 / Focal length of telescope in mm
APTDIAM = 432.00000000000000 / Aperture diameter of telescope in mm
APTPAREA = 136341.22091133319 / Aperture area of telescope in mm²
EGAIN = 0.3799999952312942 / Electronic gain in e- / ADU
SUNCREAT = 'Maxim DL Version 5.23 130126 1x53' / Name of software that created the image
CALSTAT = 'D' / Calibration
XPIKSZ = 5.400000000000000 / Pixel width in microns (after binning)
CCD-TEMP = -7.715799152448377 / CCD temperature at start of exposure in C
YPIKSZ = 5.400000000000000 / Pixel height in microns (after binning)
SBSTDEV = 'SBFITSFILE Version 1.0' / Version of SBFITSFILE standard in effect
HISTORY Dark subtraction (Simple Auto-dark)
YBINNING = 1 / Binning factor in height
PEDESTAL = -100 / Correction to add for zero-based ADU
TELESCOPE = 'PlaneWave CDK-17' / telescope used to acquire this image
INSTRUME = 'SBIG STF-8300 CCD Camera'
SERVER = 'AFRL / 711 HPW / RHCV at WPAFB'
NOTES = '
FLIPSTAT = '
SWOWNER = 'Vincent A Schmidt' / Licensed owner of software
```

Figure 1-2. FITS header
1.2. Thesis outline

This thesis is organized as follows:

- **Chapter two**: This chapter provides a detailed study of ten different methods of Image fusion techniques and their theoretical background and their step-by-step detailed algorithmic implementation and then introduces thirteen measures of effectiveness with their theoretical background and their step-by-step detailed algorithmic implementation.

- **Chapter three**: This chapter provides an introductory study of parallel computing, major concepts in their implementation and discusses issues facing the parallel implementation. Then, a parallel computing implementation of the Image fusion is applied using two methods and the results are discussed.

- **Chapter four**: This chapter explores the emerging field of cloud computing and its advantages over regular processing. Then, a cloud computing implementation of the Image fusion is applied using two methods and the results are discussed.

- **Chapter five**: This chapter explores other implementation for image fusion especially in the field of Space Situational Awareness (SSA), image characterization and big data image fusion.

- **Chapter six**: This chapter then concludes this thesis study and proposed future work to be done in this field.
2. Image Fusion

Chapter 2

In this chapter, this work will explore Image fusion and different Image fusion algorithms and their detailed implementation, and also we will discuss various measures of effectiveness to test Image fusion quality and how to write their code in MATLAB and Python and then we will discuss the results of our findings.

2.1. Image Fusion Techniques

In our work we focused on the following image fusion techniques:

1. Select Minimum
2. Select Maximum
3. Average Method
4. Principal Component Analysis (PCA)
5. Discrete Cosine transform & Laplacian Pyramid (DCT LP)
6. Discrete Wavelet Transformation -- using Daubechies (DWT Daub)
7. Discrete Wavelet Transformation -- using Haar (DWT Haar)
8. Discrete Stationary Wavelet Transform (DSWT) [1 level]
9. Discrete Stationary Wavelet Transform (DSWT) [2 levels]
10. Dual-Tree Complex Wavelet Transform (DTCWT)

2.1.1. Select Minimum

In the select minimum image fusion algorithm, it compares every corresponding pixel intensity value from each input image and selects the minimum value and that value in inserted in the final fused image [3]

\[ I_{\text{used}}(i,j) = \min_{0 \leq i \leq m, 0 \leq j \leq n} [I_1(i,j), I_2(i,j)] \] (2-1)

Where \( I_{\text{used}} \) is the final fused image and \( I_1 \) and \( I_2 \) are the two input images

\( m \) is the height of the image in terms of pixels, which is also the number of rows in that image

\( n \) is the width of the image in terms of pixels, which is also the number of columns in that image
Steps to perform select minimum algorithm:

1) Compare the intensity value at pixel location \((i,j)\) in all input images

2) Assign the minimum intensity value to the corresponding pixel of the output fused image

3) Repeat step 1 and 2 for all pixel locations

4) After going through the entire images pixel locations, the resulting matrix would be the final fused image.

2.1.2. Select Maximum

In the select maximum image fusion algorithm, it compares every corresponding pixel intensity value from each input image and selects the maximum value and that value in inserted in the final fused image [3]

\[
I_{fused}(i,j) = \max_{0 \leq i \leq m, \ 0 \leq j \leq n} [I_1(i,j), I_2(i,j)]
\]  

(2-2)

Where \(I_{fused}\) is the final fused image and \(I_1\) and \(I_2\) are the two input images.
m is the height of the image in terms of pixels, which is also the number of rows in that image.

n is the width of the image in terms of pixels, which is also the number of columns in that image.

1) Compare each intensity value for each pixel location of the corresponding Image
2) Select the maximum value for the resulting fused Image

Figure 2-2. Select Maximum

Steps to perform select maximum algorithm:

1) Compare the intensity value at pixel location (i,j) in all input images

2) Assign the maximum intensity value to the corresponding pixel of the output fused image

3) Repeat step 1 and 2 for all pixel locations

4) After going through the entire images pixel locations, the resulting matrix would be the final fused image.
2.1.3. Average Method

In the average image fusion algorithm, which is a more advanced version than the select maximum and minimum, it calculates the average of the sum of each corresponding pixel intensity value from each input image and inserts the result in the corresponding location in the final fused image.

\[ I_{\text{fused}}(i,j) = \frac{I_1(i,j) + I_2(i,j)}{2}, \text{ for } 0 \leq i \leq m \text{ and } 0 \leq j \leq n \]  \hspace{1cm} (2-3)

Where \( I_{\text{fused}} \) is the final fused image and \( I_1 \) and \( I_2 \) are the two input images.

\( m \) is the height of the image in terms of pixels, which is also the number of rows in that image.

\( n \) is the width of the image in terms of pixels, which is also the number of columns in that image.

\[ \text{Image 1} \]
1) using the intensity value at each pixel location of the corresponding Image
2) calculate the average intensity value and insert that value in the resulting fused image.

\[ \text{Image 2} \]

\[ \text{Fused Image} \]

Figure 2-3. Average Method
Steps to perform average algorithm:

1) extract the intensity value at pixel location (i,j) in all input images
2) compute the average of these extracted intensity values from step 1 and assign it to the corresponding pixel of the output fused image
3) Repeat step 1 and 2 for all pixel locations
4) After going through the entire images pixel locations, the resulting matrix would be the final fused image.

2.1.4. Principal component analysis

The first step in applying the principal component analysis image fusion algorithm is to reorganize the image dataset to be able to be used in the calculation [4]. The goal is to convert the input images matrix $I_1$ and $I_2$ of a dimension $m$ by $n$ to one by $m$ multiplied $n$ matrix, essentially a one column matrix. This needs to be repeated for all input images. The next step is to calculate the covariance between the two new datasets $X$ and $Y$ created from the input images matrices $I_1$ and $I_2$.

$$C = cov(X, Y) = E[(X - E[X])(Y - E[Y])]\quad (2-4)$$

Where $E[X]$ and $E[Y]$ are the expected value of $X$ and $Y$ respectively.

So to find the covariance, we first need to find the $E[X]$ and $E[Y]$ which is basically calculating the empirical mean of each datasets $X$ and $Y$ and then subtracting these values from the $X$ and $Y$ datasets to get the deviations from the mean which essentially means $(X-E[X])$ and $(Y-E[Y])$. 

11
Next using the resulting matrix \( C \) from the covariance, we try to find the
Eigenvector and Eigenvalue such that

\[
Cv = Dv \quad (2-5)
\]

Where \( C \) is the covariance matrix, \( v \) is the Eigenvector matrix and \( D \) is the
Eigenvalue.

After getting the Eigenvector \( V \) and Eigenvalue \( D \) of matrix \( C \), we sort them by
decreasing Eigenvalue. The Resulting \( V \) and \( D \) are of dimension two by two, and the first
column of \( V \) will correspond to larger Eigenvalue and then can be used to compute the
final PCA value (weighted value) of each picture \( p_1 \) and \( p_2 \).

\[
p_1 = \frac{v(1)}{\sum v}, \quad p_2 = \frac{v(2)}{\sum v} \quad (2-6)
\]

Where \( v \) is the Eigenvector matrix.

Finally to get the fused image we multiply each weighted average to the original
picture and sum the result together to get the final fused image

\[
I_{fused} = p_1 I_1 + p_2 I_2 \quad (2-7)
\]

Where \( I_{fused} \) is the final fused image and \( I_1 \) and \( I_2 \) are the two input images.
Steps to perform PCA algorithm:

1) Generate the column vectors, respectively, from the input images matrices

2) Calculate the covariance matrix of the column vectors formed in step 1.

3) Calculate the Eigen vectors and Eigen values of that covariance matrix

4) Normalize the column vector corresponding to the larger Eigenvalue
5) Create the weight values datasets which are the values of the normalized Eigenvector from step 4 divided by the mean of the Eigenvector

6) Multiply the weight values datasets from step 5 with each pixel of the input images and summing them together to get the final fused image matrix

2.1.5. Discrete wavelet transformation using Daubechies

In the discrete wavelet transformation using Daubechies algorithm, there are three main steps to be performed to successfully fuse the input images. Decompose, fuse and reconstruct but before starting the user needs to choose how many levels, k, will the algorithm implement [5]. This needs to be predetermined before starting the algorithm

a) Image decomposition

The first step in applying discrete wavelet transform image fusion is decomposing each input image. By treating images as a signal and decomposing them using a Daubechies wavelet transform based low pass filter ,g(n) and a Daubechies wavelet transform based high pass filter h(n)

![Figure 2-5. Daubechies wavelet transform](image)

14
And then repeat that decomposition steps $K$ times to reach to the final decomposition level. This series of decomposition is represented as a tree is known as a filter bank.

![Diagram of filter bank](image)

Figure 2.6. Daubechies filter bank

b) Fuse the multiscale image

Secondly, after getting the final decomposition level for each level image, we will end up with two coefficients for each image; the approximation coefficient and the detail coefficients. So the next step would be to fuse them and to do that there are three different fusing methods to use

1) Select Maximum

2) Select minimum

3) Average

In the Select Maximum, we compare each matrix element value in a matrix and choose the maximum value while in the minimum we choose the smaller out of the two values and finally the average method we calculate the average of both values.

c) Image reconstruction of the fused image
Finally, after getting the fused approximation coefficient and the detail coefficients values, we can get the final fused image by applying inverse wavelet transform on these values k times to reach the original starting level.

![Diagram showing the steps of DWT using Daubechies](image)

*Figure 2-7. DWT using Daubechies*

Steps to perform discrete wavelet transformation -- Daubechies algorithm:

1) Choose the level, k, of decomposition required to apply to the images for fusion

2) Decompose the input images using discrete wavelet transformation with Daubechies wavelet filter

3) Repeat step 2 K times, decided upon from step 1, to reach the targeted level of decomposition

4) After obtaining the final level of decomposed images matrices from step 3, a fused decomposed matrix is created using one of the following methods
   a. Average
   b. Select Maximum
   c. Select Minimum
5) Reconstructure a matrix by applying inverse discrete wavelet transformation with Daubechies wavelet filter to the fused decomposed matrix obtained in step 4

6) Repeat Step K times, decided upon from step 1, till we reach the original level.

7) The resulting matrix from step 6 will be the final fused image

2.1.6. Discrete wavelet transformation using Haar

In the discrete wavelet transformation using Haar algorithm, there are three main steps to be performed to successfully fuse the input images [6] [7]. Decompose, fuse and reconstruct but before starting the user needs to choose how many levels, k, will the algorithm implement. This needs to be predetermined before starting the algorithm

a) Image decomposition

The first step in applying discrete wavelet transform image fusion is decomposing each input image. By treating images as a signal and decomposing them using a Haar wavelet transform based low pass filter, g(n) and a Haar wavelet transform based high pass filter h(n)

![Haar wavelet transform](image)

*Figure 2-8. Haar wavelet transform*
And then repeat that decomposition steps K times to reach to the final decomposition level. This series of decomposition is represented as a tree is known as a filter bank.

![Image of Haar filter bank]

Figure 2.9. Haar filter bank

b) Fuse the multiscale image

Secondly, after getting the final decomposition level for each level image, we will end up with two coefficients for each image; the approximation coefficient and the detail coefficients. So the next step would be to fuse them and to do that there are three different fusing methods to use

1) Select Maximum

2) Select minimum

3) Average

In the Select Maximum, we compare each matrix element value in a matrix and choose the maximum value while in the minimum we choose the smaller out of the two values and finally the average method we calculate the average of both values.

c) Image reconstruction of the fused image
Finally, after getting the fused approximation coefficient and the detail coefficients values, we can get the final fused image by applying inverse wavelet transform on these values \( k \) times to reach the original starting level.

Figure 2.10. DWT using Haar

Steps to perform discrete wavelet transformation -- Haar algorithm:

1) Choose the level, \( k \), of decomposition required to apply to the images for fusion

2) Decompose the input images using discrete wavelet transformation with Haar wavelet filter

3) Repeat step 2 \( K \) times, decided upon from step 1, to reach the targeted level of decomposition

4) After obtaining the final level of decomposed images matrices from step 3, a fused decomposed matrix is created using one of the following methods
   
   a. Average
   
   b. Select Maximum
   
   c. Select Minimum
5) Reconstruct a matrix by applying inverse discrete wavelet transformation with Haar wavelet filter to the fused decomposed matrix obtained in step 4

6) Repeat Step K times, decided upon from step 1, till we reach the original level.

7) The resulting matrix from step 6 will be the final fused image

2.1.7. Discrete stationary wavelet transform [1 Level]

In the discrete stationary wavelet transform algorithm, there are three main steps to be performed to successfully fuse the input images [8]. Decompose, fuse and reconstruct but before starting the user needs to choose how many levels, k, will the algorithm implement. This need to be predetermined before starting the algorithm and, in this case, K should be equal one.

a) Image decomposition

The first step in applying discrete stationary wavelet transform image fusion is decomposing each input image. By treating images as a signal and decomposing them using a low pass filter, g(n) and a high pass filter h(n).

\[ g[n] \xrightarrow{\uparrow 2} g_{j-1}[n] \]

\[ h[n] \xrightarrow{\uparrow 2} h_{j-1}[n] \]

*Figure 2-11. Discrete stationary wavelet transform*
And then repeat that decomposition steps K times to reach to the final
decomposition level and generating new low and high filter in each step. This series of
decomposition is represented as a tree is known as a filter bank.

Figure 2-12. DSWT filter bank

b) Fuse the multiscale image

Secondly, after getting the final decomposition level for each level image, we will
end up with two coefficients for each image; the approximation coefficient and the detail
coefficients. So the next step would be to fuse them and to do that there are three
different fusing methods to use

1) Select Maximum

2) Select minimum

3) Average

In the Select Maximum, we compare each matrix element value in a matrix and
choose the maximum value while in the minimum we choose the smaller out of the two
values and finally the average method we calculate the average of both values.

c) Image reconstruction of the fused image
Finally, after getting the fused approximation coefficient and the detail coefficients values, we can get the final fused image by applying inverse discrete stationary wavelet transform on these values k times to reach the original starting level.

![Diagram of DSWT (1 Level)](image)

**Figure 2-13. DSWT (1 Level)**

Steps to perform discrete stationary wavelet transform –1 level algorithm:

1) Choose the level, k, of decomposition required to apply to the images for fusion and, in this case, k=1

2) Decompose the input images using discrete stationary wavelet transform

3) Repeat step 2 K times, decided upon from step 1, to reach the targeted level of decomposition

4) After obtaining the final level of decomposed images matrices from step 3, a fused decomposed matrix is created using one of the following methods
   a. Average
   b. Select Maximum
   c. Select Minimum
5) Reconstruct a matrix by applying inverse discrete stationary wavelet transform to the fused decomposed matrix obtained in step 4

6) Repeat Step K times, decided upon from step 1, till we reach the original level.

7) The resulting matrix from step 6 will be the final fused image

2.1.8. Discrete stationary wavelet transform [2 Levels]

In the discrete stationary wavelet transform algorithm, there are three main steps to be performed to successfully fuse the input images [8]. Decompose, fuse and reconstruct but before starting the user needs to choose how many levels, k, will the algorithm implement. This need to be predetermined before starting the algorithm and, in this case, K should be equal two.

a) Image decomposition

The first step in applying discrete stationary wavelet transform image fusion is decomposing each input image. By treating images as a signal and decomposing them using a low pass filter, \( g(n) \) and a high pass filter \( h(n) \).

\[
\begin{align*}
g[n] & \quad \xrightarrow{\uparrow 2} \quad g_{j-1}[n] \\
h_{j}[n] & \quad \xrightarrow{\uparrow 2} \quad h_{j-1}[n]
\end{align*}
\]

*Figure 2-14. Discrete stationary wavelet transform (2 levels)*
And then repeat that decomposition steps K times to reach to the final decomposition level and generating new low and high filter in each step. This series of decomposition is represented as a tree is known as a filter bank.

![Filter Bank Diagram](image)

*Figure 2-15. DSWT filter bank (2 levels)*

b) Fuse the multiscale image

Secondly, after getting the final decomposition level for each level image, we will end up with two coefficients for each image; the approximation coefficient and the detail coefficients. So the next step would be to fuse them and to do that there are three different fusing methods to use

1) Select Maximum

2) Select minimum

3) Average

In the Select Maximum, we compare each matrix element value in a matrix and choose the maximum value while in the minimum we choose the smaller out of the two values and finally the average method we calculate the average of both values.

c) Image reconstruction of the fused image
Finally, after getting the fused approximation coefficient and the detail coefficients values, we can get the final fused image by applying inverse discrete stationary wavelet transform on these values k times to reach the original starting level.

**Figure 2-16. DSWT (2 Levels)**

Steps to perform discrete stationary wavelet transform –2 level algorithms:

1) Choose the level, k, of decomposition required to apply to the images for fusion and, in this case, k=2

2) Decompose the input images using discrete stationary wavelet transform

3) Repeat step 2 K times, decided upon from step 1, to reach the targeted level of decomposition

4) After obtaining the final level of decomposed images matrices from step 3, a fused decomposed matrix is created using one of the following methods

   a. Average

   b. Select Maximum

   c. Select Minimum
5) Reconstruct a matrix by applying inverse discrete stationary wavelet transform to the fused decomposed matrix obtained in step 4

6) Repeat Step K times, decided upon from step 1, till we reach the original level.

7) The resulting matrix from step 6 will be the final fused image

2.1.9. Dual-tree complex wavelet transform

In the dual-tree complex wavelet transform algorithm, there are three main steps to be performed to successfully fuse the input images [9] [10]. Decompose, fuse and reconstruct but before starting the user needs to choose how many levels, k, will the algorithm implement. This need to be predetermined before starting the algorithm and, in this case, K should be equal one.

a) Image decomposition

The first step in applying dual-tree complex wavelet transform image fusion is decomposing each input image. To calculate the complex transform decomposition the images used a signal is decomposed using two separate discrete wavelet transformation decompositions (Tree a and Tree b) each with a different low pass filter,\( g(n) \) and high pass filter \( h(n) \).
And then repeat that decomposition steps $K$ times to reach to the final decomposition level and generating new low and high filter in each step. This series of decomposition is represented as a tree is known as a filter bank.

b) Fuse the multiscale image

Secondly, after getting the final decomposition level for each level image, we will end up with two coefficients for each image; the approximation coefficient and the detail coefficients. So the next step would be to fuse them and to do that there are three different fusing methods to use.
1) Select Maximum

2) Select minimum

3) Average

In the Select Maximum, we compare each matrix element value in a matrix and choose the maximum value while in the minimum we choose the smaller out of the two values and finally the average method we calculate the average of both values.

c) Image reconstruction of the fused image

Finally, after getting the fused approximation coefficient and the detail coefficients values, we can get the final fused image by applying inverse dual-tree complex wavelet transform on these values k times to reach the original starting level.

---

**Figure 2-19. DTCWT**
Steps to perform dual-tree complex wavelet transform algorithm:

1) Choose the level, k, of decomposition required to apply to the images for fusion

2) Decompose the input images using dual-tree complex wavelet transform

3) Repeat step 2 K times, decided upon from step 1, to reach the targeted level of decomposition

4) After obtaining the final level of decomposed images matrices from step 3, a fused decomposed matrix is created using one of the following methods
   a. Average
   b. Select Maximum
   c. Select Minimum

5) Reconstruct a matrix by applying inverse dual-tree complex wavelet transform to the fused decomposed matrix obtained in step 4

6) Repeat Step K times, decided upon from step 1, till we reach the original level.

7) The resulting matrix from step 6 will be the final fused image

2.1.10. Discrete cosine transform based Laplacian pyramid

In the discrete cosine transform based Laplacian pyramid algorithm, there are three main steps to be performed to successfully fuse the input images [4] [11]. Decompose, fuse and reconstruct but before starting the user needs to choose how many levels, k, will the algorithm implement. This needs to be predetermined before starting the algorithm


a) Image decomposition

The first step in applying discrete cosine transform based Laplacian pyramid image fusion is to decompose each input image using discrete cosine transform (DCT) and then we need to construct a Laplacian pyramid representing how many levels, k, the algorithm would apply.

The two-dimensional discrete cosine transform $Z(u,v)$ of an image $I(i,j)$ of dimensional size is $m$ by $n$ is calculated using the following equation

$$Z(u,v) = \alpha(u) \alpha(v) \sum_{i=0}^{m} \sum_{j=0}^{n} I(i,j) \cos \left( \frac{\pi(2i+1)u}{2m} \right) \cos \left( \frac{\pi(2j+1)v}{2n} \right)$$

Where

$$\alpha(u) = \begin{cases} \frac{1}{\sqrt{m}}, & u = 0 \\ \frac{2}{\sqrt{m}}, & 1 \leq u \leq m \end{cases}$$

$$\alpha(v) = \begin{cases} \frac{1}{\sqrt{n}}, & v = 0 \\ \frac{2}{\sqrt{n}}, & 1 \leq v \leq n \end{cases}$$

And then repeat that decomposition steps $K$ times to reach to the final decomposition level. This series of decomposition is represented as a pyramid and is known as a Laplacian pyramid.
b) Fuse the multiscale image

Secondly, after getting the final decomposition level for each image, we will end up with a multiscale image for each image. So the next step would be to fuse them and to do that there are three different fusing methods to use

1) Select Maximum

2) Select minimum

3) Average

In the Select Maximum, we compare each matrix element value in a matrix and choose the maximum value while in the minimum we choose the smaller out of the two values and finally the average method we calculate the average of both values.

c) Image reconstruction of the fused image
Finally, after getting the fused multiscale image, we can get the final fused image by applying inverse discrete cosine transform on the image k times to reach the original starting level which is the base of our Laplacian pyramid model.

Steps to perform discrete cosine transform & Laplacian pyramid algorithm:

1) Choose the level, k, of decomposition required to apply to the images for fusion
2) Decompose the input images using a Laplacian pyramid of the discrete cosine transform
3) Repeat step 2 $K$ times, decided upon from step 1, to reach the targeted level of decomposition
4) After obtaining the final level of decomposed images matrices from step 3, a fused decomposed matrix is created using one of the following methods

*Figure 2-21. DCTLP*
a. Average
b. Select Maximum
c. Select Minimum

5) Reconstruct a matrix by applying the inverse of a Laplacian pyramid of the discrete cosine transform to the fused decomposed matrix obtained in step 4

6) Repeat Step K times, decided upon from step 1, till we reach the original level.

7) The resulting matrix from step 6 will be the final fused image

2.2. Implementation of Fusion techniques using MATLAB

An implementation of these 10 different Image fusion techniques was written using MATLAB R2015B with Image processing and computer vision as an Add-on which includes the following packages

- Image Processing Toolbox
- Signal Processing Toolbox

Please refer to Appendix A and Appendix B, which provide the code and step-by-step explanation of the code being executed. The code accepts Fit/Fits files which are extension name of the astronomical images.

The results from this MATLAB code used a machine specification of

- Intel Core i7-4790 CPU @ 3.60 GHz
- Installed memory (RAM) 16 GB
- Running windows 7 Enterprise 64-bit operating system
2.3. Image Measures Of Effectiveness (MOE)

In order to evaluate the results of the different image fusion algorithms and determine which algorithm gave the best results, a measure of image effectiveness and quality needed to be introduced. The Measures of Effectiveness (MOE) are the measure of image quality in regard to a specific need and algorithm and in this study’s case it would be measured against image fusion. There are several techniques and measures that can be used in evaluating the images. This work uses and implements the following Measures of Effectiveness (MOE) to the final fused image results:

1. Root Mean Square Error
2. Percentage Fit Error
3. Mean-Squared Error
4. Mean Absolute Error
5. Difference Entropy
6. Normalized cross correlation
7. Signal to noise Ratio
8. Peak signal to noise Ratio
9. Structural content
10. Laplacian Mean Squared Error
11. Structural Similarity Index for measuring image quality
12. Mutual Information
13. Universal Image Quality Index
2.3.1. Root Mean Square Error (RMSE)

The Root Mean Square Error (RMSE) is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed as described in Equation 2-9 [12]. The algorithm that gives the minimum Root Mean Square Error (RMSE) value is preferable here.

\[
RMSE = \sqrt{\frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - B_{ij})^2}
\]  

(2-9)

Where m is the height of the image in terms of pixels, which is also the number of rows in that image

n is the width of the image in terms of pixels, which is also the number of columns in that image

\(A_{ij}\) is the pixel intensity value of the test image (perfect image) at the location i and j

\(B_{ij}\) is the pixel intensity value of the fused image at location i and j.
2.3.2. Percentage Fit Error (PFE)

The Percentage Fit Error (PFE) is calculated using the relation described in Equation 2-10 [12]. The algorithm that gives the minimum Percentage Fit Error (PFE) value is preferable here.

\[
PFE = \frac{\text{norm}(A - B)}{\text{norm}(A)} \times 100
\]  

(2-10)

Where norm is the operator to compute the largest singular value

A is the test image (perfect image)

B is the fused image

2.3.3. Mean-Squared Error (MSE)

The Mean Square Error (MSE) is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed as described in Equation 2-11. The algorithm that gives the minimum Mean Square Error (MSE) value is preferable here.

\[
MSE = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - B_{ij})^2
\]  

(2-11)

Where m is the height of the image in terms of pixels, which is also the number of rows in that image

n is the width of the image in terms of pixels, which is also the number of columns in that image
$A_{ij}$ is the pixel intensity value of the test image (perfect image) at the location $i$ and $j$

$B_{ij}$ is the pixel intensity value of the fused image at location $i$ and $j$.

### 2.3.4. Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed as described in Equation 2-12 [13]. The algorithm that gives the minimum Mean Absolute Error (MAE) value is preferable here.

$$MAE = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} |A_{ij} - B_{ij}|$$  \hspace{1cm} (2-12)

Where $m$ is the height of the image in terms of pixels, which is also the number of rows in that image

$n$ is the width of the image in terms of pixels, which is also the number of columns in that image

$A_{ij}$ is the pixel intensity value of the test image (perfect image) at the location $i$ and $j$

$B_{ij}$ is the pixel intensity value of the fused image at location $i$ and $j$.

### 2.3.5. Difference Entropy

Image entropy (He) is a quantity which is used to describe the amount of information which exists in an image as described by Equation 2-13. While the difference entropy can be
described using Equation 2-14. The algorithm that gives the maximum difference entropy value is preferable here. [14]

\[
He = -\sum_{i=0}^{L} h_f(i) \log_2 h_f(i) \quad (2-13)
\]

Where He is the Entropy of an Image f

L is the number of pixel intensity levels value in an Image f

i is a specific intensity level value

\(h_f(i)\) is the probability of occurrence of a particular gray level i

Difference Entropy (dent)

\[
dent = |He_A - He_B|
\]

\[
dent = | - \sum_{i=0}^{L} h_A(i) \log_2 h_A(i) + \sum_{i=0}^{L} h_B(i) \log_2 h_B(i) |
\quad (2-14)
\]

Where dent is the Difference Entropy

A is the test image (perfect image), and B is the fused image

\(He_A\) is the Entropy of test Image A

\(He_B\) is the Entropy of fused image B

L is the number of pixel intensity levels value in image A and B

i is a specific intensity level value

\(h_A(i)\) is the probability of occurrence of a particular gray level i in image A

\(h_B(i)\) is the probability of occurrence of a particular gray level i in image B
2.3.6. Normalized Cross Correlation (CORR)

Normalized Cross Correlation (CORR) is the measure of correlations between two images datasets as described by Equation 2-15. The algorithm that gives the Normalized Cross Correlation (CORR) value close to the value one is preferable here.

\[
\text{CORR} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij}B_{ij})}{\sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij})^2} \quad (2-15)
\]

Where \( m \) is the height of the image in terms of pixels, which is also the number of rows in that image

\( n \) is the width of the image in terms of pixels, which is also the number of columns in that image

\( A_{ij} \) is the pixel intensity value of the test image (perfect image) at the location \( i \) and \( j \)

\( B_{ij} \) is the pixel intensity value of the fused image at location \( i \) and \( j \).

2.3.7. Signal to Noise Ratio (SNR)

Signal to Noise Ratio (SNR) is a measure that compares the level of a desired signal to the level of background noise as described by Equation 2-16. The algorithm that gives the maximum Signal to Noise Ratio (SNR) value is preferable here.
\[ SNR = 20 \times \log_{10} \left( \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij})^2}{\frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - B_{ij})^2} \right) \] (2-16)

Where \( m \) is the height of the image in terms of pixels, which is also the number of rows in that image.

\( n \) is the width of the image in terms of pixels, which is also the number of columns in that image.

\( A_{ij} \) is the pixel intensity value of the test image (perfect image) at the location \( i \) and \( j \).

\( B_{ij} \) is the pixel intensity value of the fused image at location \( i \) and \( j \).

2.3.8. Peak Signal to Noise Ratio (PSNR)

Peak Signal to Noise Ratio (PSNR) is a measure that compares the maximum level of a desired signal to the level of background noise as described by Equation 2-17. The algorithm that gives the maximum Peak Signal to Noise Ratio (PSNR) value is preferable here.

\[ PSNR = 20 \times \log_{10} \left( \frac{L^2}{\frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - B_{ij})^2} \right) \] (2-17)

Where \( L \) is the number of gray levels in the image.

\( m \) is the height of the image in terms of pixels, which is also the number of rows in that image.

\( n \) is the width of the image in terms of pixels, which is also the number of columns in that image.
$A_{ij}$ is the pixel intensity value of the test image (perfect image) at the location $i$ and $j$.

$B_{ij}$ is the pixel intensity value of the fused image at location $i$ and $j$.

### 2.3.9. Structural Content (SC)

Structural Content (SC) is a measure of similarity between two images as described by Equation 2-18 [15]. The algorithm that gives the Structural Content (SC) value close to the value one is preferable here.

\[
SC = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij})^2}{\sum_{i=1}^{m} \sum_{j=1}^{n} (B_{ij})^2}
\]  

(2-18)

Where SC is the structural content.

- $m$ is the height of the image in terms of pixels, which is also the number of rows in that image.
- $n$ is the width of the image in terms of pixels, which is also the number of columns in that image.

$A_{ij}$ is the pixel intensity value of the test image (perfect image) at the location $i$ and $j$.

$B_{ij}$ is the pixel intensity value of the fused image at location $i$ and $j$. 

41
2.3.10. Laplacian Mean Squared Error (LMSE)

Laplacian Mean Squared Error (LMSE) that is based on the importance of edges measurement as described by Equation 2-19. The algorithm that gives the minimum Laplacian Mean Squared Error (LMSE) value is preferable here [16].

\[
LMSE = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} (\nabla^2 A_{ij} - \nabla^2 B_{ij})^2}{\sum_{i=1}^{m} \sum_{j=1}^{n} (\nabla^2 A_{ij})^2} \quad (2-19)
\]

Where m is the height of the image in terms of pixels, which is also the number of rows in that image

n is the width of the image in terms of pixels, which is also the number of columns in that image

\(A_{ij}\) is the pixel intensity value of the test image (perfect image) at the location \(i\) and \(j\)

\(B_{ij}\) is the pixel intensity value of the fused image at location \(i\) and \(j\).

And the Laplacian operator \(\nabla^2\) is defined by the following expression

\[
\nabla^2 u = \left(\frac{d^2 u}{d^2 x} + \frac{d^2 u}{d^2 y}\right) \quad (2-20)
\]

Let \(u\) be defined as a function of \((x,y)\)
2.3.11. Structural Similarity Index Measurement (SSIM)

The Structural Similarity Index Measurement (SSIM) is a method for predicting the perceived quality of an image as described by Equation 2-21. The algorithm that gives the structural similarity index (SSIM) value close to the value one is preferable here [16].

\[
SSIM = \frac{(2\mu_A \mu_B + C_1)(2\sigma_{AB} + C_2)}{\left(\mu_A^2 + \mu_B^2 + C_1\right)\left(\sigma_A^2 + \sigma_B^2 + C_2\right)} \quad (2-21)
\]

Equation 2-1

Where

\[
\mu_A = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}
\]

\[
\mu_B = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} B_{ij}
\]

\[
\sigma_A^2 = \frac{1}{mn - 1} \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - \mu_A)^2
\]

\[
\sigma_B^2 = \frac{1}{mn - 1} \sum_{i=1}^{m} \sum_{j=1}^{n} (B_{ij} - \mu_B)^2
\]

\[
\sigma_{AB} = \frac{1}{mn - 1} \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - \mu_A)(B_{ij} - \mu_B) \quad (2-22)
\]

Where m is the height of the image in terms of pixels, which is also the number of rows in that image.
n is the width of the image in terms of pixels, which is also the number of columns in that image

\( A_{ij} \) is the pixel intensity value of the test image (perfect image) at the location \( i \) and \( j \)

\( B_{ij} \) is the pixel intensity value of the fused image at location \( i \) and \( j \).

2.3.12. Mutual Information (MI)

Mutual Information (MI) is a measure of similarity between two images as described by Equation 2-23. The algorithm that gives the maximum Mutual Information (MI) value is preferable here [17].

\[
MI = \sum_{i=1}^{m} \sum_{j=1}^{n} h_{AB}(i,j) \log_2 \left( \frac{h_{AB}(i,j)}{h_A(i,j)h_B(i,j)} \right)
\]  \hspace{1cm} (2-23)

Where \( m \) is the height of the image in terms of pixels, which is also the number of rows in that image.

n is the width of the image in terms of pixels, which is also the number of columns in that image.

\( A_{ij} \) is the pixel intensity value of the test image (perfect image) at the location \( i \) and \( j \)

\( B_{ij} \) is the pixel intensity value of the fused image at location \( i \) and \( j \).
2.3.13. Universal Image Quality Index (QI)

Universal Image Quality Index (QI) is an objective image quality index, which is applicable to various image processing applications as described by Equation 2-24. [15]

\[
QI = \frac{4\sigma_{AB}(\mu_A+\mu_B)}{(\sigma_A^2+\sigma_B^2)(\mu_A^2+\mu_B^2)} \quad (2-24)
\]

Where

\[
\mu_A = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}
\]

\[
\mu_B = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} B_{ij}
\]

\[
\sigma_A^2 = \frac{1}{mn-1} \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - \mu_A)^2
\]

\[
\sigma_B^2 = \frac{1}{mn-1} \sum_{i=1}^{m} \sum_{j=1}^{n} (B_{ij} - \mu_B)^2
\]

\[
\sigma_{AB} = \frac{1}{mn-1} \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - \mu_A)(B_{ij} - \mu_B) \quad (2-25)
\]

Where \(m\) is the height of the image in terms of pixels, which is also the number of rows in that image.

\(n\) is the width of the image in terms of pixels, which is also the number of columns in that image.

\(A_{ij}\) is the pixel intensity value of the test image (perfect image) at the location \(i\) and \(j\).
$B_{ij}$ is the pixel intensity value of the fused image at location i and j.

2.4. Implementation of Image measures of effectiveness

An implementation of these 13 different measures of effectiveness techniques was written using MATLAB R2015B with Image processing and computer vision as an Add-on which includes the following packages

- Image Processing Toolbox
- Signal Processing Toolbox

Please refer to Appendix A and Appendix C, which provide the code and step-by-step explanation of the code being executed. The code accepts Fit/Fits files which are extension name of the astronomical images.

The results from this MATLAB code used a machine specification of

- Intel Core i7-4790 CPU @ 3.60 GHz
- Installed memory (RAM) 16 GB
- Running windows 7 Enterprise 64-bit operating system

2.5. Results using MATLAB

2.5.1. Image Quality Results

After running the MATLAB code of Measures of Effectiveness (MOE) on the resulting fused images obtained from the MATLAB code of the different image fusion algorithms, results obtained are shown from Figure 2.22 till Figure 2.34.
Figure 2-22. Root Mean Square Error

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
<th>DCT_LP</th>
<th>DSWT_1L</th>
<th>DSWT_2L</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
<th>DTCWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root mean square error</td>
<td>107.934</td>
<td>107.934</td>
<td>74.002</td>
<td>80.371</td>
<td>77.502</td>
<td>77.834</td>
<td>74.002</td>
<td>74.002</td>
<td>36.699</td>
<td>74.002</td>
</tr>
</tbody>
</table>

Figure 2-23. Percentage Fit Error

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
<th>DCT_LP</th>
<th>DSWT_1L</th>
<th>DSWT_2L</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
<th>DTCWT</th>
</tr>
</thead>
</table>
Figure 2-24. Mean Squared Error

Figure 2-25. Mean Absolute Error
Figure 2-26. Difference Entropy

<table>
<thead>
<tr>
<th>Method</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
<th>DCT_LP</th>
<th>DSWT_1L</th>
<th>DSWT_2L</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
<th>DTCWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difference entropy</td>
<td>8.09E-06</td>
<td>8.09E-06</td>
<td>8.09E-06</td>
<td>5.20E-06</td>
<td>5.20E-06</td>
<td>5.20E-06</td>
<td>8.09E-06</td>
<td>8.09E-06</td>
<td>8.09E-06</td>
<td>8.09E-06</td>
</tr>
</tbody>
</table>

Figure 2-27. Correlation

<table>
<thead>
<tr>
<th>Method</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
<th>DCT_LP</th>
<th>DSWT_1L</th>
<th>DSWT_2L</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
<th>DTCWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation</td>
<td>0.952</td>
<td>0.952</td>
<td>0.974</td>
<td>0.970</td>
<td>0.972</td>
<td>0.972</td>
<td>0.974</td>
<td>0.974</td>
<td>0.994</td>
<td>0.974</td>
</tr>
</tbody>
</table>
Figure 2-28. Signal to Noise Ratio

Figure 2-29. Peak Signal to Noise Ratio
Figure 2-30. Structural Content

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
<th>DCT_LP</th>
<th>DSWT_1L</th>
<th>DSWT_2L</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
<th>DTCWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>structural content</td>
<td>0.755</td>
<td>0.755</td>
<td>0.991</td>
<td>0.971</td>
<td>0.978</td>
<td>0.976</td>
<td>0.991</td>
<td>0.991</td>
<td>1.009</td>
<td>0.991</td>
</tr>
</tbody>
</table>

Figure 2-31. Laplacian Mean Squared Error

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
<th>DCT_LP</th>
<th>DSWT_1L</th>
<th>DSWT_2L</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
<th>DTCWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laplacian Mean Squared Error</td>
<td>0.998</td>
<td>0.998</td>
<td>0.507</td>
<td>0.964</td>
<td>0.858</td>
<td>0.870</td>
<td>0.507</td>
<td>0.507</td>
<td>0.125</td>
<td>0.507</td>
</tr>
</tbody>
</table>
### Structural Similarity Index (SSIM)

![SSIM Graph](image)

<table>
<thead>
<tr>
<th>Method</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
<th>DCT_LP</th>
<th>DSWT_1L</th>
<th>DSWT_2L</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
<th>DTCWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Similarity Index</td>
<td>0.410</td>
<td>0.410</td>
<td>0.652</td>
<td>0.607</td>
<td>0.643</td>
<td>0.643</td>
<td>0.652</td>
<td>0.652</td>
<td>0.919</td>
<td>0.652</td>
</tr>
</tbody>
</table>

*Figure 2-32. Structural Similarity Index*

### Mutual Information

![Mutual Information Graph](image)

<table>
<thead>
<tr>
<th>Method</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
<th>DCT_LP</th>
<th>DSWT_1L</th>
<th>DSWT_2L</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
<th>DTCWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutual Information</td>
<td>0.093</td>
<td>0.093</td>
<td>0.069</td>
<td>0.119</td>
<td>0.103</td>
<td>0.105</td>
<td>0.069</td>
<td>0.069</td>
<td>0.189</td>
<td>0.069</td>
</tr>
</tbody>
</table>

*Figure 2-33. Mutual Information*
Figure 2-34. Universal Image Quality Index
2.5.2. Image Fusion Results

The fused image results after running the MATLAB code of the different image fusion algorithms are shown from Figure 2.35 till Figure 2.44.

Average Method Image Fusion Results

<table>
<thead>
<tr>
<th>Image A</th>
<th>Image B</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image A" /></td>
<td><img src="image2.png" alt="Image B" /></td>
</tr>
</tbody>
</table>

Fused Image using Average Method

![Fused Image](fusion.png)

<table>
<thead>
<tr>
<th>CORR</th>
<th>dent</th>
<th>LMSE</th>
<th>MAE</th>
<th>MI</th>
<th>MSE</th>
<th>PFE</th>
<th>PSNR</th>
<th>QI</th>
<th>RMSE</th>
<th>SC</th>
<th>SNR</th>
<th>ssimval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg</td>
<td>0.9741</td>
<td>8.09E-06</td>
<td>0.5068</td>
<td>30.2104</td>
<td>0.0695</td>
<td>5.48E+03</td>
<td>22.7911</td>
<td>29.4724</td>
<td>0.6574</td>
<td>74.0019</td>
<td>0.9912</td>
<td>12.8447</td>
</tr>
</tbody>
</table>

*Figure 2.35. Average Method Image Fusion Results*
Discrete cosine transform based Laplacian pyramid Image Fusion Results

Image A

Image B

Fused Image using Discrete cosine transform based Laplacian pyramid

<table>
<thead>
<tr>
<th>DCTLP</th>
<th>CORR</th>
<th>LMSE</th>
<th>MAE</th>
<th>MSE</th>
<th>PFE</th>
<th>PSNR</th>
<th>QI</th>
<th>RMSE</th>
<th>SC</th>
<th>SNR</th>
<th>ssimval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.9698</td>
<td>5.20E-06</td>
<td>0.9643</td>
<td>36.7258</td>
<td>0.1186</td>
<td>6.46E+03</td>
<td>24.7527</td>
<td>29.1138</td>
<td>0.6099</td>
<td>80.3714</td>
<td>0.9711</td>
</tr>
</tbody>
</table>

Figure 2-36. Discrete cosine transform based Laplacian pyramid Image Fusion Results

55
Discrete stationary wavelet transform [1 Level] Image Fusion Results

Figure 2-37. Discrete stationary wavelet transform [1 Level] Image Fusion Results
Discrete stationary wavelet transform [2 Levels] Image Fusion Results

Image A

Image B

Fused Image using Discrete stationary wavelet transform [2 Levels]

<table>
<thead>
<tr>
<th></th>
<th>CORR</th>
<th>COE</th>
<th>LMSE</th>
<th>MAE</th>
<th>MI</th>
<th>MSE</th>
<th>PFE</th>
<th>PSNR</th>
<th>QI</th>
<th>RMSE</th>
<th>SC</th>
<th>SNR</th>
<th>ssimval</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSWT_2L</td>
<td>0.9716</td>
<td>5.20E-06</td>
<td>0.8702</td>
<td>35.2076</td>
<td>0.1047</td>
<td>6.06E+03</td>
<td>23.9713</td>
<td>29.2531</td>
<td>0.6482</td>
<td>77.8340</td>
<td>0.9764</td>
<td>12.4062</td>
<td>0.6426</td>
</tr>
</tbody>
</table>

*Figure 2-38. Discrete stationary wavelet transform [2 Levels] Image Fusion Results*
Dual-tree complex wavelet transforms Image Fusion Results

![Image A](imageA.png) ![Image B](imageB.png)

Fused Image using Dual-tree complex wavelet transform

<table>
<thead>
<tr>
<th>T</th>
<th>CORR</th>
<th>dent</th>
<th>LMSE</th>
<th>MAE</th>
<th>MI</th>
<th>MSE</th>
<th>PFE</th>
<th>PSNR</th>
<th>QI</th>
<th>RMSE</th>
<th>SC</th>
<th>SNR</th>
<th>ssimval</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTCWT</td>
<td>0.9741</td>
<td>8.09E-06</td>
<td>0.5068</td>
<td>30.2104</td>
<td>0.0695</td>
<td>5.48E+03</td>
<td>22.7911</td>
<td>29.4724</td>
<td>0.6574</td>
<td>74.0019</td>
<td>0.9912</td>
<td>12.8447</td>
<td>0.6518</td>
</tr>
</tbody>
</table>

*Figure 2-39. Dual-tree complex wavelet transforms Image Fusion Results*
Discrete wavelet transformation using Daubechies Image Fusion Results

Image A

Image B

Fused Image using Discrete wavelet transformation using Daubechies

<table>
<thead>
<tr>
<th>T</th>
<th>CORR</th>
<th>dent</th>
<th>LMSE</th>
<th>MAE</th>
<th>MI</th>
<th>MSE</th>
<th>PFE</th>
<th>PSNR</th>
<th>QI</th>
<th>RMSE</th>
<th>SC</th>
<th>SNR</th>
<th>ssimval</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWT_Daub</td>
<td>0.9741</td>
<td>8.09E-06</td>
<td>0.5068</td>
<td>30.2104</td>
<td>0.0695</td>
<td>5.48E+03</td>
<td>22.7911</td>
<td>29.4724</td>
<td>0.6574</td>
<td>74.0019</td>
<td>0.9912</td>
<td>12.8447</td>
<td>0.6518</td>
</tr>
</tbody>
</table>

*Figure 2-40. Discrete wavelet transformation using Daubechies Image Fusion Results*
Discrete wavelet transformation using Haar Image Fusion Results

Image A

Image B

Fused Image using Discrete wavelet transformation using Haar

<table>
<thead>
<tr>
<th></th>
<th>CORR</th>
<th>dent</th>
<th>LMSE</th>
<th>MAE</th>
<th>MI</th>
<th>MSE</th>
<th>PFE</th>
<th>PSNR</th>
<th>QI</th>
<th>RMSE</th>
<th>SC</th>
<th>SNR</th>
<th>ssimval</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWT_Haar</td>
<td>0.9741</td>
<td>8.08E-06</td>
<td>0.5068</td>
<td>30.2304</td>
<td>0.0695</td>
<td>5.48E-03</td>
<td>22.7911</td>
<td>29.4724</td>
<td>0.6574</td>
<td>74.0019</td>
<td>0.9912</td>
<td>12.8447</td>
<td>0.6518</td>
</tr>
</tbody>
</table>

*Figure 2-41. Discrete wavelet transformation using Haar Image Fusion Results*
Select Maximum Image Fusion Results

![Image A](image_a.png) ![Image B](image_b.png)

Fused Image using Select Maximum Method

![Fused Image](fused_image.png)

<table>
<thead>
<tr>
<th>.T</th>
<th>CORR</th>
<th>dent</th>
<th>LMSE</th>
<th>NAE</th>
<th>MI</th>
<th>MSE</th>
<th>PFE</th>
<th>PSNR</th>
<th>QI</th>
<th>RMSE</th>
<th>SC</th>
<th>SNR</th>
<th>ssimval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>0.9525</td>
<td>8.09E-06</td>
<td>0.9984</td>
<td>41.0527</td>
<td>0.0930</td>
<td>1.16E+04</td>
<td>33.2415</td>
<td>27.8332</td>
<td>0.4150</td>
<td>107.9341</td>
<td>0.7553</td>
<td>9.5664</td>
<td>0.4100</td>
</tr>
</tbody>
</table>

*Figure 2-42. Select Maximum Image Fusion Results*
Select Minimum Image Fusion Results

Image A

Image B

Fused Image using Select Minimum Method

<table>
<thead>
<tr>
<th>CORR</th>
<th>dnt</th>
<th>LMSE</th>
<th>MAE</th>
<th>MI</th>
<th>MSE</th>
<th>PFE</th>
<th>PSNR</th>
<th>QI</th>
<th>RMSE</th>
<th>SC</th>
<th>SNR</th>
<th>ssimval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0.9525</td>
<td>8.0E-06</td>
<td>0.9884</td>
<td>0.0930</td>
<td>1.16×10⁻⁴</td>
<td>33.2415</td>
<td>27.8332</td>
<td>0.4150</td>
<td>107.9341</td>
<td>0.7553</td>
<td>9.5664</td>
<td>0.4100</td>
</tr>
</tbody>
</table>

Figure 2-43. Select Minimum Image Fusion Results
Principal component analysis (PCA) Image Fusion Results

Image A

Image B

Fused Image using Principal component analysis (PCA)

<table>
<thead>
<tr>
<th>Method</th>
<th>CORR</th>
<th>RMSE</th>
<th>MAE</th>
<th>MI</th>
<th>MSE</th>
<th>PSNR</th>
<th>QI</th>
<th>RMSE</th>
<th>SC</th>
<th>SNR</th>
<th>ssimval</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA</td>
<td>0.9936</td>
<td>8.09E-06</td>
<td>0.1246</td>
<td>14.9821</td>
<td>0.1892</td>
<td>1.35E+03</td>
<td>11.3026</td>
<td>32.5182</td>
<td>0.9204</td>
<td>36.6993</td>
<td>1.0087</td>
</tr>
</tbody>
</table>

*Figure 2-44. Principal component analysis (PCA) Image Fusion Results*
2.5.3. Time Results

Using MATLAB’s timer function, tic toc, the execution time performance of the MATLAB code of the different image fusion algorithms was measured and the results are shown in Figure 2.45 and Table 2-1.

![Figure 2-45. MATLAB Time Results](image)

**Table 2-1. MATLAB Time Results**

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
<th>DCT_LP</th>
<th>DSWT_1L</th>
<th>DSWT_2L</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
<th>DTCWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>0.46</td>
<td>0.46</td>
<td>0.02</td>
<td>4.72</td>
<td>3.86</td>
<td>7.55</td>
<td>2.79</td>
<td>2.76</td>
<td>0.14</td>
<td>1.58</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>4.55</td>
<td>4.53</td>
<td>0.22</td>
<td>46.97</td>
<td>38.55</td>
<td>76.65</td>
<td>27.84</td>
<td>27.81</td>
<td>0.22</td>
<td>15.88</td>
</tr>
</tbody>
</table>
2.6. Implementation of Fusion techniques using Python

After getting the results of the MATLAB, an implementation of four different Image fusion techniques was written using Python; these fusion techniques were chosen depending on either their image quality results or execution time or their complexity. These fusion techniques are

- Average Method
- Discrete Wavelet Transformation (DWT) using Daubechies filter
- Discrete Wavelet Transformation (DWT) using Haar filter
- Principal Component Analysis (PCA) Method

Python was chosen since it is widely used high-level, general-purpose, interpreted, dynamic programming language and for its ability to work on cross-platforms and its advances in parallel computing, compared to MATLAB, and finally for the availability of cloud services that already accepts Python in its approved programming languages.

Please refer to Appendix D, which provides the code and step-by-step explanation of the code being executed. The code accepts Fit/Fits files which are extension name of the astronomical images.

The code used different Python packages to implement the algorithms. They are

- AstroPy: a Python library for Astronomy
- NumPy: a scientific computing package in Python
- SciPy: an open source library of scientific tools in Python
• PyWavelets: a discrete wavelet transform package in Python

• TkInter: Python standard Graphical User Interface (GUI) package

The results from this Python code used a machine specification of

• Intel Core i5-M520 CPU @ 2.40 GHz (Dual Core)

• Installed memory (RAM) 8 GB

• Running windows 10 64-bit operating system

2.7. Results using Python

Using Python’s timer function the execution time performance of the Python code of the different image fusion algorithms was measured and the results are shown in Table 2-2 which will be used later as a reference when this work implements parallel and cloud computing.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Avg</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>0.062</td>
<td>3.113</td>
<td>2.842</td>
<td>0.645</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>0.556</td>
<td>31.570</td>
<td>28.321</td>
<td>6.267</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>1.067</td>
<td>63.602</td>
<td>56.673</td>
<td>12.089</td>
</tr>
</tbody>
</table>

*Table 2-2. Python Sequential Time Results*
2.8. Conclusion

Working with astronomical images gave different results than expected in regard to image quality and measure of effectiveness. The top algorithms, in ascending order, were

1. Principal Component Analysis (PCA)
2. Discrete wavelet transformation using Daubechies filter
3. Discrete wavelet transformation using Haar
4. Average Method

But in MATLAB, the fastest algorithms were

1. Average Method
2. Principal Component Analysis (PCA)
3. Select Maximum
4. Select Minimum

The slowest algorithms were

1. Discrete stationary wavelet transform
2. Discrete cosine transform based Laplacian pyramid
3. Discrete wavelet transformation using Daubechies filter
4. Discrete wavelet transformation using Haar

From these results, only four image fusion techniques were chosen to compare their sequential, parallel and cloud computing speed in Python. They were chosen for their great measure of effectiveness results and two of them for their great time results
and two were for their slow processing time results with the aim to be able to improve the results. These four fusions techniques were compared in sequential, parallel and cloud computing implemented in Python. These fusion techniques are

- Average Method
- Discrete Wavelet Transformation (DWT) using Daubechies filter
- Discrete Wavelet Transformation (DWT) using Haar filter
- Principal Component Analysis (PCA) Method
3. Parallel Computing

Chapter 3

In this Chapter, we will discuss parallel computing and its classification and main issues that it faces and discuss the implementation we did to our Image fusion code using Python and then we will discuss the results of our findings.

3.1. Introduction to parallel computing

Traditionally, computer programs were written to be executed in sequential order, meaning the algorithm instructions executed in a serial sequence so only one instruction is executed at a time but recently especially with the emerging technology of multi-core processors, the focus and interest on parallel computing increased which lead to the emerge of a vast majority of new concepts and definitions that usually scientist and engineers seem to get confused by, so let us introduce some fundamental concepts that are the core of parallel computing. So parallel computing is the use of multiple processing elements simultaneously to solve a problem and execute the code [18].
A major fundamental concept behind parallel computing is concurrency which is a property of a system in which multiple tasks that comprise the system remain active and make progress at the same time” [19]. While concurrency is an old concept, it was an integral concept in reaching parallel computing. Most of the mainstream operating systems have been heavily using concurrency concept usually by implementing time-sharing between different tasks and threads and by switching between them [20]. This way, if a process is stalled waiting for something, the other process can continue their progress and hence give the incorrect feel of multi-processing even in a single core system, but essentially that is known as multi-tasking which is having multiple processes and tasks running at the same time sharing CPU resource by time-slicing [21].

So to implement parallel computing, we need to introduce the concept of parallelism which is the exploitation of concurrency concept in a program with the goal of solving a problem in less time using multi-processing systems with multi-core hardware. So essentially, parallel computing is an application of concurrency. We cannot have parallelism without concurrency and to execute a parallel program in parallel; we must have hardware with multiple processing elements, so concurrent tasks execute in parallel.

So assuming we have the hardware that supports parallel computing, the next step will be to create software and programs that can utilize parallelism of the hardware, which then bring us to the parallelism of algorithms. So an algorithm is a sequence of commands and steps a program takes to solve a problem and traditionally the algorithm is executed in sequential order, so the commands run one at a time in a well-defined order.
So to achieve parallelism in algorithms we first need to achieve concurrency in the algorithm [22]. Concurrency in an algorithm implies that instead of a single serial sequence of steps, we will have multiple sequences of steps that execute together. These steps are interleaved in different ways depending on how the tasks are scheduled for execution.

So there are three main types of parallel algorithms based on the primary source of concurrency is the algorithm design [23]

- Task parallelism
- Data parallelism
- Implicit parallelism

So in a task parallel model, the algorithm will focus more on the execution of the process and threads and the communication between them in accessing shared memory and data. While in data parallel model, the algorithm will focus more on executing the program on a specific data set which is usually a regularly structured array which it’s data is independent so it can be disjoint and used in parallel processing and computing.

Finally, in the implicit parallel model, the flow of data and the algorithm goes through a network of processing stages that the run-time and hardware is responsible for and do the programmer or the compiler which should achieve automatic parallelization of algorithms.

The main issues that face parallel computing can be summed into: [24] [25]

- Race condition
A race condition occurs when two or more threads or processes can access shared data, and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data or as in our case the threads or processes are running in parallel. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, meaning that both threads/process are "racing" to access/change the data. Hence, when designing algorithms that include concurrency, you have to use constructs that force ordered access to memory in just those places where read/write sharing takes place which is essentially called synchronization constructs.

While a deadlock occurs when two or more thread or processes are each waiting for the other to finish to continue and this neither ever does as in the example shown below where process P2 will have to wait on process P1 to finish before it can resume and same for process P3 which is waiting on process P2. [26] [27]
Another performance problem can arise is due to load balancing, where tasks that are running in parallel are not equally distributed that will then cause a delay in the program waiting for one task to finish as shown in Figure 3.1 where task C, which is much larger than task B will cause the program to delay and wait until task C is finished before continuing to task D, hence, wasting computing time and not utilizing the time efficiency of parallel computing.

Finally, another problem would be the scalability of the program where parallelism of an algorithm does not necessarily speed the execution time, and that can be caused by different reason such as the algorithm overhead, which deals with the parallelizing, can
then make a simple algorithm gives a better result than its parallel counterpart [26] [27]. Also, the earlier discussed reason can lead to the same scalability problems.

3.2. Implementation of parallel computing using Python

Implementation of the four different Image fusion techniques using parallel processing was written using Python. These fusion techniques are

- Average Method
- Discrete Wavelet Transformation (DWT) using Daubechies filter
- Discrete Wavelet Transformation (DWT) using Haar filter
- Principal Component Analysis (PCA) Method

The code used different Python packages to implement the algorithms. They are

- AstroPy: a Python library for Astronomy
- NumPy: a scientific computing package in Python
- SciPy: an open source library of scientific tools in Python
- PyWavelets: a discrete wavelet transform package in Python
- TkInter: Python standard Graphical User Interface (GUI) package

And to implement the parallel processing in these algorithms, there were two different packages used in two separate methods for comparison. The first method uses Python’s default parallel processing library called “multiprocessing” package and the second method uses another package called “Joblib.” [29].
Please refer to Appendix E and Appendix F, which provide the code and step-by-step explanation of the code being executed. The code accepts Fit/Fits files which are extension name of the astronomical images.

The results from this Python code used a machine specification of

- Intel Core i5-M520 CPU @ 2.40 GHz (Dual Core)
- Installed memory (RAM) 8 GB
- Running windows 10 64-bit operating system

To measure the performance of the parallel program, the industry uses the term “Speed-up” which is the ratio of a program’s runtime in parallel to the runtime of the best available sequential implementation of that same algorithm.

\[ S = \frac{T_{\text{sequential}}}{T_{\text{parallel}}} \]  

When we have a perfectly parallel program, the speed-up (S) should scale linearly with the number of processing elements (P) meaning S should be equal P. This is called “perfectly linear speedup.”
3.3. Results

After running the two different methods of parallel computing in Python, the execution results was obtained as shown in Table 3-1 and Table 3-2.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Avg</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>1.446</td>
<td>5.526</td>
<td>5.326</td>
<td>2.024</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>12.318</td>
<td>29.891</td>
<td>28.782</td>
<td>20.938</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>24.292</td>
<td>56.846</td>
<td>55.802</td>
<td>51.626</td>
</tr>
</tbody>
</table>

3.4. Conclusion

So in parallel processing in Python, coding with the JobLib Python package gave a better result than the default Python’s multiprocessing library. A comparison of the performance of each image fusion algorithm is shown in Figure 3.2 to Figure 3.5 and Table 3-3 to Table 3-6.
Figure 3-2. Parallel processing comparison of Average Method

Table 3-3. Parallel processing comparison of Average Method

<table>
<thead>
<tr>
<th>Comparison of Avg</th>
<th>Iteration</th>
<th>Sequential</th>
<th>Parallel One</th>
<th>Parallel Two</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 iteration (2 Images fused)</td>
<td>0.062</td>
<td>1.446</td>
<td>2.419</td>
</tr>
<tr>
<td></td>
<td>10 iteration (20 Images fused)</td>
<td>0.556</td>
<td>12.318</td>
<td>6.363</td>
</tr>
<tr>
<td></td>
<td>20 iteration (40 Images fused)</td>
<td>1.067</td>
<td>24.292</td>
<td>13.738</td>
</tr>
</tbody>
</table>
Figure 3-3. Parallel processing comparison of Principal Component Analysis

Table 3-4. Parallel processing comparison of Principal Component Analysis

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Sequential</th>
<th>Parallel One</th>
<th>Parallel two</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>0.645</td>
<td>2.024</td>
<td>5.266</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>6.267</td>
<td>20.938</td>
<td>13.514</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>12.089</td>
<td>51.626</td>
<td>22.413</td>
</tr>
</tbody>
</table>
Figure 3-4. Parallel processing comparison of discrete wavelet transformation using Daub

Table 3-5. Parallel processing comparison of discrete wavelet transformation using Daub

<table>
<thead>
<tr>
<th>Comparison of DWT Daub</th>
<th>Sequential</th>
<th>Parallel One</th>
<th>Parallel two</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 images fused)</td>
<td>3.113</td>
<td>5.526</td>
<td>7.103</td>
</tr>
<tr>
<td>10 iteration (20 images fused)</td>
<td>31.570</td>
<td>29.891</td>
<td>21.320</td>
</tr>
<tr>
<td>20 iteration (40 images fused)</td>
<td>63.602</td>
<td>56.846</td>
<td>38.097</td>
</tr>
</tbody>
</table>
Figure 3-5. Parallel processing comparison of discrete wavelet transformation using Harr

Table 3-6. Parallel processing comparison of discrete wavelet transformation using Harr

<table>
<thead>
<tr>
<th>Comparison of DWT_Haar</th>
<th>Iteration</th>
<th>Sequential</th>
<th>Parallel One</th>
<th>Parallel two</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 iteration (2 Images fused)</td>
<td>2.842</td>
<td>5.326</td>
<td>6.908</td>
</tr>
<tr>
<td></td>
<td>10 iteration (20 Images fused)</td>
<td>28.321</td>
<td>28.782</td>
<td>20.183</td>
</tr>
<tr>
<td></td>
<td>20 iteration (40 Images fused)</td>
<td>56.673</td>
<td>55.802</td>
<td>36.469</td>
</tr>
</tbody>
</table>
Chapter 4

In this Chapter, we will discuss the emerging technology of cloud computing and its classification and main characteristic of its services. Then we will discuss the implementation we did to our Image fusion code of Python using Amazon Web Service (AWS)’s cloud service and then we will discuss the results of our findings.

4.1. Introduction to Cloud Computing

Cloud computing; also, sometimes known as on-demand computing is the collection of computing applications and services available from a decentralized network of servers which usually make use of the internet that is why it is considered an internet-based computing.

Cloud computing is typically classified using two methods: [30]

- Location of the cloud computing
- Type of services offered
The classifications based on the location of the cloud are [31]

- **Public Cloud**: In this model the cloud infrastructure is hosted by a cloud service providers, this enables a consumer to develop and deploy a service in the cloud with very little financial outlay compared to the capital expenditure requirements normally associated with other deployment options.

- **Private cloud**: In this model the cloud infrastructure is hosted and deployed and maintained for a specific organization. The operation can be within the premises of the organization or on a third party premises.

- **Community cloud**: In this model the cloud infrastructure is very similar to the private cloud but can be shared between organizations of the same community or with similar interest and requirements. The operation can be within the premises of the organization or on a third party premises.

- **Hybrid cloud**: In this model the cloud infrastructure consists of the combination of any number of the cloud types discussed above.
The classification based on the service models for cloud computing are: [32]

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

The Software as a service (SaaS) gives the users the ability to use an application or service that is hosted in the cloud as an example would be Salesforce.com, which provides its customers with Customer Relationship Management (CRM) software to use in the cloud. Another example would be Microsoft online version of office called Microsoft Office 365 or Google online documents, Google Docs, and a final example would be all the popular email systems used today such as Google’s Gmail or Microsoft’s
Hotmail. While Platform as a service (PaaS) gives the user a development platform to deploy it in a very specific task or service, and usually the user has no control over the operating systems and the network access, and there are usually constrictions of what type of application to be deployed, an example would be Google App Engine which gives a platform to build web and mobile apps. And finally Infrastructure as a service (IaaS) which gives the user the total control in choosing every aspect of his cloud computing service in terms of the operating systems, applications, storage, and network connectivity, an example would be Amazon EC2, Amazon S3, Microsoft Azure and Google Container Engine [33].

Figure 4.2. Cloud classifications based on the Service of the cloud [32]
The benefits and features of using cloud computing are: [30]

- **Cost saving:** Companies would reduce their expenses since they first will start with a lower initial investment and the use of the cloud would be faster due to the faster deployment and they will not have to worry about the maintenance expenses.

- **Scalability:** most cloud service providers offer their customers the ability to scale their cloud so they can easily grow or shrink their cloud usage based on their needs and requirements.

- **Flexibility:** similar to the scalability feature, the user has the ability to increase the scale of their cloud but in this feature based on-demand which then allow users to use extra resources at peak times.

- **Accessibility:** since the services are hosted in the cloud, the user will have the flexibility of accessing it from anywhere and any device since it is device independent and can be accessed from anywhere there is internet service.

- **Reliability:** most of the cloud service providers uses multiple redundancies and disaster recovery options so the users won’t have to worry about losing their services.

- **Security:** the users of cloud computing will not have to worry about implementing the latest in technology regarding security because most of the current cloud services providers include very secure operations in their cloud services.
4.2. Implementation of cloud computing

Implementation of the four different Image fusion techniques using cloud processing was written using Python. These fusion techniques are

- Average Method
- Discrete Wavelet Transformation (DWT) using Daubechies filter
- Discrete Wavelet Transformation (DWT) using Haar filter
- Principal Component Analysis (PCA) Method

The code used different Python packages to implement the algorithms. They are

- AstroPy: a Python library for Astronomy
- NumPy: a scientific computing package in Python
- SciPy: an open source library of scientific tools in Python
- PyWavelets: a discrete wavelet transform package in Python
- TkInter: Python standard Graphical User Interface (GUI) package

Please refer to Appendix D, which provides the code and step-by-step explanation of the code being executed. The code accepts Fit/Fits files which are extension name of the astronomical images.
After comparing different cloud services option, as shown in Figure 4.3 and Figure 4.4, the cloud computing service used was Amazon Web Services (AWS) [33] which offers different cloud computing services and products depending on the user’s need

- Compute
- Networking
- Storage and Content Delivery
For our implementation, we used Amazon Elastic Compute Cloud (EC2) which allows us to rent virtual machines designed specifically for cloud computing. Amazon calls each virtual machine an “instance” and for our implementation, we used two different instances for comparison. The first instance was running Ubuntu server 14.04 LTS and the other instance was running Microsoft Windows server 2012 R2 Base.

Figure 4-5. Amazon Elastic Compute Cloud (EC2) instances used
Amazon provides the user with the ability to customize the instance using five different parameters

- Memory: users can choose from a range of 1 GB to 244 GB
- The number of virtual CPUs for the instance: ranges from 1 to 40
- Storage: users can choose from a range of 1 GB to 24 x 2048 GB
- Network performance: which indicates the level of the rate of data transfers which goes up to non-blocked 10 Gigabit Ethernet network
- EBS-optimized: This provides additional, dedicated throughput for Amazon EBS I/O which provides improved performance for the computing power of the instance.

Our Ubuntu cloud server specifications are:

- Intel Xeon Family @ 2.5 GHz (Single Core)
- Installed memory (RAM) 1 GB
- Running Ubuntu Server 14.04 LTS (HVM)
- Volume type is Solid-state drive (SSD)

Our Windows cloud server specifications are:

- Intel Xeon Family @ 2.5 GHz (Single Core)
- Installed memory (RAM) 1 GB
- Running Microsoft Windows Server 2012 R2 Base
4.3. Results

After running the two different methods of cloud computing in Python, the execution results were obtained as shown in Table 4-1 and Table 4-2.

Table 4-1. Microsoft Windows Server Cloud Time Results

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Avg</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>0.078</td>
<td>2.266</td>
<td>2.250</td>
<td>0.672</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>0.672</td>
<td>22.499</td>
<td>22.015</td>
<td>6.734</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>1.312</td>
<td>45.233</td>
<td>43.999</td>
<td>13.250</td>
</tr>
</tbody>
</table>

Table 4-2. Ubuntu server Cloud Time Results

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Avg</th>
<th>DWT_Daub</th>
<th>DWT_Haar</th>
<th>PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>0.057</td>
<td>0.717</td>
<td>0.596</td>
<td>0.288</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>0.575</td>
<td>7.842</td>
<td>6.548</td>
<td>2.761</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>1.148</td>
<td>15.643</td>
<td>13.215</td>
<td>5.536</td>
</tr>
</tbody>
</table>

4.4. Conclusion

So in the cloud processing the Linux-based Ubuntu gave better results than the windows server based system. A comparison of the performance of each image fusion algorithm is shown in Figure 4.6 to Figure 4.9 and Table 3-3 to Table 3-6.
Figure 4-6. Cloud processing comparison of Average Method

Table 4-3. Cloud processing comparison of Average Method

<table>
<thead>
<tr>
<th>Comparison of Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
</tr>
<tr>
<td>1 iteration (2 Images fused)</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
</tr>
</tbody>
</table>
Figure 4-7. Cloud processing comparison of Principal Component Analysis

Table 4-4. Cloud processing comparison of Principal Component Analysis

<table>
<thead>
<tr>
<th>Comparison of PCA</th>
<th>Iteration</th>
<th>Reference</th>
<th>windows</th>
<th>Ubuntu</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 iteration (2 Images fused)</td>
<td>0.645</td>
<td>0.672</td>
<td>0.288</td>
</tr>
<tr>
<td></td>
<td>10 iteration (20 Images fused)</td>
<td>6.267</td>
<td>6.734</td>
<td>2.761</td>
</tr>
<tr>
<td></td>
<td>20 iteration (40 Images fused)</td>
<td>12.089</td>
<td>13.250</td>
<td>5.536</td>
</tr>
</tbody>
</table>
Figure 4-8. Cloud processing comparison of discrete wavelet transformation using Daub

Table 4-5. Cloud processing comparison of discrete wavelet transformation using Daub

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
<th>windows</th>
<th>Ubuntu</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>3.113</td>
<td>2.266</td>
<td>0.717</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>31.570</td>
<td>22.499</td>
<td>7.842</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>63.602</td>
<td>45.233</td>
<td>15.643</td>
</tr>
</tbody>
</table>
Figure 4-9. Cloud processing comparison of discrete wavelet transformation using Harr

Table 4-6. Cloud processing comparison of discrete wavelet transformation using Harr

<table>
<thead>
<tr>
<th>Comparison of DWT_Haar</th>
<th>Iteration</th>
<th>Reference</th>
<th>windows</th>
<th>Ubuntu</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 iteration (2 Images fused)</td>
<td>2.842</td>
<td>2.250</td>
<td>0.596</td>
</tr>
<tr>
<td></td>
<td>10 iteration (20 Images fused)</td>
<td>28.321</td>
<td>22.015</td>
<td>6.548</td>
</tr>
<tr>
<td></td>
<td>20 iteration (40 Images fused)</td>
<td>56.673</td>
<td>43.999</td>
<td>13.215</td>
</tr>
</tbody>
</table>
5. Further Image Fusion implementation

Chapter 5

Images fusion is having higher impact and uses in the astronomical science and technology. In this chapter, an introduction to some of the uses of image fusion in Space Situational Awareness (SSA) programs, also, a method for image characterization will be introduced and finally, the results of implementing image fusion on big data would be viewed.

5.1. Space Situational Awareness (SSA)

The definition of Space Situational Awareness (SSA) refers to the comprehensive ability to identify and view space objects, both natural and manmade objects, and possess the ability to track, understand and predict their future location with the objective of avoiding collisions [34]. Through achieving that objective, we will end up with a system that holds a routine, operational service of detection, correlation, characterization and orbit determination of space objects.

Currently, there are two main Space Situational Awareness (SSA) programs

- European Space Agency's Space Situational Awareness (SSA) Programme
• United States of America’s Geosynchronous Space Situational Awareness Program (GSSAP)

The European Space Agency’s Space Situational Awareness (SSA) Programme is classified into three main areas [34]

• Space Weather (SWE): monitoring conditions at the Sun and in the solar wind, and in Earth’s magnetosphere, ionosphere, and thermosphere, that can affect spaceborne and ground-based infrastructure or endanger human life or health

• Near-Earth Objects (NEO): detecting natural objects that can potentially impact Earth and cause damage

• Space Surveillance and Tracking (SST): watching for active and inactive satellites, discarded launch stages and fragmentation debris orbiting Earth

While the United States of America’s Geosynchronous Space Situational Awareness Program (GSSAP) names its components into intelligence, surveillance, reconnaissance, environmental monitoring, and command and control.

Usually, the monitoring and tracking of these space, objects are completed using ground and space-based radar and imagery systems such as astronomical images. So with the use of data and image fusion in Space Situational Awareness (SSA) programs will lead to ease and speeding up of the monitoring and tracking process. Studies already show that observers can localize a target in a scene with a significantly higher accuracy and with a greater amount of confidence when they perform image fusion compared to individual image [35].
Systems already started to emerge that implements data and image fusion for Space Situational Awareness (SSA) implementation such as Ibex, which is MIT program funded by Defense Advanced Research Projects Agency (DARPA) and Air Force program. But Ibex focus more on data fusion rather than image fusion so sensor data will go first through feature-based cataloging, as shown in Figure 5.1, and the resulting data will be fused rather than using first image fusion to sensor’s images and then running the fused images through the feature-based cataloging algorithms and then fusing the data from the results which according to research in that field should give better results [36].

![Figure 5-1. Ibex SSA data fusion implementation][1]

[1]: #/content/images/Figure_5-1.png
5.2. Image Characterization

Image characterization is the process of getting information from an image or more that describes information of a character that an image has or share with other images. So in our work, the problem we tried to tackle that if given a group of astronomical images can we discover if they came from the same source or not.

Astronomical images are taken by space observatories spread across the globe using telescopes equipped with Charge-Coupled Device (CCD) camera [37], which is similar to any other camera lens, emits a noise signal and each noise signal is unique to the camera being used which can be an identifiable factor which is according to Ali et al. is one way to identify satellite images which are by comparing their the noise on each image [38].

The paper describes the process as first normalizing the image and then apply contrast stretching on all images and then remove the noise in image using a two-dimensional adaptive noise-removal filtering such as Wiener Filtering and after getting the noise image from each image we convert to a binary image, to a set of zeros and one, and compare them together to give us a probability if these images come from the same source [38]. In an effort to improve that process instead of just binarization of the noise images, we create a multilevel image threshold instead of just two, zero and one. So, in essence, we won’t be just comparing for the existence of noise but also, we account for the intensity value of that noise which would be similar if it came from the same camera.
The Wiener filter estimates the local mean and variance around each pixel described in Equation 5-1 [39]

\[
\mu = \frac{1}{NM} \sum_{i,j \in \eta} A(i,j)
\]

\[
\sigma^2 = \frac{1}{NM} \sum_{i,j \in \eta} A(i,j) - \mu^2
\]

Where \( \mu \) is local mean

\( \sigma \) is variance

\( \eta \) is the N-by-M local neighborhood of each pixel in the image A

then creates a pixel-wise Wiener filter using estimates from Equation 5-2 [39]

\[
B(i,j) = \mu + \left( \frac{\sigma^2 - \nu^2}{\sigma^2} \ast (A(i,j) - \mu) \right)
\]

Where \( \nu^2 \) is the noise variance which is the average of all the local estimated variances.

Algorithm steps:

1. Normalize both Images
2. Apply Contrast stretching on all images
3. Create the noise image using Wiener two-dimensional adaptive noise-removal filter
4. Create a multilevel noise image thresholds
5. Compare both of the noise images and calculate the probability
5.3. Big-Data Image Fusion

Image fusion algorithm based on PCA was implemented on 30,000 astronomical images enclosed in eighteen folders producing a final eighteen fused images, each one fused image representing the folder’s astronomical images. Figure 5.2 through Figure 5.5 shows these results.

Figure 5-2. Big Data Image Fusion Results 1
Figure 5-3. Big Data Image Fusion Results 2
Figure 5-4. Big Data Image Fusion Results.
These results were obtained using Python on a machine specification of

- Intel Core i5-M520 CPU @ 2.40 GHz (Dual Core)
- Installed memory (RAM) 8 GB
- Running windows 10 64-bit operating system

Some of the resulting fused imagery shows a pattern that can be recognized in object tracking and other resulting fused imagery shows the possibility of different astronomical scenes. It is also not known what was the intention of the person taking these images and therefore cannot provide much detail.
6. Conclusion and Future work

Chapter 6

After applying different image fusion algorithms to astronomical images, the image quality and measure of effectiveness results were unexpectedly different from regular digital images. The top image fusion algorithms that provided the best image quality and measure of effectiveness results, in the order of best first and worst last, were

1. Principal Component Analysis
2. Discrete Wavelet Transformation using Daubechies filter
3. Discrete Wavelet Transformation using Haar filter
4. Average Method

But in MATLAB, the fastest algorithms were

1. Average Method
2. Principal Component Analysis
3. Select Maximum
4. Select Minimum
The slowest algorithms in MATLAB were

1. Discrete Stationary Wavelet Transform
2. Discrete Cosine Transform based Laplacian Pyramid
3. Discrete Wavelet Transformation using Daubechies filter
4. Discrete Wavelet Transformation using Haar filter

From these results, only four image fusion techniques were chosen to compare their sequential, parallel and cloud computing speed in Python. They were chosen for their great measure of effectiveness results. These four fusions techniques were

- Average Method
- Discrete Wavelet Transformation (DWT) using Daubechies filter
- Discrete Wavelet Transformation (DWT) using Haar filter
- Principal Component Analysis (PCA) Method

So in parallel processing in Python, coding with the JobLib Python package gave a better result than the default Python’s multiprocessing library. While in the cloud processing the Linux-based Ubuntu gave better results than the windows server based system. To compare the best results from each processing and compare it to sequential we have chosen the best in each category to represent it, meaning JobLib for Parallel and Ubuntu for the cloud.
The Four different techniques of average, PCA, and two DWTs were compared based on the “speed-up”.

106
### Table 6-5. Speed-up comparison of Average Method

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Parallel</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>0.026</td>
<td>1.095</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>0.087</td>
<td>0.966</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>0.078</td>
<td>0.929</td>
</tr>
</tbody>
</table>

### Table 6-6. Speed-up comparison of Principal Component Analysis

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Parallel</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>0.122</td>
<td>2.238</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>0.464</td>
<td>2.270</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>0.539</td>
<td>2.184</td>
</tr>
</tbody>
</table>

### Table 6-7. Speed-up comparison of discrete wavelet transformation using Daub

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Parallel</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>0.438</td>
<td>4.341</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>1.481</td>
<td>4.026</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>1.669</td>
<td>4.066</td>
</tr>
</tbody>
</table>

### Table 6-8. Speed-up comparison of discrete wavelet transformation using Harr

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Parallel</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 iteration (2 Images fused)</td>
<td>0.411</td>
<td>4.770</td>
</tr>
<tr>
<td>10 iteration (20 Images fused)</td>
<td>1.403</td>
<td>4.325</td>
</tr>
<tr>
<td>20 iteration (40 Images fused)</td>
<td>1.554</td>
<td>4.289</td>
</tr>
</tbody>
</table>

To conclude, cloud computing definitely gives better results than parallel and sequential even though the machine specification of the cloud computer was significantly lower than both the sequential and parallel machine but having a dedicated machine that
is optimized for computation makes a big difference. Moving forward, would be combining parallel computing with cloud computing to achieve even better results but we need to keep in mind that simpler algorithms might not give us better results when parallelized as the average method that gave us worst results but with more advanced algorithms, such as the Discrete Wavelet Transforms, gave us considerable better computing time results.
References


Appendix A

MATLAB Script for Image Fusion and Image Measures of Effectiveness

% Mohamed AbouRayan
% Jan 21, 2016
% Image Fusion using different methods
% (using two Images in FIT/FITS/FTS format)
% 2nd Demo

% Stores desired files location into a variable to be easily reused
promt1 = ('please specify the location and name of the first Image ?');
fileLocation1 = input(promt1, 's');
promt2 = ('please specify the location and name of the second Image ?');
fileLocation2 = input(promt2, 's');
% For manual entry use this:
% fileLocation1 = 'E:\Image Fusion\Matllab code & data files\FTS\Autosave Image -001B.fit';
% fileLocation2 = 'E:\Image Fusion\Matllab code & data files\FTS\Autosave Image -001R.fit';

% Stores desired output file location after Image Fusion
promt3 = ('please specify the desired location and name of the resulting Fused Image ?');
fusionFileLocation = input(promt3, 's');
% For manual entry use this:
% fusionFileLocation = 'E:\Image Fusion\Matllab code & data files\FTS\FusionResult.fits';
% Read data from FITS files and store it to a variable FITS Image
fitsImage1 = fitsread(fileLocation1);
fitsImage2 = fitsread(fileLocation2);

% Image Fusion using Fusion_func script

disp('Please choose from the following algorithms to perform Image Fusion');
disp('1-Maximum Method');
disp('2-Minimum Method');
disp('3-Average Method ');
disp('4-Discrete cosine transform (DCT) & LP (Laplacian pyramid) ');
disp('5-Discrete stationary wavelet transform (one level) ');
disp('6-Discrete stationary wavelet transform (two levels) ');
disp('7-Discrete wavelet transformation (DWT) -- Daubechies ');
disp('8-Discrete wavelet transformation (DWT) -- Haar ');
disp('9-Principal Component Analysis (PCA) Method ');
disp('10-Dual-tree complex wavelet transform (DTCWT)' );
promt4=('Please choose which Image Fusion Algorithm to perform: ');
fusion = input(promt4,'s');

switch fusion
    case '1'
        disp('You Chose the 1-Maximum Method')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'max');
    case '2'
        disp('You Chose the 2-Minimum Method')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'min');
    case '3'
        disp('You Chose the 3-Average Method')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'avg');
    case '4'
        disp('You Chose the 4-Discrete cosine transform (DCT) & LP (Laplacian pyramid)')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'lp');
    case '5'
        disp('You Chose the 5-Discrete stationary wavelet transform (one level)')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'swt');
    case '6'
        disp('You Chose the 6-Discrete stationary wavelet transform (two levels)')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'swt2');
    case '7'
        disp('You Chose the 7-Discrete wavelet transformation (DWT) -- Daubechies')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'dwt');
    case '8'
        disp('You Chose the 8-Discrete wavelet transformation (DWT) -- Haar')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'dwt');
    case '9'
        disp('You Chose the 9-Principal Component Analysis (PCA) Method')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'pca');
    case '10'
        disp('You Chose the 10-Dual-tree complex wavelet transform (DTCWT)')
        fusedImage=Fusion_func(fitsImage1,fitsImage2,'dtcwt');
end
disp('You Chose the 4-Discrete cosine transform (DCT) & LP (Laplacian pyramid)')

fusedImage=Fusion_func(fitsImage1,fitsImage2,'DCT_LP');
case '5'
    disp('You Chose the 5-Discrete stationary wavelet transform (one level)')
fusedImage=Fusion_func(fitsImage1,fitsImage2,'DSWT_1L');
case '6'
    disp('You Chose the 6-Discrete stationary wavelet transform (two levels)')
fusedImage=Fusion_func(fitsImage1,fitsImage2,'DSWT_2L');
case '7'
    disp('You Chose the 7-Discrete wavelet transformation (DWT) -- Daubechies')
fusedImage=Fusion_func(fitsImage1,fitsImage2,'DWT_Daub');
case '8'
    disp('You Chose the 8-Discrete wavelet transformation (DWT) -- Haar')
fusedImage=Fusion_func(fitsImage1,fitsImage2,'DWT_Haar');
case '9'
    disp('You Chose the 9-Principal Component Analysis (PCA) Method')
fusedImage=Fusion_func(fitsImage1,fitsImage2,'PCA');
case '10'
    disp('You Chose the 10-Dual-tree complex wavelet transform (DTCWT)')
fusedImage=Fusion_func(fitsImage1,fitsImage2,'DTCWT');
otherwise
    disp('You Chose the 3-Average Method')
fusedImage=Fusion_func(fitsImage1,fitsImage2,'avg');
end

% Displaying FIT images
subplot(1,3,1);
imagesc(fitsImage1);
title('Image 1');
subplot(1,3,2);
imagesc(fitsImage2);
title('Image 2');
colormap(gray)

% Displaying the resulting FITS fused image
subplot(1,3,3);
imagesc(fusedImage);
title('Image Fusion');

% Create a FITS file with the resulting fusion
fitswrite(fusedImage,fusionFileLocation);

% computing Measures of Effectiveness (MOE)
[RMSE,PFE,MAE,den,CORR,SNR,PSNR,SC,LMS,E,VIF,PSR_HVAS,PSR_ 
_HAV,MSE,ssimval,MI,QI,SVDQM]=imageEval(fitsImage1,fusedImage);
Appendix B

MATLAB function that applies Image Fusion

% Mohamed AbouRayan
% Jan 02 ,2016
% Image Fusion using 10 differnt Algorithms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% How to use:
% By ethier calling Fusion_func(A,B) or Fusion_func(A,B,C)
% or Fusion_func(A,B,C,K)
% A and B are images needed to be fused. They need to be
% same size.
% C is the Fusion method unless specified Average method is
% going to be used
% K is the K value needed for some Algorithm
% (DCT_LP,DWT_Daub,DTCWT and DWT_Haar)
% unless specified a K value of 2 is going to be used
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Choices of Algorithm (C): max, min, avg, DCT_LP, DSWT_1L,
% DSWT_2L, DWT_Daub , DWT_Haar, DTCWT or PCA
% max: Image Fusion using Maximum Method
% min: Image Fusion using Minimum Method
% avg: Image Fusion using Average Method
% DCT_LP: Image Fusion using Discrete cosine transform
% (DCT) & LP (Laplacian pyramid) (Can Take a K Value)
% DSWT_1L: Image Fusion using Discrete stationary wavelet
% transform (one level)
% DSWT_2L: Image Fusion using Discrete stationary wavelet
% transform (two levels)
% DWT_Daub: Image Fusion using Discrete wavelet
% transformation (DWT) -- Daubechies (Can Take a K Value)
function [fusedImage] = Fusion_func(varargin)
% setting up the inputs for the processing
fitsImage1=varargin{1};
fitsImage2=varargin{2};
% accepting which algorithm of Image fusion the user wants to run
% Also, sets the default to Average algorithm
if nargin>2

% DWT_Haar: Image Fusion using Discrete wavelet transformation (DWT) -- Haar (Can Take a K Value)
% PCA: Image Fusion using Principal Component Analysis (PCA) Method
% DTCWT: Image Fusion using Dual-tree complex wavelet transform (DTCWT) (Can Take a K Value)

% Example:
% Fusion_func(A,B,'PCA'):This Will fuse Image A and B using PCA Method
% Fusion_func(A,B,'DWT_Haar',4):This will fuse Image A and B using DWT - Haar Method

% Inputs : accepted inputs are either 2 , 3 or 4 inputs.
% Fusion_func(A,B)
% Fusion_func(A,B,C)
% Fusion_func(A,B,C,K)
% A and B are images needed to be fused. They need to be same size.
% C is the Fusion method unless specified Average method is going to be used
% K is the K value needed for some Algorithm

% Outputs: The output is the fused image
% Acknowledgment: some elements of Dr. VPS Naidu's Matlab code was used to develop some parts of this MATLAB script.
% https://scholar.google.com/citations?user=pdAZ1wsAAAAJ&hl=en

% Default Values: unless specified [Fusion algorithm is average] and [K value is 2]
fusion=varargin{3};
else
    fusion='avg';
end
if nargin>3
    k=varargin{4};
else
    % default value for K
    k=2;
end
switch fusion
    case 'avg'
        fusedImage=Fusion_avg(fitsImage1,fitsImage2);
    case 'DCT_LP'
        fusedImage=Fusion_DCT_LP(fitsImage1,fitsImage2,k);
    case 'DSWT_1L'
        fusedImage=Fusion_DSWT_1L(fitsImage1,fitsImage2);
    case 'DSWT_2L'
        fusedImage=Fusion_DSWT_2L(fitsImage1,fitsImage2);
    case 'DWT_Daub'
        fusedImage=Fusion_DWT_Daub(fitsImage1,fitsImage2,k);
    case 'DWT_Haar'
        fusedImage=Fusion_DWT_Haar(fitsImage1,fitsImage2,k);
    case 'max'
        fusedImage=Fusion_max(fitsImage1,fitsImage2);
    case 'min'
        fusedImage=Fusion_min(fitsImage1,fitsImage2);
    case 'PCA'
        fusedImage=Fusion_PCA(fitsImage1,fitsImage2);
    case 'DTCWT'
        fusedImage=Fusion_DTCWT(fitsImage1,fitsImage2,k);
    otherwise
        fusedImage=Fusion_avg(fitsImage1,fitsImage2);
end

% Image Fusion using Averaging (using two Images in FIT/FITS/FTS format)
function [fusedImage] = Fusion_avg(fitsImage1,fitsImage2)
% Image Fusion using averaging algorithm
fusedImage = (fitsImage1 + fitsImage2) / 2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Image Fusion using DCT (Discrete cosine transform) & LP
% (Laplacian pyramid)
% (using two Images in FIT/FITS/FTS format)
function [imf] = Fusion_DCT_LP(IM1, IM2, k)
for i = 1:k
    IM = reduce2d(IM1);
    Id1 = IM1 - expand2d(IM);
    IM1 = IM;
    IM = reduce2d(IM2);
    Id2 = IM2 - expand2d(IM);
    IM2 = IM;
    dl = abs(Id1) - abs(Id2) >= 0;
    Idf{i} = dl .* Id1 + (~dl) .* Id2;
end
imf = 0.5 * (IM1 + IM2);
for i = k:-1:1
    imf = Idf{i} + expand2d(imf);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Image Fusion using discrete stationary wavelet transform
% (one level)
% (using two Images in FIT/FITS/FTS format)
function [fusedImage] = Fusion_DSWT_1L(fitsImage1, fitsImage2)
% image decomposition using discrete stationary wavelet transform
[A1L1, H1L1, V1L1, D1L1] = swt2(fitsImage1, 1, 'sym2');
[A2L1, H2L1, V2L1, D2L1] = swt2(fitsImage2, 1, 'sym2');
% fusion start
AfL1 = 0.5*(A1L1+A2L1);
D = (abs(H1L1)-abs(H2L1))>=0;
HfL1 = D.*H1L1 + (~D).*H2L1;
D = (abs(V1L1)-abs(V2L1))>=0;
VfL1 = D.*V1L1 + (~D).*V2L1;
D = (abs(D1L1)-abs(D2L1))>=0;
DfL1 = D.*D1L1 + (~D).*D2L1;
% fused image
fusedImage = iswt2(AfL1,HfL1,VfL1,DfL1,'sym2');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Image Fusion using discrete stationary wavelet transform
%(two levels)
%%%% (using two Images in FIT/FITS/FTS format)
function [fusedImage] =
Fusion_DSWT_2L(fitsImage1,fitsImage2)
% image decomposition using discrete stationary wavelet transform
[A1L1,H1L1,V1L1,D1L1] = swt2(fitsImage1,1,'sym2');
[A2L1,H2L1,V2L1,D2L1] = swt2(fitsImage2,1,'sym2');
[A1L2,H1L2,V1L2,D1L2] = swt2(A1L1,1,'sym2');
[A2L2,H2L2,V2L2,D2L2] = swt2(A2L1,1,'sym2');
% fusion at level2
AfL2 = 0.5*(A1L2+A2L2);
D = (abs(H1L2)-abs(H2L2))>=0;
HfL2 = D.*H1L2 + (~D).*H2L2;
D = (abs(V1L2)-abs(V2L2))>=0;
VfL2 = D.*V1L2 + (~D).*V2L2;
D = (abs(D1L2)-abs(D2L2))>=0;
DfL2 = D.*D1L2 + (~D).*D2L2;
% fusion at level1
D = (abs(H1L1)-abs(H2L1))>=0;
HfL1 = D.*H1L1 + (~D).*H2L1;
D = (abs(V1L1)-abs(V2L1))>=0;
VfL1 = D.*V1L1 + (~D).*V2L1;
D = (abs(D1L1)-abs(D2L1))>=0;
DfL1 = D.*D1L1 + (~D).*D2L1;
% fused image
AfL1 = iswt2(AfL2,HfL2,VfL2,DfL2,'sym2');
fusedImage = iswt2(AfL1,HfL1,VfL1,DfL1,'sym2');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Image Fusion using Discrete wavelet transformation (DWT) -- Daubechies
function [fusedImage] = Fusion_DWT_Daub(fitsImage1, fitsImage2, k)
% Image fusion
fusedImage = wfusimg(fitsImage1, fitsImage2, 'db2', k, 'mean', 'mean');

% Image Fusion using Discrete wavelet transformation (DWT) -- Haar
function [fusedImage] = Fusion_DWT_Haar(fitsImage1, fitsImage2, k)
% Image fusion
fusedImage = wfusimg(fitsImage1, fitsImage2, 'haar', k, 'mean', 'mean');

% Image Fusion using Maximum Method (using two Images in FIT/FITS/FTS format)
function [fusedImage] = Fusion_max(fitsImage1, fitsImage2)
% Image Fusion using Maximum algorithm
% Determining the image dimensions
[d1, d2] = size(fitsImage1);
% prelocating space for Fused image for speed
fusedImage = zeros(d1, d2);
% Gathering the Maximum value of each pixel of both images and storing it
% to fused image
for i = 1:d1;
    for j = 1:d2;
        if fitsImage1(i, j) > fitsImage2(i, j)
            fusedImage(i, j) = fitsImage1(i, j);
        else
            fusedImage(i, j) = fitsImage2(i, j);
        end
    end
end
% Image Fusion using Minimum Method (using two Images in FIT/FITS/FTS format)
function [fusedImage] = Fusion_min(fitsImage1, fitsImage2)
% Image Fusion using Minimum algorithm
% Determining the image dimensions
[d1, d2] = size(fitsImage1);
% prelocating space for Fused image for speed
fusedImage = zeros(d1, d2);
% Gathering the Minimum value of each pixel of both images
% and storing it to fused image
for i = 1:d1;
    for j = 1:d2;
        if fitsImage1(i, j) > fitsImage2(i, j)
            fusedImage(i, j) = fitsImage1(i, j);
        else
            fusedImage(i, j) = fitsImage2(i, j);
        end
    end
end

% Image Fusion using Averaging (using two Images in FIT/FITS/FTS format)
function [fusedImage] = Fusion_PCA(fitsImage1, fitsImage2)
% Image Fusion using PCA algorithm
% creating the covariance matrix
C = cov([fitsImage1(:) fitsImage2(:)]);
% Obtain the Eigenvectors and Eigenvalues from the covariance matrix
[V, D] = eig(C);
if D(1,1) >= D(2,2)
    pca = V(:,1)./sum(V(:,1));
else
    pca = V(:,2)./sum(V(:,2));
end
% create Fused Image after figureing out PCA for each Image
fusedImage = pca(1)*fitsImage1 + pca(2)*fitsImage2;

% Image Fusion using Dual-tree complex wavelet transform (DTCWT) Image Fusion
function [fusedImage] = Fusion_DTCWT(fitsImage1, fitsImage2, K)
wt1 = dddtree2('dwt', fitsImage1, K, 'sym2');
wt2 = dddtree2('dwt', fitsImage2, K, 'sym2');
ft2 = wt2;
ft2.cfs{1} = 0.5 * (wt1.cfs{1} + wt2.cfs{1});
ft2.cfs{2} = 0.5 * (wt1.cfs{2} + wt2.cfs{2});
ft2.cfs{3} = 0.5 * (wt1.cfs{3} + wt2.cfs{3});
fusedImage = idddtree2(ft2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MATLAB function that applies Image Measures of Effectiveness

% Mohamed AbouRayan
% Jan 25, 2016
% Measures of Effectiveness (MOE) for Image Fusion using 17 different measurements
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% How to use:
% By just calling imageEval(imt, imf)
% imt: Initial Image
% imf: Final Image (For example, Fused Image)
% and then save the 17 outputs to their targeted variables
% 
% [RMSE, PFE, MAE, dent, CORR, SNR, PSNR, SC, LMSE, VIF, PSNR_HVSM, PSNR_HVS, MSE, ssimval, MI, QI, SVDQM]
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Example:
% 
% [RMSE, PFE, MAE, dent, CORR, SNR, PSNR, SC, LMSE, VIF, PSNR_HVSM, PSNR_HVS, MSE, ssimval, MI, QI, SVDQM]=imageEval(imt, imf);
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inputs: This algorithm takes 2 input images (imt, imf)
% imt: Initial Image
% imf: Final Image (For example, Fused Image)
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Outputs: There are 17 outputs and they will be saved in this order
% 1-RMSE: Root Mean Square Error
% 2-PFE: Percentage Fit Error
% 3-MAE: Mean Absolute Error
% 4-dent: Difference Entropy
% 5-CORR: Normalized cross correlation
% 6-SNR: Signal to noise Ratio
% 7-PSNR: Peak signal to noise Ratio
% 8-SC: Structural content
% 9-LMSE: Laplacian Mean Squared Error
% 10-VIF: Visual information fidelity measure between the two images
% 11-PSNR_HVSM: Human Visual system (HVS) based PSNR (using Contrast Sensitivity Function (CSF) and coefficient contrast masking)
% 12-PSNR_HVS: Human Visual system (HVS) based PSNR (using Contrast Sensitivity Function (CSF))
% 13-MSE: Mean-Squared Error
% 14-ssimval: Structural Similarity Index (SSIM) for measuring image quality
% 15-MI: Mutual Information
% 16-QI: Universal Image Quality Index
% 17-SVDQM: SVD-based Image Quality Measure

%%% Acknowledgment: some elements of Dr. VPS Naidu's Matlab code was used to develop some parts of this MATLAB script.

```matlab
function [RMSE, PFE, MAE, dent, CORR, SNR, PSNR, SC, LMSE, VIF, PSNR_HVSM, PSNR_HVS, MSE, ssimval, MI, QI, SVDQM] = imageEval(imt, imf)

% Mean-squared error
MSE = immse(imt, imf);

% Structural Similarity Index (SSIM) for measuring image quality
ssimval = ssim(imt, imf);

% Root mean square error (RMSE)
[m, n] = size(imt);
RMSE = sqrt(sum((imt(:)-imf(:)).^2)/(m*n));

% Percentage fit error (PFE)
PFE = 100*norm(imt(:)-imf(:))/norm(imt(:));
```


% mean absolute error (MAE)
MAE = sum(sqrt((imt(:)-imf(:)).^2))/(m*n);

% Difference entropy H
dent = abs(entropy(imt)-entropy(imf));

% Correlation (CORR)
Rtf = sum(sum(imt.*imf));
Rt = sum(sum(imt.*imt));
Rf = sum(sum(imf.*imf));
CORR = 2*Rtf/(Rt+Rf);

% signal to noise ration (SNR)
st = mean(mean((double(imt)).^2));
ntf = mean(mean((double(imt-imf)).^2));
SNR = 10*log10(st/ntf);

% Peak signal to noise Ratio (PSNR)
L = 256;
PSNR = 10*log10(L^2/RMSE);

% mutual information(MI)
MI = minf2(imt,imf);

% universal image quality index (QI)
QI = uiqi(imt,imf,8);

%SVD-based Image Quality Measure
SVDQM = SVDQualityMeasure(imt,imf,8);

%structural content
SC = structural_content(imt,imf);

% LMSE(Laplacian Mean Squared Error
LMSE = LMS_error(imt,imf);

%VIF the visual information fidelity measure between the two images
VIF = vifp_mscale(imt,imf);

% PSNR-HVS-M and PSNR-HVS image quality measures
[PSNR_HVSM, PSNR_HVS] = psnrhvsim(imt,imf,8);
function [SC] = structural_content(A,B)
% SC (Structural Content)
Bs = sum(sum(B.^2));
Pk = sum(sum(A.^2));
if (Bs == 0)
    SC = Inf;
else
    SC = Pk/sum(sum(B.^2)); % SC
end

function [LMSE] = LMS_error(A,B)
% LMSE (Laplacian Mean Squared Error)
OP = 4*del2(A);
LMSE = sum(sum((OP-4*del2(B)).^2))/sum(sum(OP.^2));

function [vifp] = vifp_mscale(ref,dist)
% Input: (1) img1: The reference image as a matrix
%        (2) img2: The distorted image (order is important)
% Output: (1) VIF the visual information fidelity measure between the two images
% Advanced Usage:
% Users may want to modify the parameters in the code.
% (1) Modify sigma_nsq to find tune for your image dataset.

sigma_nsq = 2;
num = 0;
den = 0;
for scale = 1:4
    N = 2^(4-scale+1)+1;
    win = fspecial('gaussian',N,N/5);
if (scale > 1)
    ref = filter2(win, ref, 'valid');
    dist = filter2(win, dist, 'valid');
    ref = ref(1:2:end, 1:2:end);
    dist = dist(1:2:end, 1:2:end);
end

mu1 = filter2(win, ref, 'valid');
mu2 = filter2(win, dist, 'valid');
mu1_sq = mu1.*mu1;
mu2_sq = mu2.*mu2;
mu1_mu2 = mu1.*mu2;
sigma1_sq = filter2(win, ref.*ref, 'valid') - mu1_sq;
sigma2_sq = filter2(win, dist.*dist, 'valid') - mu2_sq;
sigma12 = filter2(win, ref.*dist, 'valid') - mu1_mu2;

sigma1_sq(sigma1_sq<0)=0;
sigma2_sq(sigma2_sq<0)=0;

g = sigma12./(sigma1_sq+1e-10);
sv_sq = sigma2_sq-g.*sigma12;

g(sigma1_sq<1e-10)=0;
sv_sq(sigma1_sq<1e-10)=sigma2_sq(sigma1_sq<1e-10);
sigma1_sq(sigma1_sq<1e-10)=0;

g(sigma2_sq<1e-10)=0;
sv_sq(sigma2_sq<1e-10)=0;

sv_sq(g<0)=sigma2_sq(g<0);
g(g<0)=0;
sv_sq(sv_sq<=1e-10)=1e-10;

num = num + sum(sum(log10((1+g.^2.*sigma1_sq./((sv_sq+sigma_nsq)))
)));

end
vifp = num/den;

%====================================================================================================
% Calculation of PSNR-HVS-M and PSNR-HVS image quality measures
% PSNR-HVS-M is Peak Signal to Noise Ratio taking into account
% Contrast Sensitivity Function (CSF) and between-coefficient
% contrast masking of DCT basis functions
% PSNR-HVS is Peak Signal to Noise Ratio taking into account only CSF

% PSNR-HVS-M:
% [1] Nikolay Ponomarenko, Flavia Silvestri, Karen Egiazarian, Marco Carli,
% Jaakko Astola, Vladimir Lukin, "On between-coefficient contrast masking
% of DCT basis functions", CD-ROM Proceedings of the Third International
% Workshop on Video Processing and Quality Metrics for Consumer Electronics
% PSNR-HVS:
% M. Carli, New full-reference quality metrics based on HVS, CD-ROM
% Proceedings of the Second International Workshop on Video Processing
% and Quality Metrics, Scottsdale, USA, 2006, 4 p.
% Input : (1) img1: the first image being compared
%         (2) img2: the second image being compared
%         (3) wstep: step of 8x8 window to calculate DCT coefficients. Default value is 8.
% Output: (1) p_hvs_m: the PSNR-HVS-M value between 2 images.
%         If one of the images being compared is regarded as
perfect quality, then PSNR-HVS-M can be considered as the quality measure of the other image. If compared images are visually undistinguished, then PSNR-HVS-M = 100000.

(2) p_hvs: the PSNR-HVS value between 2 images.

% Default Usage:
% Given 2 test images img1 and img2, whose dynamic range is 0-255
% [p_hvs_m, p_hvs] = psnrhvs(img1, img2);

% See the results:
% p_hvs_m % Gives the PSNR-HVS-M value
% p_hvs  % Gives the PSNR-HVS value

LenXY = size(img1); LenY = LenXY(1); LenX = LenXY(2);

CSFCof = [1.608443, 2.339554, 2.573509, 1.608443,
          1.072295, 0.643377, 0.504610, 0.421887,
          2.144591, 2.144591, 1.838221, 1.354478,
          0.989811, 0.443708, 0.428918, 0.467911,
          1.838221, 1.979622, 1.608443, 1.072295,
          0.643377, 0.451493, 0.372972, 0.459555,
          1.838221, 1.513829, 1.169777, 0.887417,
          0.504610, 0.295806, 0.321689, 0.415082,
          1.429727, 1.169777, 0.695543, 0.459555,
          0.378457, 0.236102, 0.249855, 0.334222,
          1.072295, 0.735288, 0.467911, 0.402111,
          0.317717, 0.247453, 0.227744, 0.279729,
          0.525206, 0.402111, 0.329937, 0.295806,
          0.249855, 0.212687, 0.214459, 0.254803,
          0.357432, 0.279729, 0.270896, 0.262603,
          0.229778, 0.257351, 0.249855, 0.259950];
% see an explanation in [2]

MaskCof = [0.390625, 0.826446, 1.000000, 0.390625,
          0.173611, 0.062500, 0.038447, 0.026874,
          0.694444, 0.694444, 0.510204, 0.277008,
          0.147929, 0.029727, 0.027778, 0.033058,
          0.510204, 0.591716, 0.390625, 0.173611,
          0.062500, 0.030779, 0.021004, 0.031888;]
S1 = 0; S2 = 0; Num = 0;
X=1; Y=1;
while Y <= LenY-7
    while X <= LenX-7
        A = img1(Y:(Y+7),X:(X+7));
        B = img2(Y:(Y+7),X:(X+7));
        A_dct = dct2(A); B_dct = dct2(B);
        MaskA = maskeff(A,A_dct,MaskCof);
        MaskB = maskeff(B,B_dct,MaskCof);
        if MaskB > MaskA
            MaskA = MaskB;
        end
        X = X + step;
    end
    for k = 1:8
        for l = 1:8
            u = abs(A_dct(k,l)-B_dct(k,l));
            S2 = S2 + (u*CSFCof(k,l)).^2;  % PSNR-HVS
            if (k~=1) | (l~=1) % See equation 3 in [1]
                if u < MaskA/MaskCof(k,l)
                    u = 0;
                else
                    u = u - MaskA/MaskCof(k,l);
                end
            end
        end
        S1 = S1 + (u*CSFCof(k,l)).^2;  % PSNR-HVS-M
    end
end
X = 1; Y = Y + step;
end

if Num ~=0
    S1 = S1/Num; S2 = S2/Num;
if S1 == 0
    p_hvs_m = 100000; % img1 and img2 are visually undistinguished
else
    p_hvs_m = 10*log10(255*255/S1);
end
if S2 == 0
    p_hvs = 100000; % img1 and img2 are identical
else
    p_hvs = 10*log10(255*255/S2);
end

function [m] = maskeff(z,zdct,MaskCof)
% Calculation of Enorm value (see [1])
m = 0;
for k = 1:8
    for l = 1:8
        if (k~=1) | (l~=1)
            m = m + (zdct(k,l).^2) * MaskCof(k,l);
        end
    end
end
pop=vari(z);
if pop ~= 0
    pop=(vari(z(1:4,1:4))+vari(z(1:4,5:8))+vari(z(5:8,5:8))+vari(z(5:8,1:4)))/pop;
end
m = sqrt(m*pop)/32; % sqrt(m*pop/16/64)

function [d] = vari(AA)
d=var(AA(:))*length(AA(:));
function [MI] = minf2(im1,im2)

%%% MUTUAL INFORMATION: compute the mutual information between two images
%%% mutualinformation( im1, im2 )
%%% im1: grayscale or color image, or 1-D signal
%%% im2: must be same size as im1

bins = [0:8:255];
N = length(bins);

%%% JOINT HISTOGRAM
joint = zeros( N, N);
X1 = im1(:);
X2 = im2(:);
maxval = max([X1; X2]);
X1 = round(X1 * (N-1)/maxval) + 1;
X2 = round(X2 * (N-1)/maxval) + 1;
for k = 1 : length(X1)
    joint(X1(k), X2(k)) = joint(X1(k), X2(k)) + 1;
end
joint = joint / sum(joint(:));

%%% MARGINALS
NX2 = sum(joint);
NX1 = sum(joint');

%%% MUTUAL INFORMATION
MI = 0;
for i = 1 : N
    for j = 1 : N
        if( joint(i,j)>eps & NX1(i)>eps & NX2(j)>eps )
            MI = MI + joint(i,j) * log(joint(i,j)/NX1(i)*NX2(j));
        end
    end
end
function [quality] = uiqi(img1, img2, block_size)
N = block_size.^2;
sum2_filter = ones(block_size);

img1_sq = img1.*img1;
img2_sq = img2.*img2;
img12 = img1.*img2;

img1_sum = filter2(sum2_filter, img1, 'valid');
img2_sum = filter2(sum2_filter, img2, 'valid');
img1_sq_sum = filter2(sum2_filter, img1_sq, 'valid');
img2_sq_sum = filter2(sum2_filter, img2_sq, 'valid');
img12_sum = filter2(sum2_filter, img12, 'valid');

img12_sum_mul = img1_sum.*img2_sum;
img12_sq_sum_mul = img1_sum.*img1_sum + img2_sum.*img2_sum;
numerator = 4*(N*img12_sum - img12_sum_mul).*img12_sum_mul;
denominator1 = N*(img1_sq_sum + img2_sq_sum) - img12_sq_sum_mul;
denominator = denominator1.*img12_sq_sum_mul;

quality_map = ones(size(denominator));
index = (denominator1 == 0) & (img12_sq_sum_mul ~= 0);
quality_map(index) = 2*img12_sum_mul(index)./img12_sq_sum_mul(index);
index = (denominator ~= 0);
quality_map(index) = numerator(index)./denominator(index);
quality = mean2(quality_map);

%==========================================================

function [scaMeasure] = SVDQualityMeasure(refImg, distImg, blkSize)

%Program for SVD-based Image Quality Measure

% Aleksandr Shnayderman, Alexander Gusev, and Ahmet M. Eskicioglu,
% "An SVD-Based Grayscale Image Quality Measure for Local and Global Assessment",
% IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 15, NO. 2, FEBRUARY 2006.

% Parameters
% refImg - Input Reference Gray Image
% distImg - Input Distorted Gray Image
% blkSize - Window size for block processing
% graMeasure - Graphical Image quality measure
% scaMeasure - Numerical Image quality measure

k = size(refImg, 1);
blkx = blkSize;
blkx = blkSize;
blockwise1 = MatDec(refImg, blkx);
blockwise2 = MatDec(distImg, blkx);
[blkx blkx imgx imgy] = size(blockwise1);
graMeasure = zeros(imgx, imgy);
blockwise1 = double(blockwise1);
blockwise2 = double(blockwise2);

for i = 1:imgx
    for j = 1:imgy
        temp_in_image = blockwise1(:,:,i,j);
        original_img = reshape(temp_in_image, blkx, blkx);
        temp_dist_image = blockwise2(:,:,i,j);
        distorted_img = reshape(temp_dist_image, blkx, blkx);
        graMeasure(i, j) = sqrt(sum((svd(original_img) -
        svd(distorted_img)).^2));
    end
end

graMeasure = round((graMeasure / max(max(graMeasure)))) * 255;
scaMeasure = sum(sum(abs(graMeasure -
        median(median(graMeasure))))) / ((k / blkx).^2);

function out = MatDec(inImg, blkSize)
% This program decomposes an image into blocks.
% Parameters
% inImg - Input Gray Image
% blkSize - Window size for block processing
% out - Output 4 dimensional matrix with blocks.
\[ [m,n] = \text{size}(\text{inImg}); \]
\[ r3 = \frac{m}{\text{blkSize}}; \]
\[ c3 = \frac{n}{\text{blkSize}}; \]
\[ q4 = 0; \]
\[ q1 = 0; \]

\begin{verbatim}
for i=1:r3
    for j=1:c3
        for s=1:blkSize
            for k=1:blkSize
                p3=s+q4;
                q2=k+q1;
                out(s,k,i,j)=inImg(p3,q2);
            end
        end
        q1=q1+blkSize;
    end
    q4=q4+blkSize; q1=0;
end
\end{verbatim}

%==================================================================
Appendix D

Python code that performs sequential Image Fusion

# Mohamed AbouRayan
# March 1, 2016
# Image fusion algorithms using 4 methods
# 1-Average Method
# 2-Discrete wavelet transformation (DWT) -- Daubechies
# 3-Discrete wavelet transformation (DWT) -- Haar
# 4-Principal Component Analysis (PCA) Method

# How to use:
# the user will be prompt to choose the desired algorithm and asked to choose
# two fit/fits file to fuse and a location and name for the new fused image

# Inputs:
# The program will ask for 2 file inputs and 2 other inputs choices
# 1- Desiered algorithm to perform
# 2- The location of the first fit/fits image file
# 3- The location of the second fit/fits image file
# 4- The location of the newly created fused image

# Output:
# The program will only output a fused image fit/fits file in the location the user choose
# importing packages used
import pywt
import numpy as np
import tkinter as tk
from astropy.io import fits
from scipy import linalg as LA
from tkinter import filedialog

# The Python function that performs Image fusion using the average method
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)

def Fusion_avg(Image1, Image2):
    # averaging both images
    result = (Image1 + Image2) / 2
    return result

# The Python function that performs Image fusion using principal component analysis (PCA)
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)

def Fusion_PCA(Image1, Image2):
    # Converting Image data to numpy Array to be able to do necessary calculation
    A = np.array(Image1)
    B = np.array(Image2)
    # getting Image dimensions
    temp1 = A.shape
    temp2 = B.shape
    # Starting PCA algorithm
    # creating matrix with both Images
    vector1 = np.reshape(A, temp1[0] * temp1[1], order='F')
    vector2 = np.reshape(B, temp2[0] * temp2[1], order='F')
    # Convolution of created matrix
    c = np.cov(vector1, vector2)
    # getting Eigenvalue and Eigenvector of this matrix
    D, V = LA.eig(c)
    sum1 = np.sum(V, axis=0)
# Calculating PCA

```python
if D[0] >= D[1]:
    pca = np.divide(V[:,0],sum1[0])
else:
    pca = np.divide(V[:,1],sum1[1])
```

# Creating fused image

```python
result = (pca[0]*Image1) + (pca[1]*Image2)
return result
```

# The Python function that performs Image fusion using Discrete wavelet transform (DWT) -- Daubechies

```python
def Fusion_DWT_db2(Image1, Image2):
    # decomposing each image using Discrete wavelet transform (DWT) -- Daubechies
    coeffs1 = pywt.wavedec2(Image1, 'db2', level=2)
    coeffs2 = pywt.wavedec2(Image2, 'db2', level=2)
    # creating variables to be used
    coeffs_H = list(coeffs1)
    # fusing the decomposed image data
    coeffs_H[0] = (coeffs1[0] + coeffs2[0]) * 0.5
    # creating variables to be used
    temp1 = list(coeffs1[1])
    temp2 = list(coeffs2[1])
    temp3 = list(coeffs_H[1])
    # fusing the decomposed image data
    temp3[0] = (temp1[0] + temp2[0]) * 0.5
    temp3[1] = (temp1[1] + temp2[1]) * 0.5
    coeffs_H[1] = tuple(temp3)
    # Creating fused image by reconstructing the fused decomposed image
    result = pywt.waverec2(coeffs_H, 'db2')
    return result
```

# The Python function that performs Image fusion using Discrete wavelet transform (DWT) -- Haar

```python
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)
```
### Fusion_DWT_Haar

```python
def Fusion_DWT_Haar(Image1, Image2):
    # decomposing each image using Discrete wavelet transform (DWT) -- Haar
    coeffs1 = pywt.wavedec2(Image1, 'haar', level=2)
    coeffs2 = pywt.wavedec2(Image2, 'haar', level=2)

    # creating variables to be used
    coeffs_H = list(coeffs1)
    # fusing the decomposed image data
    coeffs_H[0] = (coeffs1[0] + coeffs2[0]) * 0.5
    # creating variables to be used
    temp1 = list(coeffs1[1])
    temp2 = list(coeffs2[1])
    temp3 = list(coeffs_H[1])
    # fusing the decomposed image data
    temp3[0] = (temp1[0] + temp2[0]) * 0.5
    temp3[1] = (temp1[1] + temp2[1]) * 0.5
    coeffs_H[1] = tuple(temp3)

    # Creating fused image by reconstructing the fuesed decomposed image
    result = pywt.waverec2(coeffs_H, 'haar')
    return result
```

### The Main program

```python
# The Main program
# Print information regarding the program
print(""" This program performs Image fusion using Python you will be prompt to choose the fusion method then you will need to select two fit,fits image for input and finally you will need to specify where the output fused image should be locatated

    # Main Menu#
    Please choose from the following algorithms to perform Image Fusion
    1-Average Method
    2-Discrete wavelet transformation (DWT) -- Daubechies
    3-Discrete wavelet transformation (DWT) -- Haar
    4-Principal Component Analysis (PCA) Method """)
```

142
# Ask user to choose the image fusion algorithm to perform
while True:
    choice = int(input("Please choose which Image Fusion Algorithm to perform:"))
    if choice >= 1 and choice <= 4:
        break
    else:
        print("Error .. please choose a number between 1 and 4")
# Ask user to choose two files to be processed
print("Now you will be asked to choose two fit/fits image file to be processed")
root = tk.Tk()
root.withdraw()
file_path1 = filedialog.askopenfilename()
file_path2 = filedialog.askopenfilename()
# Opening FITS\FIT files
fileLocation1 = fits.open(file_path1)
fileLocation2 = fits.open(file_path2)
# fetching Image data from the FITS\FIT files
fitsImage1 = fileLocation1[0].data
fitsImage2 = fileLocation2[0].data
# Fuse the Image with the requested algorithm
if choice == '1':
    fusedImage = Fusion_avg(fitsImage1, fitsImage2)
elif choice == '2':
    fusedImage = Fusion_DWT_db2(fitsImage1, fitsImage2)
elif choice == '3':
    fusedImage = Fusion_DWT_Haar(fitsImage1, fitsImage2)
elif choice == '4':
    fusedImage = Fusion_PCA(fitsImage1, fitsImage2)
else:
    fusedImage = Fusion_avg(fitsImage1, fitsImage2)
# ask the user to choose a file location and name for the fused image to be saved
print("Please specify the location that the fused image can be saved")
file_path3 =
filedialog.asksaveasfilename(defaultextension='.fit')
# creating and saving the fused image
hdu = fits.PrimaryHDU(fusedImage)
hdu.writeto(file_path3)
Appendix E

Python code that performs Image Fusion in Parallel using Python’s multiprocessing package

# Mohamed AbouRayan
# March 20, 2016
# Image fusion algorithms using 4 methods
# The algorithm will be applied in parallel using Python's multiprocessing package
# 1 - Average Method
# 2 - Discrete wavelet transformation (DWT) -- Daubechies
# 3 - Discrete wavelet transformation (DWT) -- Haar
# 4 - Principal Component Analysis (PCA) Method

# How to use:
# the user will be prompt to choose the desired algorithm and asked to choose
# two fit/fits file to fuse and a location and name for the new fused image

# Inputs:
# The program will ask for 2 file inputs and 2 other inputs choices
# 1 - Desired algorithm to perform
# 2 - The location of the first fit/fits image file
# 3 - The location of the second fit/fits image file
# 4 - The location of the newly created fused image

# Output
# The program will only output a fused image fit/fits file in the location the user choose

# importing packages used
from astropy.io import fits
import tkinter as tk
from tkinter import filedialog
import multiprocessing as mp
import pywt
import numpy as np
from scipy import linalg as LA
from multiprocessing import Array

# The Python function that performs Image fusion using the average method
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)

def Fusion_avg(Image1, Image2):
    # averaging both images
    result = (Image1 + Image2) / 2
    outputs.Value = result
    return result

# The Python function that performs Image fusion using principal component analysis (PCA)
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)

def Fusion_PCA(Image1, Image2):
    # Converting Image data to numpy Array to be able to do necessary calculation
    A = np.array(Image1)
    B = np.array(Image2)
    # getting Image dimensions
    temp1 = A.shape
    temp2 = B.shape
    # Starting PCA algorithm
    # creating matrix with both Images
vector1=np.reshape(A,temp[0]*temp[1],order='F')
vector2=np.reshape(B,temp[0]*temp[1],order='F')
# Convolution of created matrix
c = np.cov(vector1,vector2)
# getting Eigenvalue and Eigenvector of this matrix
D, V = LA.eig(c)
sum1=np.sum(V,axis=0)
# Calculating PCA
if D[0] >= D[1]:
    pca = np.divide(V[:,0],sum1[0])
else:
    pca = np.divide(V[:,1],sum1[1])
# Creating fused image
result =(pca[0]*Image1) + (pca[1]*Image2)
outputs.Value=result
return result

########################################################################
# The Python function that performs Image fusion using
# Discrete wavelet transform (DWT) -- Daubechies
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)
########################################################################
def Fusion_DWT_db2(Image1,Image2):
    # decomposing each image using Discrete wavelet
    transform (DWT) -- Daubechies
    coeffs1=pywt.wavedec2(Image1, 'db2', level=2)
    coeffs2=pywt.wavedec2(Image2, 'db2', level=2)
    # creating variables to be used
    coeffs_H=list(coeffs1)
    # fusing the decomposed image data
    coeffs_H[0]=(coeffs1[0]+coeffs2[0]) * 0.5
    # creating variables to be used
    temp1=list(coeffs1[1])
    temp2=list(coeffs2[1])
    temp3=list(coeffs_H[1])
    # fusing the decomposed image data
    temp3[0]=(temp1[0]+temp2[0]) * 0.5
    temp3[1]=(temp1[1]+temp2[1]) * 0.5
    coeffs_H[1]=tuple(temp3)
    # Creating fused image by reconstructing the fuesed
    decomposed image
result = pywt.waverec2(coeffs_H, 'db2')
outputs.Value = result
return result

# The Python function that performs Image fusion using Discrete wavelet transform (DWT) -- Haar
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)
def Fusion_DWT_Haar(Image1, Image2):
    # decomposing each image using Discrete wavelet transform (DWT) -- Haar
    coeffs1 = pywt.wavedec2(Image1, 'haar', level=2)
    coeffs2 = pywt.wavedec2(Image2, 'haar', level=2)
    # creating variables to be used
    coeffs_H = list(coeffs1)
    # fusing the decomposed image data
    coeffs_H[0] = (coeffs1[0] + coeffs2[0]) * 0.5
    # creating variables to be used
    temp1 = list(coeffs1[1])
    temp2 = list(coeffs2[1])
    temp3 = list(coeffs_H[1])
    # fusing the decomposed image data
    temp3[0] = (temp1[0] + temp2[0]) * 0.5
    temp3[1] = (temp1[1] + temp2[1]) * 0.5
    coeffs_H[1] = tuple(temp3)
    # Creating fused image by reconstructing the fuesed decomposed image
    result = pywt.waverec2(coeffs_H, 'haar')
    outputs.Value = result
    return result

# The Main program
def main():
    # Print information regarding the program
    print('This program performs Image fusion using Python you will be prompt to choose the fusion method')
then you will need to select two fit, fits image for input
and finally you will need to specify where the output fused image should be located.

# Main Menu

Please choose from the following algorithms to perform Image Fusion:

1. Average Method
2. Discrete wavelet transformation (DWT) -- Daubechies
3. Discrete wavelet transformation (DWT) -- Haar
4. Principal Component Analysis (PCA) Method

# Ask user to choose the image fusion algorithm to perform

```python
while True:
    choice = int(input("Please choose which Image Fusion Algorithm to perform:"))
    if choice >= 1 and choice <= 4:
        break
    else: print("Error .. please choose a number between 1 and 4")
```

# Ask user to choose two files to be processed

```python
print("Now you will be asked to choose two fit/fits image file to be processed")
```

```python
root = tk.Tk()
root.withdraw()
file_path1 = filedialog.askopenfilename()
file_path2 = filedialog.askopenfilename()
```

# Opening FITS\FIT files

```python
fileLocation1 = fits.open(file_path1)
fileLocation2 = fits.open(file_path2)
```

# fetching Image data from the FITS\FIT files

```python
fitsImage1 = fileLocation1[0].data
fitsImage2 = fileLocation2[0].data
```

# setting up variables to be used

```python
outputs = Array('i', range(len(fitsImage1)))
```

# Creating fused image

```python
if choice == '1':
    processes = [mp.Process(target=Fusion_avg, args=(fitsImage1, fitsImage2)) for x in range(2)]
ellif choice == '2':
```

148
processes = [mp.Process(target=Fusion_DWT_db2, args=(fitsImage1, fitsImage2)) for x in range(2)]
    elif choice=='3':
        processes = [mp.Process(target=Fusion_DWT_Haar, args=(fitsImage1, fitsImage2)) for x in range(2)]
    elif choice=='4':
        processes = [mp.Process(target=Fusion_PCA, args=(fitsImage1, fitsImage2)) for x in range(2)]
    else:
        processes = [mp.Process(target=Fusion_avg, args=(fitsImage1, fitsImage2)) for x in range(2)]
    for p in processes:
        p.start()
    # Exit the completed processes
    for p in processes:
        p.join()
    # ask the user to choose a file location and name for the fused image to be saved
    print("Please specify the location that the fused image can be saved")
    file_path3 = filedialog.asksaveasfilename(defaultextension='.fit')
    # creating and saving the fused image
    hdu = fits.PrimaryHDU(outputs)
    hdu.writeto(file_path3)
    # This part of the code is needed to be able to work on windows machines
if __name__ == '__main__':
    main()
Appendix F

Python code that performs Image Fusion in Parallel using joblib package

# Mohamed AbouRayan
# March 20, 2016
# Image fusion algorithms using 4 methods
# The algorithm will be applied in parallel using joblib package
# 1- Average Method
# 2- Discrete wavelet transformation (DWT) -- Daubechies
# 3- Discrete wavelet transformation (DWT) -- Haar
# 4- Principal Component Analysis (PCA) Method

How to use:
# the user will be prompt to choose the desired algorithm and asked to choose
# two fit/fits file to fuse and a location and name for the new fused image

Inputs:
# The program will ask for 2 file inputs and 2 other inputs choices
# 1- Desired algorithm to perform
# 2- The location of the first fit/fits image file
# 3- The location of the second fit/fits image file
# 4- The location of the newly created fused image

Output
# The program will only output a fused image fit/fits file in the location the user choose

# importing packages used
from joblib import Parallel, delayed
import multiprocessing
from astropy.io import fits
import time
import pywt
import numpy as np
import tkinter as tk
from scipy import linalg as LA
from tkinter import filedialog

# The Python function that performs Image fusion using the average method
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)

def Fusion_avg(Image1, Image2):
    global outputs
    # averaging both images
    result = (Image1 + Image2) / 2
    outputs = result
    return result

# The Python function that performs Image fusion using principal component analysis (PCA)
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)

def Fusion_PCA(Image1, Image2):
    global outputs
    # Converting Image data to numpy Array to be able to do necessary calculation
    A = np.array(Image1)
    B = np.array(Image2)
    # getting Image dimensions
    temp1 = A.shape
    temp2 = B.shape
    # Starting PCA algorithm
# creating matrix with both Images
vector1 = np.reshape(A, temp1[0] * temp1[1], order='F')
vector2 = np.reshape(B, temp2[0] * temp2[1], order='F')

# Convolution of created matrix
c = np.cov(vector1, vector2)

# getting Eigenvalue and Eigenvector of this matrix
D, V = LA.eig(c)
sum1 = np.sum(V, axis=0)

# Calculating PCA
if D[0] >= D[1]:
    pca = np.divide(V[:, 0], sum1[0])
else:
    pca = np.divide(V[:, 1], sum1[1])

# Creating fused image
result = (pca[0] * Image1) + (pca[1] * Image2)
outputs.Value = result
return result

#############################################################################
# The Python function that performs Image fusion using Discrete wavelet transform (DWT) -- Daubechies
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)
#############################################################################

def Fusion_DWT_db2(Image1, Image2):
    global outputs
    # decomposing each image using Discrete wavelet transform (DWT) -- Daubechies
    coeffs1 = pywt.wavedec2(Image1, 'db2', level=2)
    coeffs2 = pywt.wavedec2(Image2, 'db2', level=2)

    # creating variables to be used
    coeffs_H = list(coeffs1)

    # fusing the decomposed image data
    coeffs_H[0] = (coeffs1[0] + coeffs2[0]) * 0.5

    # creating variables to be used
    temp1 = list(coeffs1[1])
    temp2 = list(coeffs2[1])
    temp3 = list(coeffs_H[1])

    # fusing the decomposed image data
    temp3[0] = (temp1[0] + temp2[0]) * 0.5
    temp3[1] = (temp1[1] + temp2[1]) * 0.5
    coeffs_H[1] = tuple(temp3)

    return result
# Creating fused image by reconstructing the fuesed decomposed image
result=pywt.waverec2(coeffs_H, 'db2')
outputs.Value=result
return result

# The Python function that performs Image fusion using Discrete wavelet transform (DWT) -- Haar
# input: two Images dataset (matrix) to be fused
# output: one Fused Image dataset (matrix)
def Fusion_DWT_Haar(Image1,Image2):
    global outputs
    # decomposing each image using Discrete wavelet transform (DWT) -- Haar
    coeffs1=pywt.wavedec2(Image1, 'haar', level=2)
    coeffs2=pywt.wavedec2(Image2, 'haar', level=2)
    # creating variables to be used
    coeffs_H=list(coeffs1)
    # fusing the decomposed image data
    coeffs_H[0]=(coeffs1[0]+coeffs2[0]) * 0.5
    # creating variables to be used
    temp1=list(coeffs1[1])
    temp2=list(coeffs2[1])
    temp3=list(coeffs_H[1])
    # fusing the decomposed image data
    temp3[0]=(temp1[0]+temp2[0]) * 0.5
    temp3[1]=(temp1[1]+temp2[1]) * 0.5
    coeffs_H[1]=tuple(temp3)
    # Creating fused image by reconstructing the fuesed decomposed image
    result=pywt.waverec2(coeffs_H, 'haar')
    outputs.Value=result
    return result

# The Python function that controls the Image fusion and ensure the parallel processing
# input: two Images dataset (matrix) to be fused and function to be used and how many iterations
# output: one Fused Image dataset (matrix)
def test(Image1, Image2, x, y):
    num_cores = multiprocessing.cpu_count()
    start_time = time.time()
    Parallel(n_jobs=num_cores)(delayed(y)(Image1, Image2)
    for i in range(x))
    print("---%s running at %s times is: %s seconds ---" %
    (y, x, (time.time() - start_time)))

###

# The Main program
###
def main():
    global outputs
    # Print information regarding the program
    print(""" This program performs Image fusion using Python you will be prompt to choose the fusion method then you will need to select two fit,fits image for input and finally you will need to specify where the output fused image should be locatated
    # Main Menu#
    Please choose from the following algorithms to perform Image Fusion
    1-Average Method
    2-Discrete wavelet transformation (DWT) -- Daubechies
    3-Discrete wavelet transformation (DWT) -- Haar
    4-Principal Component Analysis (PCA) Method """

    # Ask use to choose the image fusion algorithm to perform
    while True:
        choice=int(input("Please choose which Image Fusion Algorithm to perform:"))
        if choice>=1 and choice<=4:
            break
        else:print("Error .. please choose a number between 1 and 4")
    # Ask user to choose two files to be processed

154
print("Now you will be asked to choose two fit.fits image file to be processed")
root = tk.Tk()
root.withdraw()
file_path1 = filedialog.askopenfilename()
file_path2 = filedialog.askopenfilename()
# Opening FITS\FIT files
fileLocation1 = fits.open(file_path1)
fileLocation2 = fits.open(file_path2)
# fetching Image data from the FITS\FIT files
fitsImage1 = fileLocation1[0].data
fitsImage2 = fileLocation2[0].data
# setting up variables to be used
outputs=Array('f', range(len(fitsImage1)))
# trying to figure out how many cores the system have
num_cores = multiprocessing.cpu_count()
print("numCores = " + str(num_cores))
# Fuse the Image with the requested algoritm
if choice=='1':
    test(fitsImage1,fitsImage2,10,Fusion_avg)
elif choice=='2':
    test(fitsImage1,fitsImage2,10,Fusion_DWT_db2)
elif choice=='3':
    test(fitsImage1,fitsImage2,10,Fusion_DWT_Haar)
elif choice=='4':
    test(fitsImage1,fitsImage2,10,Fusion_PCA)
else:
    test(fitsImage1,fitsImage2,10,Fusion_avg)
# ask the user to choose a file location and name for
# the fused image to be saved
print("Please specify the location that the fused image can be saved")
file_path3 =
filedialog.asksaveasfilename(defaultextension='.fit')
# creating and saving the fused image
hdu = fits.PrimaryHDU(outputs)
hdu.writeto(file_path3)
outputs=[]
# This part of the code is needed to be able to work on
windows machines
if __name__ == '__main__':
    main()
Appendix G

MATLAB script for Image Characterization

% Mohamed AbouRayan
% February 17, 2016
% Image Characterization using Satellite Tracks in Astronomical Images

How to use:
% user need to manual change the two variables fileLocation1 and fileLocation2 with the request image location

Inputs:
% this MATLAB script takes two Images in FIT/FITS/FTS formats as input

Output: The probability of these two images came from the same source

Acknowledgment:
% This code was inspired by the "Satellite Tracks Removal in Astronomical Images" paper
% but was implemented using enhanced technique than described in the paper (Multilevel image thresholds instead of Binarization)
% Stores desired file location into a variable to be easily reused
fileLocation1 = 'E:\Image Fusion\Matlab code & data files\FITS\Autosave Image -001B.fit';
fileLocation2 = 'E:\Image Fusion\Matlab code & data files\FITS\Autosave Image -004B.fit';

% Read data from FITS files and store it to a variable FITS Image
fitsImage1 = fitsread(fileLocation1);
fitsImage2 = fitsread(fileLocation2);

% Normalizing the Images and converting to double precision for processing
tryImage1 = im2double(fitsImage1);
tryImage2 = im2double(fitsImage2);

% Apply 2-D adaptive noise-removal filtering
[K1,noise1] = wiener2(tryImage1,[500 500]);
[K2,noise2] = wiener2(tryImage2,[500 500]);

% Getting the noise from original Image
noise11 = tryImage1 - K1;
noise22 = tryImage2 - K2;

% Setting up variables to be used
count = 0;
noncount = 0;
[m, n] = size(noise22);

% Testing for existence of noise and comparing intensity levels (Multilevel image thresholds) within a given range (Instead of Binarization)
for i=1:m;
    for j=1:n;
        if abs(noise22(i,j)-noise11(i,j))<80
            count = count + 1;
        else
            noncount = noncount + 1;
        end
    end
end
% The resulting probability of the algorithm
Result = 100 * count / (count + noncount)