FOI
Defence Research Agency
P.O. Box 1165
SE-581 11 LINKÖPING

Phone: +46 13 37 80 00
Fax: +46 13 37 81 00

www.foi.se

A Matlab Toolbox for Analysis of Multi/Hyperspectral Imagery

JÖRGEN AHLBERG

```
>> Algo = SEM('MaxIterations',10);
>> SpectralModel = gmm(nComponents);
>> SpatialModel = globalmodel('Subsampling',5);
>> TrainingData = hsi2vecs(SpatialModel,IM);
>> [Algo,SpectralModel] = run(Algo,SpectralModel,TrainingData);
>> Anomaly = hsianomaly(IM,SpectralModel,[]) > Thres;
>> y = X(:,:,[1 1 1]);
>> y = y / max(y(:));
>> y(1) = double(Anomaly);
>> imagesc(y)
>>
```

FOI-R--1962--SE Technical report
ISSN 1650-1942 March 2006

Sensor Technology

Jörgen Ahlberg

A Matlab Toolbox for Analysis of Multi/Hyperspectral Imagery

FOI-R--1962--SE Technical report **Sensor Technology**
ISSN-1650-1942 March 2006

Issuing organisation FOI – Swedish Defence Research Agency Sensor Technology P.O. Box 1165 SE-581 11 LINKÖPING	**Report number, ISRN** FOI-R--1962--SE	**Report type** Technical report
	Research area code C^4ISTAR	
	Month year March 2006	**Project no.** E3082
	Sub area code Above water Surveillance, Target Acquisition and Reconnaissance	
	Sub area code 2	
Author/s (editor/s) Jörgen Ahlberg	**Project manager** Jörgen Ahlberg	
	Approved by Mattias Severin Head, Dept. of IR Systems	
	Sponsoring agency Swedish Armed Forces	
	Scientifically and technically responsible Gustav Tolt	

Report title
A Matlab Toolbox for Analysis of Multi/Hyperspectral Imagery

Abstract

At the Department of IR Systems, Division of Sensor Technology, FOI, the ongoing research on analysis of multi- and hyperspectral imaging indirectly results in software tools. Some of these tools, developed in Matlab, are packed in a toolbox available internally at FOI. This report describes the toolbox and gives usage examples.

Keywords
hyperspectral, multispectral, Matlab

Further bibliographic information	**Language** English
ISSN ISSN-1650-1942	**Pages** 28 p.
	Price acc. to pricelist

Utgivare FOI – Totalförsvarets forskningsinstitut Sensorteknik Box 1165 581 11 LINKÖPING	**Rapportnummer, ISRN** FOI-R--1962--SE	**Klassificering** Teknisk rapport
	Forskningsområde Ledning, informationsteknik och sensorer	
	Månad år Mars 2006	**Projektnummer** E3082
	Delområde Spaningssensorer	
	Delområde 2	
Författare/redaktör Jörgen Ahlberg	**Projektledare** Jörgen Ahlberg	
	Godkänd av Mattias Severin Chef, Inst. för IR-system	
	Uppdragsgivare/kundbeteckning FM	
	Tekniskt och/eller vetenskapligt ansvarig Gustav Tolt	
Rapportens titel Mjukvara för analys av multi/hyperspektrala bilder		

Sammanfattning

På institutionen för IR-system, FOI Sensorteknik, är ett indirekt resultat av forskningen om analys av multi- och hyperspektrala bilddata att en stor mängd mjukvaruverktyg tagits fram. Ett antal av dessa verktyg, utvecklade i Matlab, har här satts samman till en ”toolbox” som är tillgänglig internt FOI. Denna rapport beskriver mjukvaruverktygen och ger några exempel på användning.

Nyckelord hyperspektral, multispektral, Matlab	
Övriga bibliografiska uppgifter	**Språk** Engelska
ISSN ISSN-1650-1942	**Antal sidor:** 28 s.
Distribution **enligt missiv**	**Pris:** **Enligt prislista**

Contents

1 Introduction

1.1 About the toolbox

At the Department of IR Systems, Division of Sensor Technology, FOI, the ongoing research on analysis of multi- and hyperspectral imaging indirectly results in software tools. Some of these tools, developed in Matlab, are packed in a toolbox, the Hyperspectral Imaging Toolbox (HSI Toolbox), available internally at FOI.

1.1.1 Requirements

The toolbox requires Matlab version 7.1. Additionally, the graphical user interface requires the Image Processing Toolbox.

1.1.2 Projects

The toolbox is developed during the projects ”Optronic Sensors” and ”Multispectral IR- & EO-sensors” during 2004 and 2005, respectively. The toolbox will continuously be further developed in the scope of the project ”Multi- and Hyperspectral Reconnaissance” during 2006–2008, and the course of development will be according to the needs within that project. All these projects are sponsored by the Swedish Armed Forces Research & Development Programme, and are parts of the European projects ”Spectral Imaging Techniques” (through WEAO/CEPA) and later ”Hipod” (through EDA).

1.2 About the report

1.2.1 Purpose

This report describes the toolbox and gives usage examples. The purpose is to provide a guide complemented by the User's Guide and the Reference Manual available both in the Matlab Help (after installing the toolbox) and on the HSI Toolbox webpages on the FOI Intranet [1].

1.2.2 Prerequisites

The reader is assumed to have knowledge of Matlab programming and basic linear algebra. The basic concepts of object oriented programming will also help.

This documents corresponds to version 0.6.13 of the hsi toolbox.

1.2.3 Outline

Chapter 2 gives a brief overview of the toolbox. Chapter 3 describes the basics, i.e., data formats and how to handle hyperspectral images. Chapter 4 treats various spectral models that are exploited for detection in Chapter 5. Chapter 6 briefly mentions some other tools available in the toolbox, and Chapter 7 describes the graphical user interface. Chapter 8, finally, describes an example application developed using the HSI Toolbox.

1.3 Hyper- and multispectral imaging

Multi- and hyperspectral electro-optical sensors are sensors (cameras) that sample the incoming light at several (multispectral sensors) or many (hyperspectral sensors) different wavelength bands. Compared to a consumer camera that, typically, uses three wavelength bands, corresponding to the red, green and blue colours, hyperspectral sensors sample the scene in a large number of wavelength (or spectral) bands, often several hundred. Moreover, these spectral bands can lie beyond the visible range, i.e., in the infrared domain. Each pixel thus forms a (spectral) vector of measurements in the different bands. This vector, the observed *spectral signature*, contains information on the material(s) present in the scene, and can be exploited for detection, classification, and recognition.

If an observed target spectrum deviates from observed background spectra, this deviation can serve as a measure of anomaly. An *anomaly detector* is thus a detector that detects pixels that "stick out" from the background, without any *a priori* knowledge about target or the background.

The theory for multi- and hyperspectral target and anomaly detection and the detection methods implemented in the toolbox are treated in [3].

1.3.1 Hyperspectral sensors

The sensor (and the scene) can be characterized with respect to spatial, spectral, radiometric and temporal resolution (and properties).

The spatial resolution and the distance from the sensor to the target determines whether a target can be *spatially resolved* or not. A spatially resolved target covers at least one pixel completely, which means that the target pixel(s) will be *pure*, in contrast the *mixed* pixel of a *sub-pixel target.* Generally, subpixel targets are very difficult to detect and must deviate substantially from the surroundings in order to be distinguishable. Spatial resolution will therefore be an important performance parameter.

The spectral resolution determines the number of spectral bands, while the radiometric resolution determines the number of bits per sample and is limited by the signal-to-noise ratio. The temporal resolution determines how often a new pixel can be produced by the sensor.

In practical sensor design, trade-offs have to be made between spatial, spectral and temporal resolution. An important issue is therefore to try to establish optimal trade-offs with respect to scenarios and applications. In the final end, tactical sensors must be both inexpensive and perform well with respect to spatial, spectral and temporal information.

2 Overview of the Toolbox

2.1 Components

The toolbox essentially consists of a set of classes each encapsulating certain functionalities. Additionally, there is a set of tools for common tasks, for example, principal component analysis, viewing hyperspectral image data, and file utilities for accessing imagery in the ENVI file formats.

For easy reference, the classes or functions discussed in the text are noted in the margin like this.

The most basic functionalities include handling hyperspectral images (which tend to occupy large amounts of space on disk or in memory), regions of interest (typically, target masks), and receiver operating characteristics for assessing the performance of detection algorithms.

On top of that, functionality for spectral models are available. In the literature, it is very common to model the spectral variation of a specific target or background type as a linear subspace, a Gaussian distribution, a certain direction in spectral vector space, or a Gaussian mixture model. Thus, classes for each of these models have been implemented.

The different models often form the basis of target or anomaly detection schemes, which can be exploited by the anomaly detection tool or the different target detectors. The most common target detectors (ACE, ASD, AMF) from the literature are implemented as detector classes. Anomaly detection is executed by creating the suitable spectral model for the background, applying a spatial model (local or global), and measure the fit of the model to each investigated pixel.

For complex backgrund models, typically Gaussian mixture models or cluster models, a set of training algorithms based on K-means or Expectation-Maximization are available.

2.2 Future development

The future development of the toolbox is determined by the demands in currently ongoing and future projects. During 2006, methods for analysis of hyperspectral longwave (thermal) infrared imagery will be developed, including atmospheric modelling and correction, and temperature-emissivity separation (TES), as required by the EDA project HIPOD.

3 Basic Functionality

3.1 Hyperspectral data

In this toolbox, we distinguish between *hyperspectral image format* (hsi format) and *vector format*. Hyperspectral data stored in a 3D matrix or in an `hsimage` object are typically in hsi format, which means that the first dimension is the lines (rows) of the image, the second dimensions is the samples (columns), and the third is the spectral bands. This is the same as for RGB images loaded using `imread`.

Thus, if `X` containts hsi data, `X(1,2,3)` is the third spectral band in the second sample in the first line.

Data in vector format is one or more (spectral) vectors in a 2D matrix. Each column is a vector, and thus the first dimension is the spectral bands and the second is the vector number. Commonly, one line of spectral data is extracted from hsi data and transformed to vector format in order to facilitate matrix operations.

If you have a function processing one spectral vector or a line of spectral data, you do not even need to write the loop yourself. The function `hsiapply` applies a function to all data in a hsi data cube and returns the result.

3.2 Hyperspectral images

When dealing with hyperspectral imagery, a common problem is the size of the data. The `hsimage` class provides a solution to the problem by letting the user access the data as if it were a 3D matrix loaded into the memory. An `hsimage` object is always connected to a file and keeps a part of the data in memory. When the user accesses different parts of the data, the correct parts of the file are swapped into memory.

`hsimage`

A function looping over a hyperspectral data cube does thus not need to know if the data is stored in a 3D matrix or in an `hsimage` object. The `hsimage` behaves like a 3D matrix in most, but not all, ways. The main difference is that you cannot apply global operators, like `+`, `-`, `*`, `/`, or `'`.

The files can be in ENVI or Matlab format. In both cases, they should be accompanied by an ENVI header file describing the file format.

Assume in the following that we have a hyperspectral image stored in the ENVI file `image.img` with the corresponding header file `image.hdr`. To create an `hsimage` object from the file, write:

```
X = hsimage('image.img');
```

The variable `X` will act as a 3D matrix with the three dimensions corresponding to the lines (rows), samples (columns), and spectral bands respectively (hsi format, see above). Moreover, if you access a single pixel, e.g., `X(3,4)`, a spectral column vector will be returned.

Accessing a part of the image that is not loaded into memory will cause the `hsimage` to load the requested part. If the currently loaded image data has been modified, it will automatically be written to file. However, to avoid accidental file modification, `hsimage` objects are by default created in read-only mode. You can change this using the `FileMode` option:

```
X = hsimage('image.img','FileMode','rw');
```

To get information about the image, simply type `X`. This will tell you the dimensions of the image, the associated file name, and how much of the image that is loaded into memory, for example like this:

```
X = [hsimage]
Properties:
  FileName:      C:/hsi/hstest.img
  nBands:        15
  nSamples:      800
  nLines:        200
  LinesInMemory: 100 (46 through 145)
  BytesInMemory: 9 MB (50% of image)
  FileMode:      Read/write
  DebugMode:     1
  Viewdata:      100 x 400, bands [7 6 4]
  FileFormat:    envi
```

3.2.1 Displaying hsimages

`view`

To display an overview of the `hsimage` object `X`, use `view(X)`. One or three bands (given by the property `DefaultBands`) are loaded and subsampled to a convenient size and displayed in an image window. A red rectangle marks the part of the image that is loaded in memory, and by clicking in the window you can load another part of the image. To view details of the image, use `view(X,2)`, which displays the overview and calls `hsiview` (see below) on the part of the image data currently loaded in memory. If another part of data is loaded, both windows are updated automatically.

`imagesc`

To display the entire image (entire spectral band(s)) in detail, use

```
imagesc(X,BAND)}
```

To display three bands simultanesously, coded as red, green, and blue respectively, use:

```
imagesc(X,[REDBAND, GREENBAND, BLUEBAND])
```

`hsiview`

To view and examine hyperspectral image data (a 3D matrix or an `hsimage` object), use the function `hsiview` which displays the image in the GUI descrbied in Chapter 7.

3.2.2 Creating new hyperspectral image files

`enviCreateIMG`
`hsiCreateImage`

To create a new file, use either `enviCreateIMG` to create an ENVI image file or `hsiCreateImage` to create Matlab-files:

```
hsiCreateImage('newimage','Size',[100 100 10] )
```

or

```
enviCreateIMG('newimage.img','Size',[100 100 10] )
```

3.2.3 How much memory does the image use?

You can chose how much memory that should be available for the image on creation, by using one of the options `MaxBytesInMemory` or `MaxLinesInMemory`:

```
X = hsimage('image','MaxBytesInMemory',1e8);
X = hsimage('image','MaxLinesInMemory',100);
```

The first example tells the image to keep approximately 100 MB of image data in memory. The second example tells the image to keep 100 lines of image data in memory. If you are going to access 50 lines of the image at a time (for example, when examining 50×50 pixel local neighbourhoods), the image should be able to keep at least 50 lines in memory, and preferrably more to avoid uneccessary swapping.

3.2.4 Closing and saving

When you do not want to use the image anymore, close it and free the memory using `close(X)`. Note that clearing the variable `X` the ordinary way (`clear X`) will not free the memory if `X` is not closed first! This is in analogy with file handles — clearing a file handle will not close the file. `close`

To close all hsimage objects, call `closeall(X)`, where `X` is any hsimage object. `closeall`

3.3 Files and file formats

Two different file formats are currently supported. The first is the ENVI file format, as defined in [2]. It is a very flexible file format, where each image consists of two files, one data file and one header file. The header is an ASCII file describing the format of the data file, and thus many file formats are supported by describing them with a correct header file.

Header files can be read and written using the class `enviheader` . `enviheader`

Additionally, hyperspectral images can be stored as Matlab files. ENVI file headers are still used, but instead of a data file there is a directory containing one or more Matlab files. Each Matlab file contains a number of lines of data.

4 Modelling

4.1 Background

Assume that we have a source (e.g., an electro-optical sensor) outputting a sequence of samples (measurements). Having no knowledge of the inner workings of the source, we regard the samples as realisations of a random variable and use the samples to build a model of the source. The model also gives us a measure telling us how well each new sample is described by the model (or, the other way around, a distance from the new sample to the model). Note that when the sensor is multi- or hyperspectral, the samples are multidimensional, i.e., vector-valued and not scalar-valued.

A simple model would be to calculate the mean of the received samples so far, and for each new sample, the deviation from the mean is computed. A large deviation is to be regarded as an *anomaly* and the scheme is thus a simple *anomaly detector*. Below, we will refer to the samples used for calculating the model parameters as the *training samples* and the new samples as the *test sample(s)*.

Naturally, we might have some knowledge about the source that we can exploit when selecting and training the model. For example, we might assume that all samples are linear mixtures of a small set of pure signautres, so called *end-members*.

Models are not necessarily trained from the actual sensor data, but might also originate from simulations and/or libraries with spectral signatures. If we, for example, are looking for certain materials in the scene and know their spectral signatures, we can compare the test samples to those signatures and report similar samples as detections. We call this *signature-based target detection* or just *target detection*, as will be discussed in the next chapter.

A related term is *clustering* (or *unsupervised classification*), which is the process of separating a set of vectors into different clusters or classes, as is discussed in Section 4.3.

4.2 Spectral models and their training

Six different spectral models are available in the toolbox. They are implemented as classes in the *toolboxpath*`/hsi/models` directory. The models are typically created from a set of training vectors, e.g., `M = spectralmodel(X)` where `X` is an $m \times n$ matrix containing n spectral vectors of dimensionality m.

- The basic spectral model (class `spectralmodel`) is the superclass to all other spectral models. It needs only one spectral vector to be created, and this vector is stored as the mean or prototype for the model. The distance measure from a test vector to the model is the squared Euclidean distance. The `spectralmodel` class includes a number of basic functions that are thus inherited (or overridden) by all other model classes: `spectralmodel`

`adapt`	Adapts an existing model to new data.
`dim`	Returns the dimensionality of the model.
`distance`	Computes the distance from one or more spectral vectors to the model.
`mean`	Returns the mean or prototype vector.
`plot`	Plots the model.
`reset`	Similar to the constructor.
`update`	Updates the model by adding more data to the training set.

`gaussianmodel`

- The Gaussian model (class `gaussianmodel`) models spectral vectors as a Gaussian distribution in spectral space, i.e., as a mean vector and a covariance matrix. The associated distance measure is the Mahalanobis distance, i.e.,

$$d_{\mathsf{M}}(\mathbf{x}, \mathcal{C}) \triangleq (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Gamma}^{-1} (\mathbf{x} - \boldsymbol{\mu}), \tag{4.1}$$

  where $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{\Gamma}$ is the covariance matrix, and $\mathbf{x}$ is the test vector.

  In order to create a Gaussian model, you can give a training set of spectral vectors, a Gaussian distribution (class `gaussian`) or a mean and covariance. The `gaussianmodel` also supports sequential updating, i.e., training vectors can be added and the model updated.

`subspacemodel`

- The subspace model (class `subspacemodel`) creates a linear subspace with a given maximum dimensionality `m`. If the training data does not fit into an `m`-dimensional space, the `m` principal dimensions are used. The associated distance measure is the squared Euclidean distance between the test vector and and its projection onto the subspace.

`spectralanglemodel`

- The spectral angle model (SAM) (class `spectralanglemodel`) is essentially the same as the `subspacemodel`, but the distance measure is the angle between the test vector and the subspace.

`nearestneighbourmodel`

- The nearest neighbour model (class `nearestneighbourmodel`) is somewhat special in that the model stores all given training data. When the distance measure is computed, the Euclidean distance is computed from each of the training vectors to the test vector, and the minimum is returned. This model is thus quite unsuitable for global spatial models (see Section 4.4).

`mixturemodel`

- The mixture model (class `mixturemodel`) is used for cluster models and mixture models. A *cluster model* is a composite model where each component is a spectral model, for example a Gaussian model. A cluster model is typically used for classification, where each cluster (component) represents one class. A *mixture model* on the other hand represents *one* class, but can represent a much more complicated structure than, for example, a Gaussian distribution or a linear subspace. The most commonly used mixture model is the Gaussian mixture model.

  A `mixturemodel` is created by by stating the type and number of components, for example:

```
    M = mixturemodel(gaussianmodel,4);
```

`scm`
`gcm`
`gmm`

  In order to provide a simple interface for creating cluster and mixture models, there are three functions `scm` , `gcm` , and `gmm` for creating *spherical cluster models*, *Gaussian cluster models*, and *Gaussian mixture models* respectively. For example, `M = gcm(5);` creates a Gaussian cluster model with five components/clusters.

4.2.1 Training mixture and cluster models

The *Linde-Buzo-Gray* (LBG) algorithm is a *K-means* method for computing a set of cluster prototypes. Its objective is to minimize LBG

$$\sum_{i=1}^{N} \sum_{\mathbf{x}\in\mathcal{C}_i} ||\mathbf{x} - \boldsymbol{\mu}||^2, \tag{4.2}$$

where N is the number of classes, by assigning class labels to all training samples and assigning a prototype vector ($\boldsymbol{\mu}_i$) to each class $\mathcal{C}_i$. This is done by iteratively computing the mean vectors

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{x}\in\mathcal{C}_i} \mathbf{x} \tag{4.3}$$

and then assign each training vector to the class with the nearest prototype vector, i.e., let $\mathbf{x} \in \mathcal{C}_i$ when $i = \arg\min ||\mathbf{x} - \boldsymbol{\mu}_i||^2$.

Classification Expectation-Maximization (CEM) is very similar to LBG/K-means, but instead of using the squared Euclidean distance the Mahalanobis distance is used. Each class thus has a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Gamma}$. CEM can be regarded as an extension of LBG or as a simplification of EM (see below). CEM

Stochastic Expectation-Maximization (SEM) is similar to CEM with two exceptions. First, each class is assigned a weight, and, second, the class labels are in each step are random. SEM is typically used for training a mixture model, i.e., a probability distribution function SEM

$$f(\mathbf{x}) = \sum_{n=1}^{N} w_n f(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Gamma}_i). \tag{4.4}$$

Expectation-Maximization (EM) is used for training a mixture model rather than a cluster model. It does not use discrete class labels, instead each training sample is assigned a probability for each component of the mixture model. EM

The classes above are subclasses to `algorithm` and are thus invoked in the same way, for example: `algorithm`

```
% Create an algorithm instance.
A = SEM('MaxIterations',10,'ConvergeAt',0.01,'Plot',3);

% Select model to be trained by the algorithm.
M = gmm(5);

% Run the algorithm on the data X.
A,M = run(A,M,X);
```

4.3 Segmenting a hyperspectral image

In order to segment a hyperspectral image, i.e., perform spectral clustering, we can use a spherical or a Gaussian cluster model, and train it using LBG or CEM (respectively). Often it is not necessary to use all the pixels of the image for training, but a small subset is quite enough (a few hundreds or thousands pixels chosen at random).

Then, we use the function `hsiclassify` to classify each pixel. In the example below, the image data in `X` is segmented into `N` clusters. `hsiclassify`

```
% Create an algorithm instance.
A = CEM('MaxIterations',100);

% Select 1\% of the image pixels for training the model.
G = globalmodel('Subsampling',10);
G = update(G,X);

% Train the cluster model on the data X.
[A,M] = run(A,gcm(N),getvecs(G));

% Classify all pixels in X and display result.
L = hsiclassify(X,M);
imagesc(L);
```

4.4 Spatial Modelling

The spectral models discussed above typically require spatial models for definition of training data. Here, we disregard spatial *patterns*, and thus the spatial models basically only tell us where to collect data to train our spectral model(s). To measure a distance from a test pixel signature to, for example, the background model, we need to define the spatial area that represents the background, i.e., what pixel signatures to chose as training vectors for the model. We define the following areas (illustrated in Figure 4.1):

- The *center pixel* is the pixel we are currently examining.
- The *global background* is the entire available image.
- The *local background* contains all pixels within a distance of n_L pixels from the center pixel. Typically, but not necessarily, the neighbourhood is square. Pixels within a distance n_G pixels, the *guard distance*, from the center pixel might be excluded from the local background.
- The *target area* contains the pixels within a certain distance from the center pixel. Each pixel is weighted as to reflect the likelihood of the target stretching to the pixel. Commonly, a Gaussian distribution is used. The resulting value should be normalized so that it sums to 1 over all target area pixels. In the current implementation, the target area is the center pixel only.

From the global and/or local background, we can build the background model $\mathcal{B}$ or even several background models if we perform a clustering of the background. The target signature $\mathbf{x}$ is estimated as the weighted average of the target area pixels. Given a target probe $\mathcal{T}$, that can be a single signature vector or a representation of a class, we then measure $d(\mathbf{x}, \mathcal{B})$ and $d(\mathbf{x}, \mathcal{T})$, as mentioned above.

A global model is useful when statistics on the entire scene is necessary (for example, end-member extraction) or when the model is advanced enough to handle a complex scene (for example, a Gaussian mixture model). Global models also have the advantage that they are not re-trained for each pixel.

Yet another spatial model is to segment the image and use different background models for different parts of the image. This corresponds to the class conditional RX detector described in Section 5.1.

globalmodel

In the hsi toolbox, three classes are used for spatial modelling. The class `globalmodel` provides data from the entire image, but it can also be subsampled or be limited to certain lines/bands/samples.

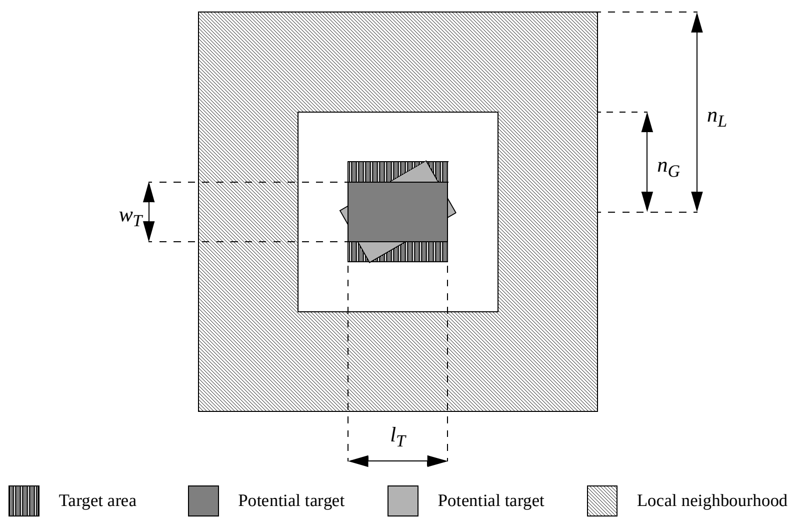

Figure 4.1: Target area, guard distance, and local neighbourhood.

k

However, in most cases, the class `hsblock` is preferrable, and `globalmodel` will probably disappear from future versions of the toolbox. A `hsblock` is defined in relation to an `hsimage` and acts like one, representing a selected subset of the `hsimage` data. For example, assume that `X` is an `hsimage`. Then,

```
B = hsblock(X11:20,11:20,[1 2 7])
```

will create a `hsblock` of the size $10 \times 10 \times 3$, and accessing `B(1,2,3)` will be the same as accessing `X(11,12,7)`.

`localmodel`

The class `localmodel` is defined by its width and guard distance, and each time it is updated with respect to an image position, the local background data is copied to the `localmodel` object. For example:

```
% Create a local model with width 11 and guard distance 3.
M = localmodel(5,3);

% Extract the local background around the pixel at (12,34) in
% the hsimage X.
M = getvecs(M,X,12,34);
```

5 Detection

Target detection is, in this context, about finding pixels (samples, spectral vectors) in images that

- do *not* correspond to some model of the background spectral signature

and/or

- *do* correspond to a target model.

The case when a target model is available, we here refer to as *signature-based target detection*, while the process of detecting an unknown target is called *anomaly detection*. Target detection is discussed in Section 5.2 and anomaly detection in Section 5.1.

In our notation, the *detector* is a function

$$D\colon \mathcal{R}^N \to \{\text{true,false}\}, \tag{5.1}$$

telling if a (spectral) test vector is a target or not. However, the detectors implemented in this toolbox is a function

$$d\colon \mathcal{R}^N \to \mathcal{R}, \tag{5.2}$$

and the user has to select a threshold so that $D = d > t$.

5.1 Anomaly detection

Anomaly detection is the case when we do not know the spectral signature of the target, and we try to find pixels that deviate from the background. We use a background model $\mathcal{B}$, a distance measure $d(\cdot)$, and a threshold t. We regard a pixel $\mathbf{x}$ as an anomaly if $d(\mathbf{x}, \mathcal{B}) > t$, and the detector is thus given by

$$D(\mathbf{x}|\mathcal{B}) = [\, d(\mathbf{x}, \mathcal{B}) > t\,]\,. \tag{5.3}$$

To exemplify, assume that we record the mean vector of a set of training samples. The model consists of the mean vector $\boldsymbol{\mu}$, and the distance measure is the Euclidean distance, i.e.,

$$D(\mathbf{x}|\mathcal{B}) = [\, \|\mathbf{x} - \boldsymbol{\mu}\| > t\,]\,. \tag{5.4}$$

Thus, a model for the background signature is needed, as well as a spatial model, i.e., from where to choose the spectral vectors to train the model. For example, we could use a local model (estimating the background signature from a local neighbourhood only) or a global model (using all available image data). Spatial models are discussed in 4.4.

Then, in order to measure the distance from each pixel signature to the background model, we need a distance measure. The choice of distance measure is restricted, or even determined, by the model used for the background and thus the assumptions about background spectral distribution.

Finally, we need to set the threshold t. A high threshold will give few detections, reducing the *detection rate* (DER), but also the *false-alarm rate* (FAR).

5.1.1 Running anomaly detection

To run an anomaly detection algorithm, use `hsianomaly`, for example

```
Y = hsianomaly( X, gaussianmodel(m,C), [] );
```

The output is an image `Y` where each pixel is an anomaly score. If a target mask is available, for example on the file `roi.txt` the ROC curve can be computed (and plotted):

```
Yt = extract( ROI('roi.txt'), Y );\\
ROC( Y, Yt );
```

5.1.2 Some common anomaly detectors

- *Distance from feature space* (DFFS): The background is modelled as a linear N-dimensional subspace.

  ```
  Y = hsianomaly( X, subspacemodel(N), globalmodel )
  ```

- *Global RX*: The background is modelled as a Gaussian distribution common for the entire image.

  ```
  Y = hsianomaly( X, gaussianmodel, globalmodel )
  ```

- *Local RX*: The background is modelled as a Gaussian distribution adapted to the local neighbourhood of each pixel.

  ```
  Y = hsianomaly( X, gaussianmodel, localmodel(10,3) )
  ```

- *Gassian cluster RX* (GCRX) or *class conditional RX*: The image is clustered spectrally into `N` clusters and the (Mahalanobis) distance to the nearest cluster is used as the anomaly score.

  ```
  % Select training algorithm.
  A = CEM('MaxIterations',10);

  % Select spatial model, i.e., how to pick training
  % vectors.
  G = globalmodel('Subsampling',10);
  G = update(G,X);
  % or
  G = hsblock(X,1:10:size(X,1),1:10:size(X,2));

  % Train the background model.
  [A,M] = run(A,gcm(N),getvecs(G));

  % Run the anomaly detection.
  Y = hsianomaly(X,M)
  ```

- *Gassian mixture model* (GMM): A spectral distribution is estimated for the image, and the negative log-likelihood for each pixel is used as anomlay score.

  ```
  A = SEM('MaxIterations',10);
  G = hsblock(X,1:10:size(X,1),1:10:size(X,2));
  [A,M] = run(A,gmm(N),getvecs(G));
  Y = hsianomaly(X,M)
  ```

5.2 Signature-based target detection

A signature-based algorithm for target detection searches for pixels that are similar to a *target probe*. The target probe is a model of a certain target signature $\mathcal{T}$, i.e., the spectral signature of the target or target class is known. In contrast, the anomaly detection discussed above assumes no such knowledge. Basically, we measure the distance from a pixel signature to the target model. That is, we can classify pixel $\mathbf{x}$ as a target pixel if $d(\mathbf{x}, \mathcal{T}) < t$ and the corresponding detector is thus

$$D(\mathbf{x}|\mathcal{T}) = [\, d(\mathbf{x}, \mathcal{T}) < t \,] \,. \tag{5.5}$$

Usually, we incorporate background suppression in our target detection scheme in order to enhance detection performance. There are basically three ways of doing this:

- **Separate thresholds**. First, run an anomaly detector

  $$D_A(\mathbf{x}|\mathcal{B}) = [\, d(\mathbf{x}, \mathcal{B}) > t_A \,] \,. \tag{5.6}$$

  All pixels marked as anomalies are then investigated by the target detector

  $$D_T(\mathbf{x}|\mathcal{T}) = [\, d(\mathbf{x}, \mathcal{T}) < t_T \,] \,. \tag{5.7}$$

  The advantage is that several different target detectors can be applied to only a small amount of the test samples.

- **Direct comparison**. Run the anomaly detector and the target detector on all test samples and use the compare the results:

  $$D(\mathbf{x}|\mathcal{B}, \mathcal{T}) = \left[\frac{d(\mathbf{x}, \mathcal{T})}{d(\mathbf{x}, \mathcal{B})} > t \right] . \tag{5.8}$$

- **Combined detector**. For certain models, a combined detector $D(\mathbf{x}|\mathcal{B}, \mathcal{T})$ can be derived. That is, instead of measuring a distance to the target and a distance to the background, a joint measure is derived.

5.2.1 Detectors

Depending on what knowledge we have about targets and backgrounds, we can use different models, and thus different detectors.

- The simplest detector is the *Adaptive Matched Filter* (AMF) implemented through the class `AMF`. The target model is an example signature, and the background model is the Gaussian distribution. The AMF detector is created from examples of target and background signatures. **`AMF`**

```
D = AMF('Targets',T,'Backgrounds',B);
```

- Modelling the targets as well as the background as Gaussian distributons, the Gaussian cluster model is used, and `hsiclassify` used to discriminate between target and background pixels.

- Modelling the background as a Gaussian distribution and targets as a linear subpace, we get the *Adaptive Coherence/Cosine Detector* (ACE) implemented in the class `ACE`. The ACE detector is created in the same way as the AMF detector. **`ACE`**

- Modelling the background as a linear subspace, we can model the target(s) as a single example signature using *Orthogonal Subspace Projection* (OSP). This is a special case of the ASD detector.

ASD

- Extending OSP, we can model both targets and backgrounds as linear subspaces using *Adaptive Subspace Detection* (ASD) implemented by the class `ASD`. It is created by examples and a maximum dimensionality of the target subspace. Setting the maximum dimensionality to 1, we get the OSP detector.

```
D = ASD('Targets',T,'Backgrounds',B,'MaxDim',N);
```

detector

All detectors above are subclasses to `detector` and used according to a common interface. They can be run in global or local mode, where the global mode requires that background examples are given when creating the detector. The detector is then simpy applied to the image data as:

```
Y = apply(D,X);
```

In local mode, a local model needs to be given (see Section 4.4). The background model is then fitted to the local neighbourhood of each examined pixel, for example:

```
Y = apply(D,X,'Local',localmodel(10,3));
```

5.3 Performance measures

By changing the threshold t above, the *detection rate* (DER) and the *false alarm rate* (FAR) can be varied. FAR and DER correspond to the probabilites of detection ($P_{\mathsf{D}}(t)$) and false alarm ($P_{\mathsf{FA}}(t)$), respectively.

Unfortunately, both are increased (decreased) simultaneously whereas the wish would be to increase the detection rate and still keep the false alarm rate low. Thus, FAR and DER must be related to be meaningful—it is easy to create a detector with 100% detection rate if no requirement is set on the false alarm rate.

There are a few common ways of presenting the performance of a detector:

- The *Receiver Operating Characteristics* (ROC) is a graph with FAR and DER on the axes, and a curve showing DER as a function of FAR (found by varying the threshold), i.e., a parametric curve

$$\mathbf{x}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \mathrm{FAR}(t) \\ \mathrm{DER}(t) \end{pmatrix}, \tag{5.9}$$

  where t is varied so that FAR and DER goes from zero to one.

- The *FAR at first detection* (FFD) is the false alarm rate when the first pixel of a certain target is detected, giving an indication of the minimum achievable FAR for that type of target, detector, and so on.

- The *Area Under Curve* (AUC) is the integral of the ROC, giving one scalar value describing the performance of the detector. Since the performance of the detector at high FAR is less interesting, the integral is sometimes computed over an interval FAR $= [0, I]$, for example $\mathrm{AUC}_{0.01}$ is defined by $\int_{t=t_0}^{t_{0.01}} \mathrm{DER}(t)dt$, where the limits t_c are defined sp that $\mathrm{FAR}(t_c) = c$.

5.3.1 The ROC class

Plotting of ROC curves is implemented through the class `ROC`. By creating an instance, either a graph is plotted or an `ROC` object is created for later plotting. For example, assume that `X` is a multispectral image, that `R` is an `ROI` object (see Section 6.2) describing the known positions of targets, and that `D` is a detector. The following code will run the detector and extract the target detection values:

```
Y = apply(D,X);
T = extract(R,Y);
```

To plot an ROC curve, type `ROC(Y,T)`. To create an `ROC` object, type `P = ROC(Y,T)`, and plot it using the `plot` function. Examples:

```
% Plot an ROC curve.
ROC(Y,T,'YScale','linear');

% Create an ROC object.
P = ROC(Y,T,'YScale','linear');

% Plot it in two different ways.
plot(P,'XScale','linear','Legend',R.Names)
plot(P,'YScale','log','XScale','log')
```

To compute the AUC in an interval $[0, I]$, use `A = AUC(P,I);`.

6 Miscellaneous Tools

A number of tools for processing hyperspectral images are available in the toolbox. The function `hsiintegratebands` is useful for reducing the number of bands by merging (summing) bands. `hsifindbands` is used to find bands within certain wavelength ranges. To search for end-members in an image, the N-FINDR algorithm is implemented in the function `hsinfindr` .

`hsiintegratebands` `hsifindbands` `hsinfindr`

6.1 Principal component analysis and transform

Principal component analysis (PCA) is a popular tool in remote sensing. It is implemented in the three functions `hsipca` , `hsipct` , and `hsiinvpct` . `hsipca` analyses an image, i.e., finds the principal components and the associated variances. `hsipct` also performs the analysis, but moreover, it transforms the image by projecting it on all or some of the principal components. `hsiinvpct` reconstructs the original image (or an approximation thereof).

`hsipca` `hsipct` `hsiinvpct`

6.2 Regions of interest

When assessing the performance of detection algorithms, some kind of ground truth is needed, for example a target mask pointing out the pixels belonging to the target(s). Such masks are here implemented by the `ROI` class. An `ROI` object stores information about one or more region of interest, where each region typically represent a target. The information stored is pixel coordinates, RGB color (for visualization), and name. An `ROI` object can be used to mask an images and to extract specific signatures, and it can also be manipulated by cropping and selection. The `ROI` class uses an ASCII file format identical to ENVI's export format.

`ROI`

Some examples of usage:

```
% Read a ROI from file.
R = ROI('roifile.txt');

% Extract all target signatures from an image.
T = extract(R,X);

% Extract a specific target.
T = extract(select(R,'Tank decoy'),X);

% Create a target mask
M = mask(R,zeros(n,m),1);
```

7 Graphical User Interface

The toolbox includes a simple graphical user interface (GUI) to facilitate examination of hyperspectral data.

To see an overview of an `hsimage` object `X`, type `view(X)`. This will display a grayscale or colour image (according to the `hsimage`'s `DefaultBands` property. If the image is large, it will be subsampled to a convenient size. A red rectangle marks the part of the image that is loaded into memory, and to load another part, the user can click on the part to be loaded. `view`

Typing `view(X,2)` also invokes `hsiview` on the part of `X` loaded into memory. `hsiview` can be called on any data in hyperspectral image format and displays the image data in a figure window, see Figure 7.1. The image can be zoomed in and out, and by selecting *Point Examination Mode*, spectral signatures in specific pixels can be examined. When a pixel is clicked, its signature is plotted in a separate window (Figure 7.3, left), and the variable `SpectralData` is assigned this signature in the main Matlab workspace. In *area examination mode*, see Figure 7.2, an entire area can be selected, plotted (Figure 7.3, right) and assigned to `SpectralData`). `hsiview`

By selecting the appropriate menu item, spectral models can be trained on the selected data and exported to the main workspace.

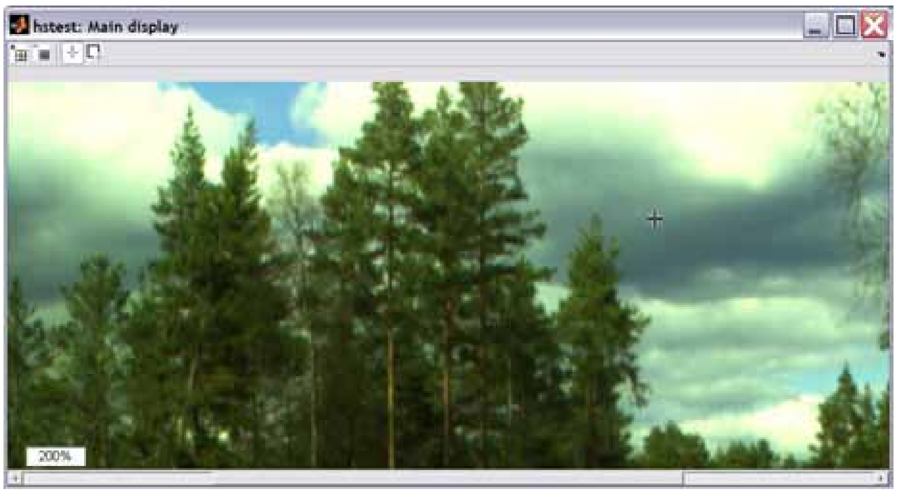

Figure 7.1: The main display of an hyperspectral image. Three bands corresponding to red, green, and blue are renderred. The buttons are *Zoom In*, *Zoom Out*, *Point Examination Mode*, and *Area Examination Mode*.

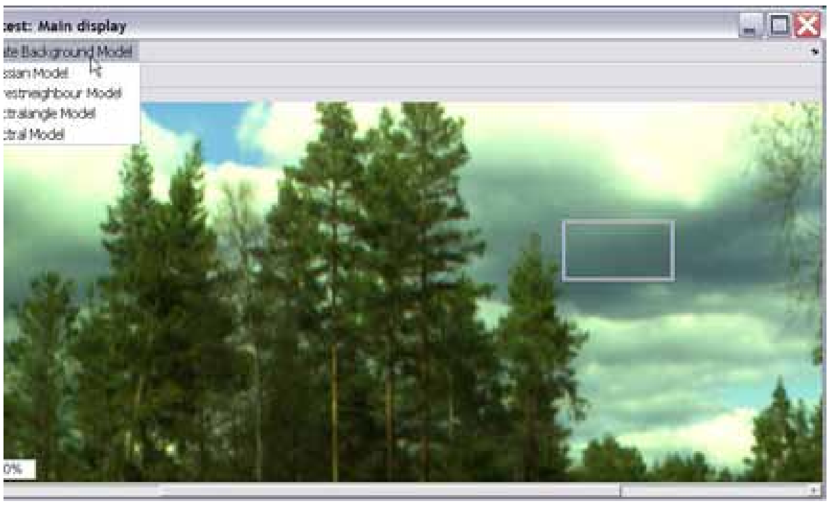

Figure 7.2: The main display of an hyperspectral image in Area Examination Mode. A spectral model can be created from the marked area.

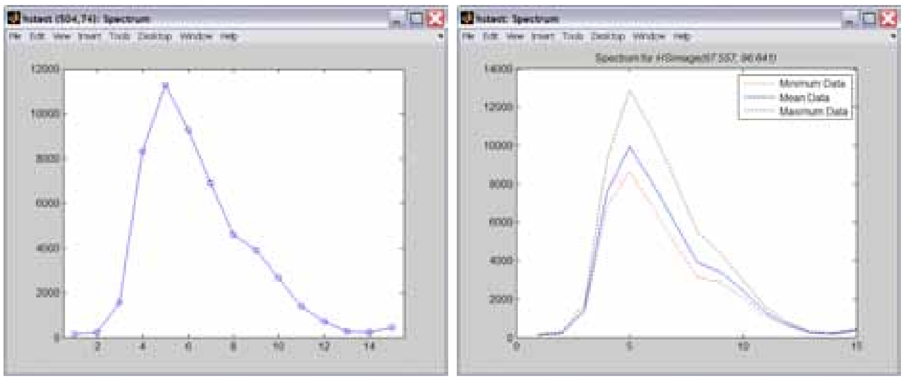

Figure 7.3: Left: The spectral signature of the pixel at the cross in the main display (Point Examination Mode). Right: The spectral signatures of the pixels within the marked area in the main display (Area Examination Mode).

8 Example Application

A demonstration application for target detection and operator support has been created using the HSI Toolbox. It is described briefly in [5], and will be more thoroughly treated in a report during spring 2006. The application reads (a sequence of) multispectral images and trains and updates background and target models on the incoming data. The user can chose to merge clusters representing different aspects of the same class (e.g., grass in shadow and grass in sunlight) and to name the classes.

The application is shown in Figure 8.1, with a segmented image (a simulated MultimIR [4] image). In Figure 8.2 (top) anomaly detection based on a Gaussian mixture model is illustrated. Note the two detected vehicles. In Figure 8.2 (bottom) the image is segmented using a Gaussian cluster model.

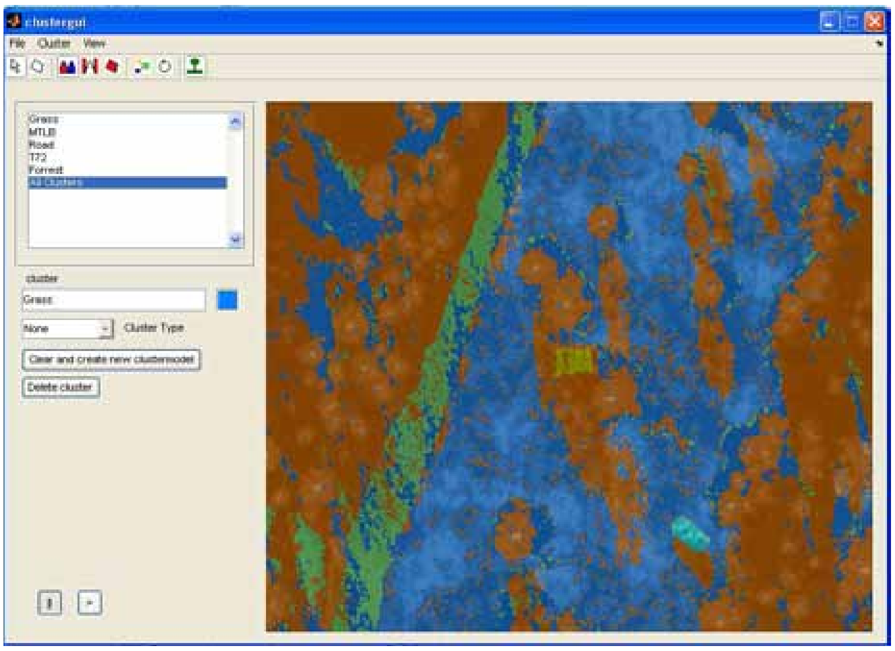

Figure 8.1: The GUI of the example application.

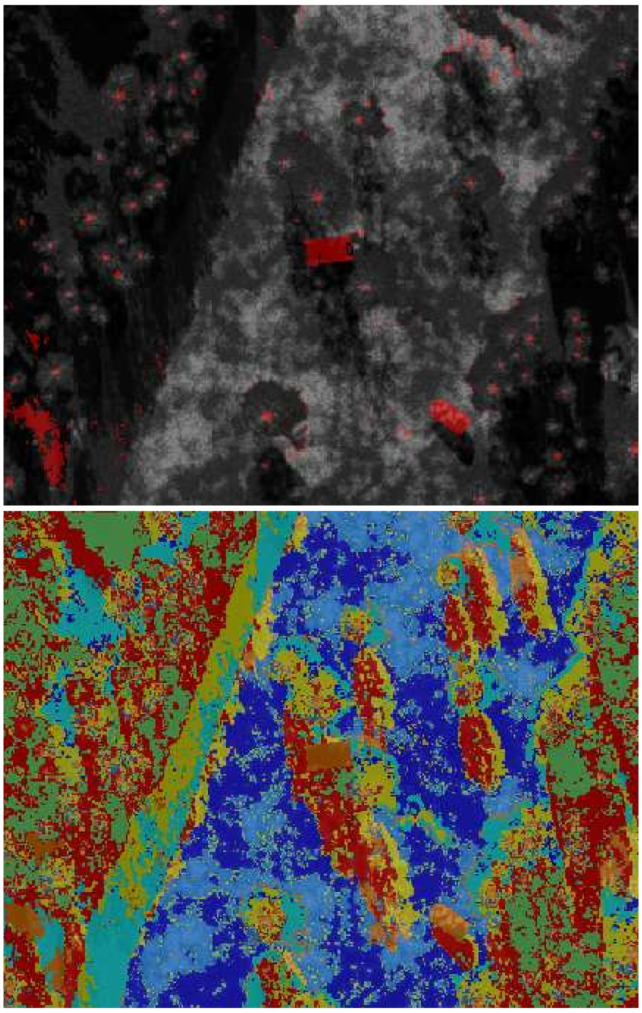

Figure 8.2: Top: Anomaly detection using a Gaussian mixture as background model. Bottom: Segmentation using a Gaussian cluster model.

Bibliography

[1] http://intranet.foi.se/templib/pages/NormalPage.aspx?id=31188

[2] RSI Inc., "Appendix B: ENVI File Formats," *ENVI User's Guide*, pp. 879–882.

[3] J. Ahlberg and I. Renhorn, *Multi- and Hyperspectral Target and Anomaly Detection*, Scientific report FOI-R--1526--SE, Swedish Defence Research Agency, 2004.

[4] J. Ahlberg et al., *Multispectral EO- & IR-sensors 2005*, User report FOI-R--1815--SE, Swedish Defence Research Agency, 2005.

[5] O. Brattberg and J. Ahlberg, "Analysis of Multispectral Reconnaissance Imagery for Target Detection and Operator Support," *Proc. Swedish Symposium on Image Analysis*, Umeå, Sweden, pp. 17–20, 2006. FOI-S--2168--SE