REAL TIME PROCESSING OF HYPERSPECTRAL IMAGES

Stefan Robila
Center for Imaging and Optics
Department of Computer Science
Montclair State University
Richardson Hall 301
Upper Montclair, NJ 07043
robilas@mail.montclair.edu

ABSTRACT

We describe the development of a real-time processing tool for hyperspectral imagery based on off-the-shelf equipment and higher level programming language implementation (C++ and Java). The algorithms we developed are derived from previously introduced spectra matching and feature extraction tools. The first group is based on spectra identification and spectral screening, a method that allows the identification of representative spectra from a data set. The second group is based on Principal Component Analysis (PCA) and Independent Component Analysis (ICA). When applied to multidimensional data, PCA linearly transforms them such that the resulting components are uncorrelated and their variance maximized. In ICA, given a linear mixture of statistical independent sources, the goal is to recover these components by producing an unmixing matrix. The effectiveness of the proposed real time algorithms were tested on an in-house system composed of a commercially available hyperspectral camera and a multiprocessor computer system. Preliminary results targeted at the feasibility of the tool show that reasonable accuracy can be maintained in the real time requirements. The described project supports the further development of hyperspectral imaging as a general tool in remote sensing.

INTRODUCTION

Hyperspectral images are based on the division of the light spectrum in intervals of wavelengths and collecting image data using these intervals. Each element from the image (pixel) is associated with a certain area of the scene surveyed and with its spectral response (see Figure 1). Grouping spectral images over several wavelength intervals for the same scene results in *multispectral* (in case of few wide spectral bands) or *hyperspectral* (in case of many narrow spectral bands) *images* (Lillesand, 2000). Usually, passive hyperspectral sensors cover wavelengths from the visible range (0.4μm-0.7μm) to the middle infrared range (2.4μm) (Richards, 1999).

The richness of information available in a hyperspectral image becomes clear when we plot the pixel vectors. Figure 2a shows a hyperspectral image band collected with the HYDICE sensor (Hydice 1995). Eight rows of panels are indicated by marks of various colors in the image, with each row containing three panels of 1mx1m, 2mx2m and 3mx3m, respectively, made of the same material. While the panels are hard to distinguish within most of the image bands, the plot of their pixel vectors displays significant differences for each. A processing tool that is able to distinguish the materials based on spectral dissimilarities will provide significantly better results than any analysis of the individual images. These characteristics have allowed hyperspectral data to be successfully employed in varied fields including agricultural monitoring, mineral identification, military, emergency planning, and medical and industrial imaging,

At the same time, due to the large size, hyperspectral data processing usually requires long execution times. This is of particular concern for real time environments where deadlines limit the processing times allowed to produce results. Recent efforts to solve the time constraints have included hardware solutions (such as on board processing or compression)(Bowles, 1997; Brilles, 1997, Chai, 2000), localized processing (i.e. processing based only on the current image line)(Chang, 2001; Du, 2003, Heinz, 2001), or data sampling (such as spectral screening) (Achalakul, 2000, Bowles, 2003). Some of these studies indicate the use of sensors and applications packages such as the ORASIS – (Bowles, 2003) or Dark Horse (Stellman, 2000) platforms, both requiring significant financial and human investments not available to small commercial or research group. Other efforts (Chang, 2001; Du, 2003, Heinz, 2001) base their results on simulations of real time environments or limit their applicability to multispectral (filter-wheel based) platforms (Achalakul, 2000).

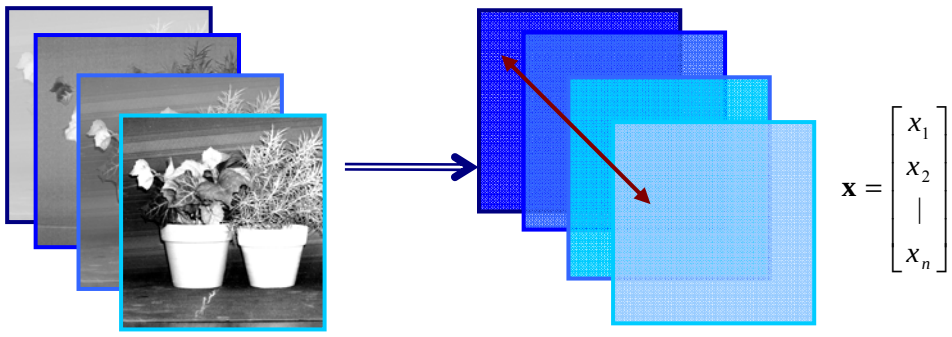
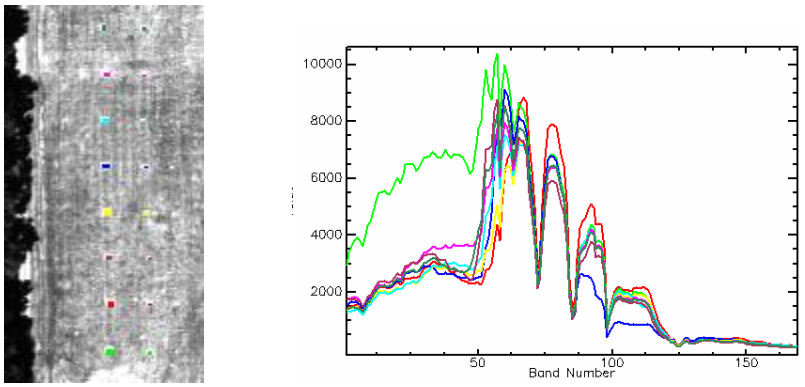


Figure 1. Pixel vectors are formed of the pixel values for the same coordinates.



(a) (b)

Figure 2. Pixel a) HYDICE data displaying rows of panels marked by squares, b) Average spectra for the eight panel categories in the scene

In this paper we present an integrated platform for real time processing that is based on both efficient algorithms and hardware speed. Our approach benefits from the recent availability of off-the-shelf hyperspectral sensors and their relatively modest prices as well as the increased RAM memory sizes, network and CPU speeds. The implementation takes in consideration the nature of the pushbroom sensor that allows us to process lines of the data while collecting the next lines. In addition, we use multithreading that allows us to take advantage of the multiprocessor technology available in current workstations. The main advantages of such system are the extremely reduced cost, the ease in extension and the flexibility in implementation. All are arguments supporting further use of hyperspectral sensors in both research and commercial applications.

REAL TIME ENVIRONMENT

A flexible hyperspectral sensing environment allows the sensor to be connected to various computing systems. At the same time, in order to support fast collection of data, all the hyperspectral sensors are designed to interact with specialized capture or graphics boards or modules. This may in turn limit the mobility of the sensor as well the upgrade options.

Figure 3 displays various configurations of the sensor and computer connectivity. When a capture board or hardware module is present, most probably this will be housed within a desktop system (a). To provide connectivity with a laptop, one must use an intermediate extension board that will further connect to the notebook through the PCMCIA or USB connections (b). When the main concern is fast processing, we would suggest the use of a computer network or cluster that will share on the processing tasks, with the main desktop serving as manager between sensor data collection, data processing and data displays (c).

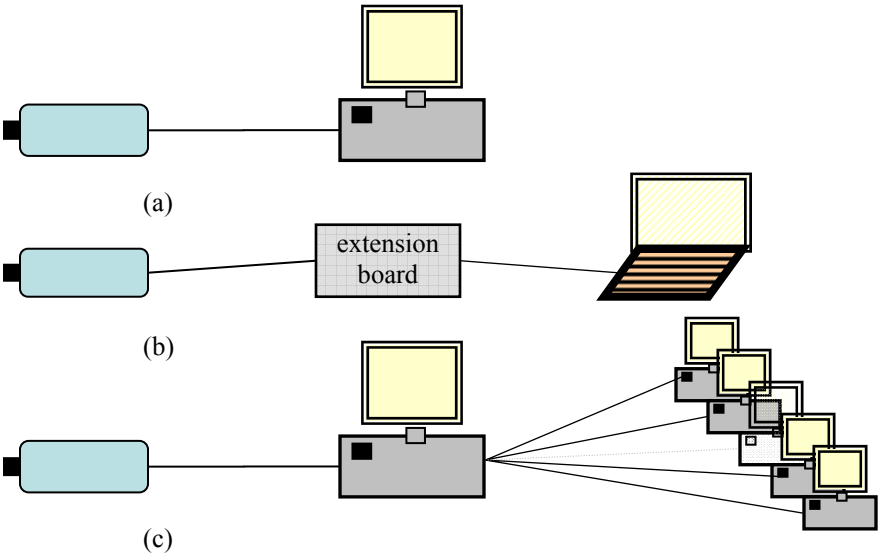


Figure 3. Various set-ups for real time data collection a) camera connected to a desktop b) camera communicating to laptop through an extension board c) camera connected to a desktop with the computationally intensive processing being supported by a multithreaded application running on a computer cluster.

Even in the later case, the master desktop will have to manage both data collection and distribution of jobs. In principle, both these tasks require CPU time and will compete for it as well as other system resources (such as system bus and memory). A possible solution is the use of capture boards that perform Direct Memory Access (DMA). DMA allows a device driver to communicate directly with the system memory without taking up CPU cycles. Most of the capture boards allow the creation of several DMA buffers leading to possibly multiple processes moving data to the memory at the same time. While this may be beneficial in some instances, one must remember that all these processes share the same communication bus (between the device and memory) and most probably will compete for it.

Another important characteristic is the way the data are collected by the sensor. Our approach focuses on pushbroom cameras, where one line of data is collected at a time. This will be beneficial for spectral matching since the match can be done on line by line basis. However, when the goal is processing based on data transform, the line by line collection can be used to distribute the processing load among various tasks.

Figure 4 displays our conceptual model based on multithreaded technology. A single process is started managing both the sensor and the data handling. Each specific task is handled by an individual thread. A *camera thread* is started at the initialization of the application and is responsible for setting and resetting camera parameters as well as for triggering DMA-based moves of the lines of data from the camera device to the memory. One or several *interface threads* ensure the communication between the user and application. A display thread will read data from the memory and display them on the screen. Two other threads were designed. The first is the matching thread that, based on a (*target*) spectra, a spectral distance measure and threshold value will mark as matched all the data spectra similar to the target. The *transform manager thread* is in charge of synchronizing the individual transform threads as they work on separate parts of the data. The number of transform threads could vary depending on the size of the data, on the number of systems available as well as the type of transform. We note that parallel and distributed processing algorithms for hyperspectral images have been previously presented (Robila, 2004). In most cases, the data are split in subcubes that are processed separately (see Figure 5). The master transform thread has the role of distributing the data, receiving the subcube partial results and combining them in full data partial results, and finally combining the full transform result. We note that the matching thread could also be extended to a multithread environment. Our experience has been that this slows down the real time display since the overhead introduced by communication has become larger than the speedup obtained by distributed processing of the data.

Synchronization among the various threads has to be provided to ensure that the collected data are accurately processed and displayed. Our application was implemented in Microsoft Visual C++ ® with the spectra saving and display component being implemented in Java. Win32 threads were used for the multithreaded implementation.

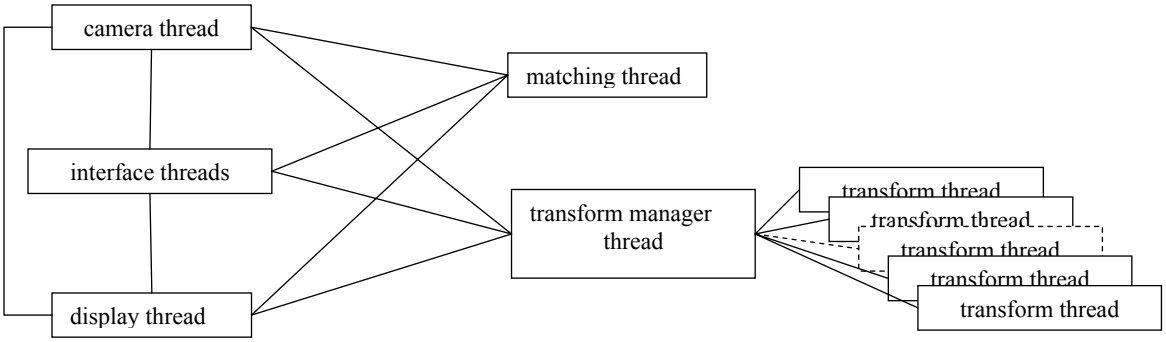


Figure 4. Map of the thread interactions within the real time application.

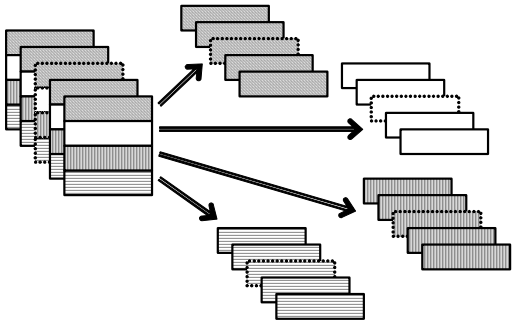


Figure 5. Multithreaded processing of the data is done by splitting the image cube in subcubes.

PROCESSING METHODS

Currently, we have completed the implementation of two types of processing methods: *spectral matching* and *data transforms*.

In the spectral matching, the user selects a target spectrum by either clicking on the displayed image, by entering the pixel coordinates or by loading it from a file. The user is also able to save the spectrum to the file and to display the spectral graph. The user is then able to select a spectral measure and a threshold. The camera is restarted and will display as marked with a specific color all the pixel locations that match the target spectra, i.e. which have the distance between them and the target less than the threshold. Currently, three methods based on three different spectral distances were implemented:

- *spectral angle (SA)* (Robila 2005b)*:*

$$SA(\mathbf{x},\mathbf{y}) = \arccos\left(\frac{\langle \mathbf{x},\mathbf{y}\rangle}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}\right) \quad (1)$$

(where **x** and **y** are two *n*-dimensional vectors, <.,.> represents the dot product of the vectors and $\|.\|_2$ represents the Euclidean norm)

- *Euclidean distance (ED)* (Keshava 2004)*:*

$$e(\mathbf{x},\mathbf{y}) = \sqrt{<\mathbf{x}-\mathbf{y},\mathbf{x}-\mathbf{y}>} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}\ . \quad (2)$$

- *Spectral Information Divergence (SID)* (Du 2004)

$$SID(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n} \left(\frac{x_i}{\sum_{j=1}^{n} x_j} - \frac{y_i}{\sum_{j=1}^{n} y_j}\right)\left(\log\frac{x_i}{\sum_{j=1}^{n} x_j} - \log\frac{y_i}{\sum_{j=1}^{n} y_j}\right). \quad (3)$$

SID is derived from the Kullback-Leibler information measure:

$$SID(\mathbf{x},\mathbf{y}) = D(\mathbf{x}\,||\,\mathbf{y}) + D(\mathbf{y}\,||\,\mathbf{x}) \quad (4)$$

For a detailed discussion on the properties of the various measures, refer to the above references or to (Robila 2005a)

On the data transform side, we implemented a distributed version of Principal Component Analysis (PCA) transform. When employed, the data are transformed as follows (Rencher 1995):

$$\mathbf{x}' = C_{\mathbf{x}}^{-\frac{1}{2}} A_{\mathbf{x}} (\mathbf{x} - |\,\mathbf{x}\,|) \quad (5)$$

where $C_{\mathbf{x}}$ is the diagonal matrix formed of the eigenvalues of the covariance matrix for the random vector $\mathbf{x}$, and $A_{\mathbf{x}}$ is the matrix formed of the corresponding eigenvectors.

If the eigenvectors are sorted according to the magnitudes of the eigenvalues, the components of **x'** will usually display most of the information in the first few frames (since after transforming through $A_{\mathbf{x}}$, they are the ones with the largest variance). In our application, we used the distributed version suggested in (Achalakul 2001) without the use of spectral screening. The bands associated with the highest three eigenvalues were combined to provide a color composite image. Given equation 5, one must have the entire data in order to compute the PCA transform. To counter this, we are assuming that any real time surveillance based on PCA will not be done based on sudden changes of the observed scene. As such, it would be reasonable to assume that the PCA transform of the previously collected scene provides a good approximation of the transform of the current scene. In our case, following the collection of the first image cube we compute the principal components. The next PCA based display is obtained by using these components on the new data. At the end of the collection of the data, the new transform is computed.

We note that the multithread model for the transform methods allows that each thread receive a subcube of the data (see Figure 5). Communication mechanisms were provided to ensure that the expected value, standard deviation and covariance matrix are correctly computed. Once this is done, the eigenvalues and the eigenvectors can be computed directly in the transform manager thread.

A second transform approach is based on Independent Component Analysis (ICA). Given an observed n-dimensional random vector **x**, ICA attempts to find a linear transformation **W** such that the obtained data **u** = **Wx** has the components that are independent. Statistical independence of the components can be expressed as (Hyvärinen 2001):

$$p(\mathbf{u}) = \prod_{i=1}^{n} p(u_i). \quad (6)$$

Distributed versions of the ICA exist. We are currently implementing and testing the approach described in (Robila 2004).

EXPERIMENTS

The developed real time application was designed to work with our available hyperspectral sensor SOC 700 ®. The camera, produced by Sensors Unlimited Spectral Band: collects 120 image bands within the 430nm-900nm wavelength interval at 640 12 bit pixels per line with a configurable number of lines, variable exposure time and line rate. A hyperspectral image collected by the sensor has approximately 90MB.

The camera (see Figure 6 for an example lab set-up) connects to the desktop through a proprietary PCI mountable capture board. The camera controls are done through serial cable and the data connection is done through regular Ethernet cable. The desktop, a Dual Intel Xeon 2.66 GHz with 2.00GB RAM was also equipped with a 3Dlabs Wildcat4 7110 Display Adapter (2) (320 MHz on board graphics processor, 128 MB on board memory). A cluster of Dell Intel Xeon 3.06 GHz workstations with 1.0 GB RAM each and 100Mbps network connectivity was connected to the main workstation to provide support with the transform threads structure. To allow for laptop connectivity, a Magma PCI/PCMCIA extension bridge was used. This device, popular within the video gaming community allows for the installation of a PCI card in a separate box and then connection of the box to a laptop through the PCMCIA slots.

As mentioned above, the application was written using C/C++ (the language used for the already available camera driver and interface software - PxelFly®) and Java (for the spectra display).. We note that both languages support multithreading and synchronization.

Figure 6. Experimental set-up.

Figure 7 shows various image results using the application. For comparison purposes, Figure 7a shows a regular color photo of the experimental scene. Two artificial plants in pots are placed on a black background. The left side plant also contains two natural leaves. The bright yellow spot in the center leave marks the pixel location used to select the target spectra. Figure 7c shows the target match result for threshold 0.095 and spectral angle was used as spectral measure. In this case, the sensor was run with exposure times of 10 microseconds. The display of data took approximately 10 seconds, a reasonable time, given that the number of columns (640) would require over 6 seconds for data capture alone. Further investigation revealed that while the spectral angle and the Euclidean distance do not have any significant execution time differences, the spectral information divergence was considerably slower. We attribute this to the higher complexity of the SID formula. We plan to revisit the code implementation in this case to determine if a more efficient approach is possible.

Figure 7c displays the result of the real time PCA. The application required considerably large time to compute the covariance matrix, leading to a sensible slowdown of the display. We also note that the significant size of the image cube (80MB) lead to a bottleneck situation in the communication of data between the master transform thread and the worker transform threads. Possible solutions to reduce the PCA execution time are discussed in the literature. One approach is the use of spectral screening described in (Achalakul, 2001; Bowles, 1997, and Robila, 2005b). However, the use of samples will reduce the results accuracy and may lead to inaccurate images. The implementation of ICA is currently under testing facing similar issues.

Finally, Figure 8 shows a screenshot of the real time hyperspectral processing application. The left side menu allows the interaction with the camera and the transform and matching options and constitutes an extension of the original PixelFly ® application. When the “Matching” button is clicked, the user is able to select the target, the spectral distance and the threshold. The matching will be done both in a single scan (collection and display of one image) or in continuous mode (once an image is processed, start acquiring the next image).

(a) (b) (c)

Figure 7. Experimental results a) Color image of two flower pots. b) Real Time display when using spectral matching with spectral angle, angle threshold 0.095 and the target spectra selected from the image c) PCA display.

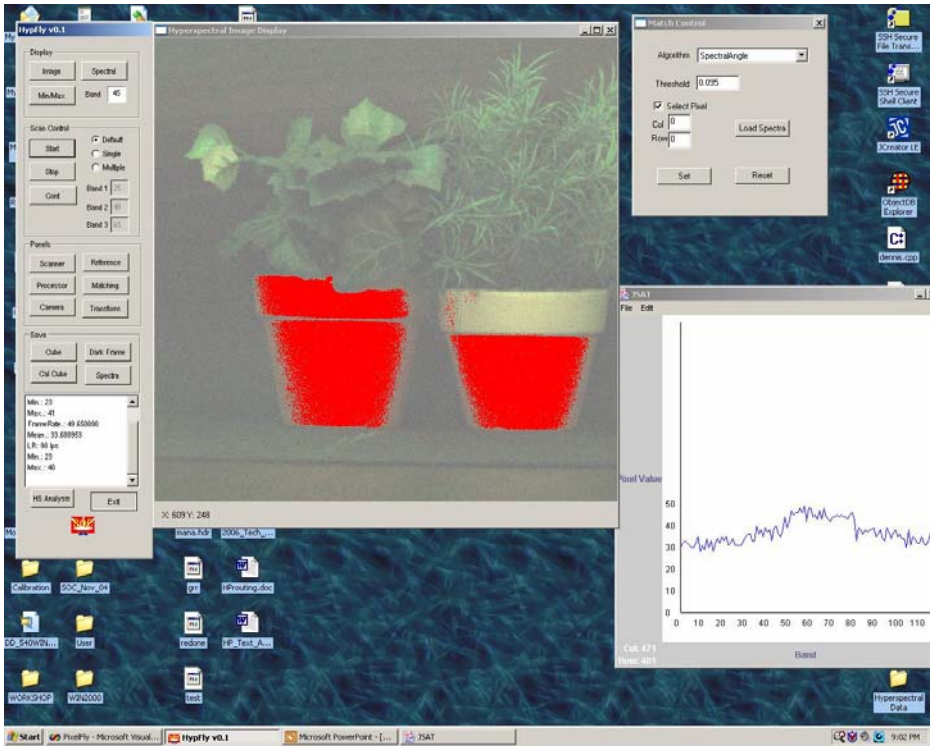


Figure 8. Real Time Hyperspectral Processing Environment

CONCLUSIONS

In this paper, we described the development of a real-time processing tool for hyperspectral imagery based on off-the-shelf equipment and higher level programming language implementation (C++ and Java). The environment has the advantage of extremely low cost compared with commercially available sensors or platforms.

To support our approach we developed and implemented two classes of algorithms based on previously introduced spectra matching and feature extraction tools. The choice for these two classes is due to their popularity within the remote sensing community. The first group is based on spectra identification and spectral screening, a method that allows the identification of representative spectra from a data set was implemented using spectral angle, Euclidean distance and spectral information divergence. Users are able to select target spectra from a hyperspectral image and then get a real time matching results of data with the target. The second group is based on Principal

Component Analysis (PCA) and Independent Component Analysis (ICA). In this case, real time constraints are harder to satisfy due to the large data size and to the fact that the data need to be fully collected before the main processing can begin. Our initial approach in using previously computed transforms allows us to display a result but the computation of the transform adds a significant overhead beyond the real time requirements.

In all, the system allows for reasonable accuracy. New computing technology directions, such as DMA, multithreading, high level languages and increased overall computing performance allow the system that we designed and maintained to perform in or near real time. This, in turn stimulates our drive to search for new applications of hyperspectral imagery.

ACKNOLWDGEMENT

The project was supported by a 2005 SBR Grant awarded by Montclair State University. We would like to thank Sensors Unlimited for providing the source code of the PixelFly® application

REFERENCES

Achalakul T., and S. Taylor (2001), Real-time multi-spectral image fusion, *Concurency and Computing: Practice and Experience*, 13(12): 1063-1081

Bowles, J. H., Antoniades, J. A., Baumback, M. M., Grossmann, J. M., Haas, D., Palmadesso, P. J., and J., Stracka (1997), Real-time analysis of hyperspectral data sets using NRL's ORASIS algorithm, *Proceedings SPIE Imaging Spectrometry III*, 3118():38-45

Bowles, J., Chen, W., and D., Gillis (2003), "ORASIS framework – benefits to working within the linear mixing model", *Proceedings Igarss03*, ( ):96-98

Brilles, S. D. (1997), Real-time onboard hyperspectral-image compression system for a parallel push broom sensor, *Proceedings SPIE Algorithms for Multispectral and Hyperspectral Imagery III*, 3071( ): 182-190

Chai, S. M., Gentile, A., Lugo-Beauchamp, W. E., Fonseca, J., Cruz-Rivera, J. L., and D. S. Wills, (2000) Focal-Plane Processing Architectures for Real-Time Hyperspectral Image Processing, *Applied Optics IP*, 39(5): 835-849

Chang, C.-I , Ren, H., and S.-S. Chiang (2001), Real-Time Processing Algorithms for Target Detection and Classification in Hyperspectral Imagery, *IEEE Transactions on Geosciences and Remote Sensing*, 39(4): 760-768

Du, Q., and H., Ren (2003), Real-time constrained discriminant analysis to target detection and classification in hyperspectral imagery, *Pattern Recognition*, 36(1): 1-12

Du H., C.-I. Chang, H. Ren, C-C Chang, J. O. Jensen, and F. M. D'Amico (2004) "New hyperspectral discrimination measure for spectral characterization", *Optical Engineering*, 43(8): 1777–1786

Heinz, D., and C.-I., Chang (2001), Real-Time Processing of an Unsupervised Constrained Linear Spectral Unmixing Algorithm, *Proceedings Igarss01,* ( ): 372-374

Hydice (1995), *Hyperspectral Digital Imagery Collection Experiment Documentation*

Hyvärinen, A., J. Karhunen, and E. Oja (2001). *Independent Component Analysis*, John Wiley & Sons, New York

Lillesand, T. M. and R.W. Kiefer (2000). *Remote sensing and image interpretation*, John Wiley and Sons, New York

Rencher, A.C. (1995), *Methods of Multivariate Analysis*, John Wiley & Sons, New York

Richards, J. A. and X. Jia (1999). *Remote Sensing Digital Image Analysis*, Springer, New York.

Robila, S. A. (2004), "Distributed Source Separation Algorithms for Hyperspectral Image Processing", *SPIE Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery X*, 5093(): 628-635

Robila, S. A. (2005a), "Using Spectral Distances for Speedup in Hyperspectral Image Processing", *International Journal of Remote Sensing*, (in press).

Robila, S. A., and A. Gershman (2005b), "Spectral Matching Accuracy in Processing Hyperspectral Data", Proceedings*IEEE ISSCS,* ( ): 163-166.

Stellman, C. M., Hazel, G., Bucholtz, F., Michalowicz, J. V., Stocker, A. D., and W., William (2000), Real-time hyperspectral detection and cuing, *Optical Engineering*, 39(07): 1928-1935