Tools to Check RasNiK and BCAM Image Quality in the LWDAQ Environment

Craig Dowell
University of Washington, Seattle

Version 0.0.1, 18 October 2009

Abstract

The ATLAS muon detector alignment system uses carefully arranged CCD camera devices to record images. These images are processed by software tools in the Long Wire Data Acquisition System (LWDAQ) to perform angular and translational measurements. The resulting information is used in the alignment system establish chamber position and distortion.

Since these measurements are critical to the performance of the detector as a whole, it is desirable to know that the images are of adequate quality. The Image Quality measurement tools address that issue by analyzing the "goodness" of each image and flagging images that do not meet specific criteria.

1. Introduction

The goal of the Image Quality tools is to identify bad images taken by muon detector alignment system CCD cameras. There are two types of Optical devices in the system: the BCAM (Boston CCD Angle Monitor) cameras; and the RasNiK (Red Alignment System of NIKHEF[1]) cameras. Detailed information regarding these systems can be found in the LWDAQ User manual, available at

http://alignment.hep.brandeis.edu/Electronics/LWDAQ/Manual.html

1.1 The BCAM

A BCAM is a CCD camera that looks at a laser light source that. A "pivot point" is defined at the camera lens on the camera axis. The angle between the camera axis at the pivot point and the light source is measured. The LWDAQ)Acquisifier Tool corresponding to the BCAM does this by finding the centroid of the brightest area on the image. In the case of the image shown in Figure 1, this approach works well.

[1] National Institute for Nuclear Physics and High Energy Physics in the Netherlands.

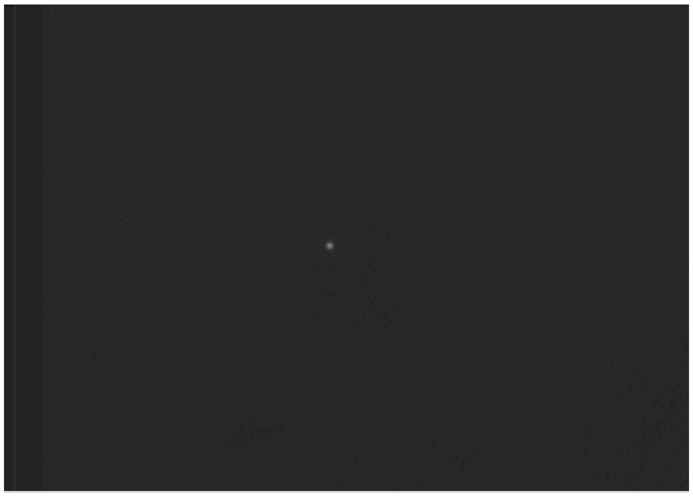


Figure 1: A Good Quality BCAM Image

In the presence of faults and errors, the analysis is no longer so straightforward, as Figure 2, below, might suggest.

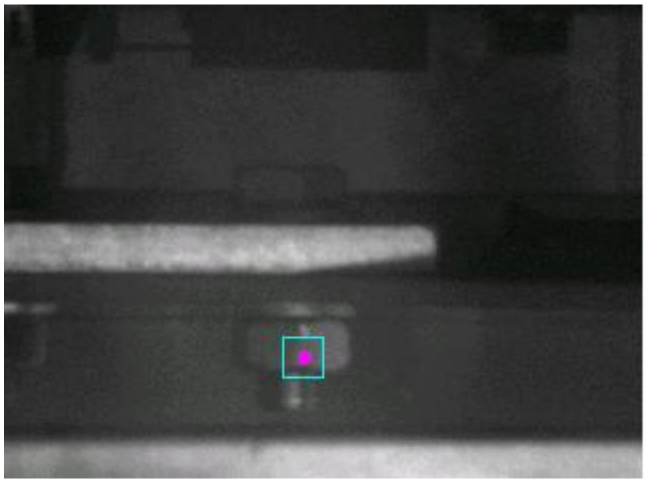


Figure 2: A Contrast Stretched Bad BCAM Image

In this image, the brightest area is highlighted (for those with color versions of this document, the brightest area is colored magenta and identified with a cyan square). If the alignment system uses the centroid of the brightest area of this image as a basis for detector geometry reconstruction, it would not be good – the point source is a reflection, not a light source.

The purpose of the BCAM Image Quality Analysis tools is to examine all of the produced BCAM images and “pass” reasonable images like Figure 1 and “fail” images like Figure 2.

1.2 The RasNiK

A RasNiK[2] is composed of a CCD camera, a lens and a mask. The camera looks at the mask through a lens, and the mask is illuminated from behind through a diffuser. Figure 3 shows a detail of a section of a RasNiK mask.

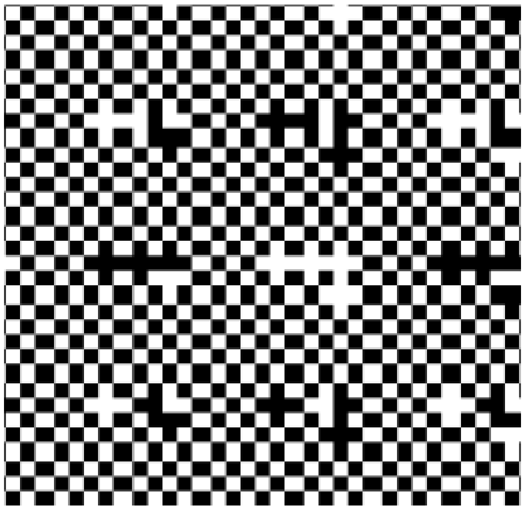


Figure 3: Detail of a RasNiK Mask

The regular chessboard pattern of the mask is interrupted by groups of squares which encode a rough position on the mask. As in the BCAM system, there are cases where the image is beautiful and the analysis can be completely trusted, as is shown in Figue 4.

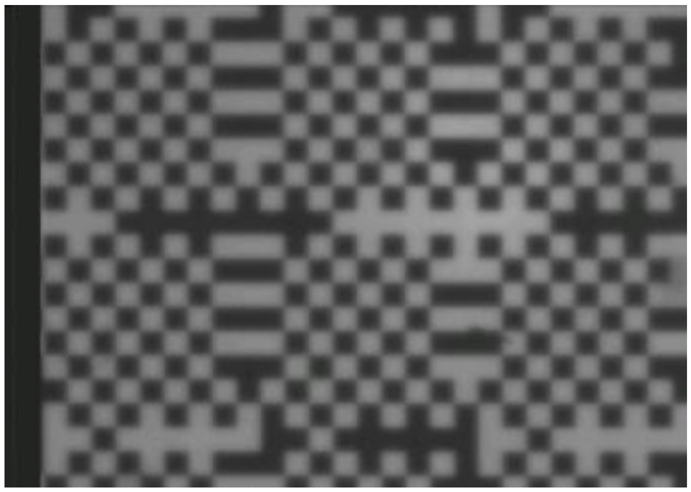


Figure 4: A Good Quality RasNiK Image

The quality of images that can be successfully analyzed by the RasNiK Acquisifier Tool is sometimes quite surprising. Figure 5 shows an image which can provide good information despite the apparent presence of large quantities of dust and hair on the mask. Presumably this is due to the large amount of redundant information.

[2] The capitalization of the letters in RasNiK varies depending on the source of the document describing the system. RasNiK is used in the NIKHEF technical description and so we use that convention here.

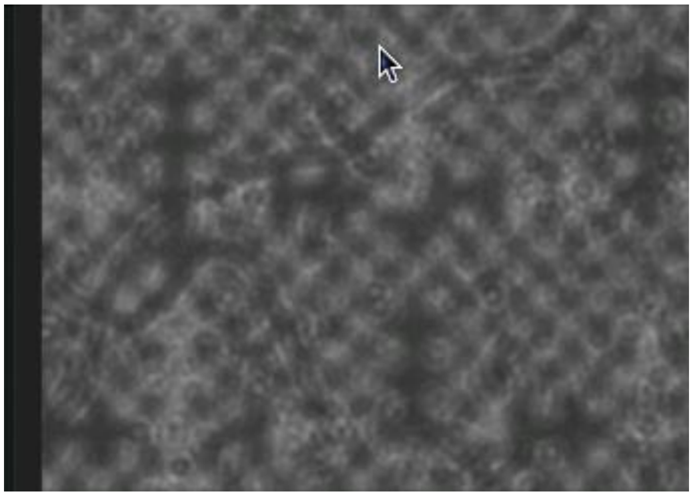


Figure 5: A Marginal Quality RasNiK Image

Images such as Figure 5 must be passed by the image quality tests; however, images such as Figure 6 must be identified as bad.

Figure 6: A Bad Quality RasNiK Image

1.3 The Image Quality Analysis Libraries

The programs that actually perform the image quality analysis are of two libraries, the BCAM Image Analysis library (bia.so) and the RasNiK. Image Analysis library (ria.so). These libraries may be found on AFS in the directory,

~align/software_dev/release

These libraries are used in an "online system" that runs under control of the LWDAQ. This system does the initial image capture, performs an image analysis and stores images for future use. The libraries are also used in an "offline" system which is used for later analysis of stored images.

1.2.1 Online Usage in the LWDAQ System

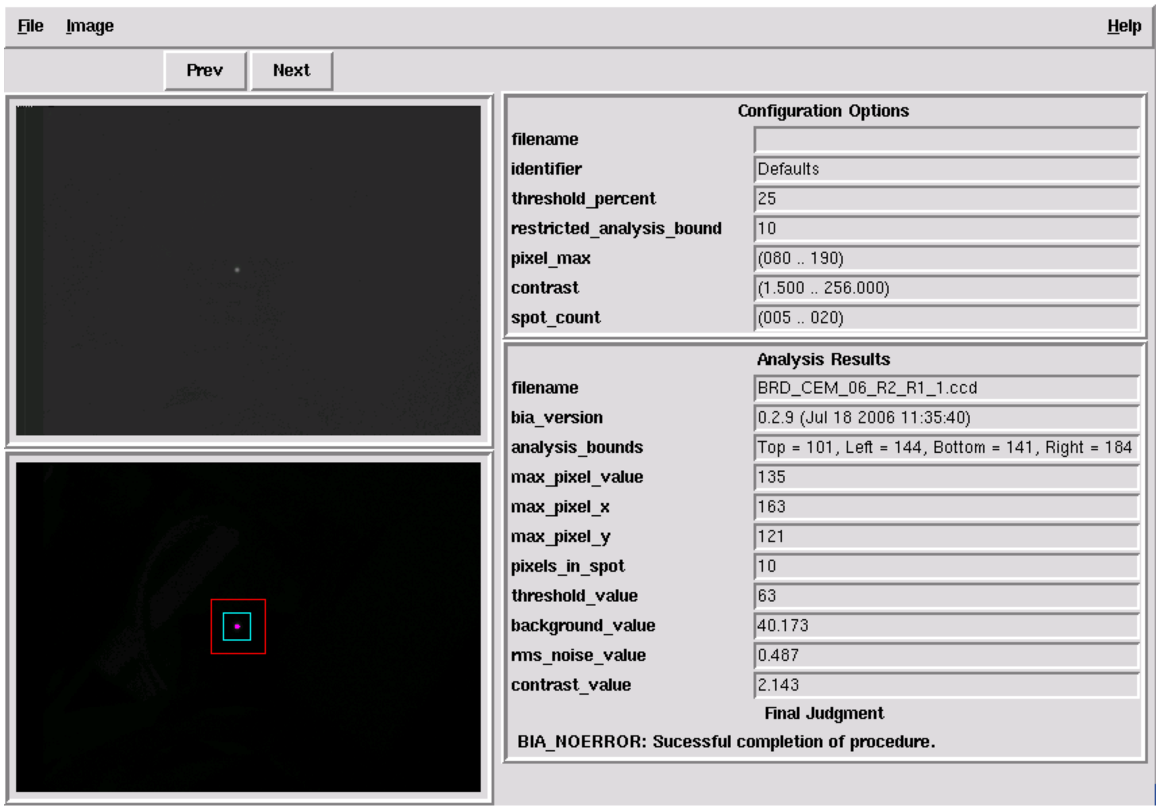
The image analysis libraries are loaded into the LWDAQ system under the control of Acquisifier Scripts. As each image is acquired by the LWDAQ, the image analysis functions are called. A single error code is returned identifying good or bad images; however, several other image characteristics are saved to aid in problem analysis.

The execution of the Image Quality library routines in the online system is specified in the default post processing sections of the LWDAQ Acquisifier Scripts controlling the actual measurement process.

1.2.2 Offline Usage in the Standalone Tools

Two tools are provided for offline use of the image analysis libraries. Every image captured by the LWDAQ system is saved, and these images can be examined individually using the tools. The tools are Tcl/Tk scripts that load the appropriate analysis library (bia.so or ria.so) and perform exactly the same analysis that is done in the LWDAQ online case.[3]

Screen shots of the two tools (BIA and RIA) are shown in Figures 7 and 8.



Configuration Options

filename	
identifier	Defaults
threshold_percent	25
restricted_analysis_bound	10
pixel_max	(080 .. 190)
contrast	(1.500 .. 256.000)
spot_count	(005 .. 020)

Analysis Results

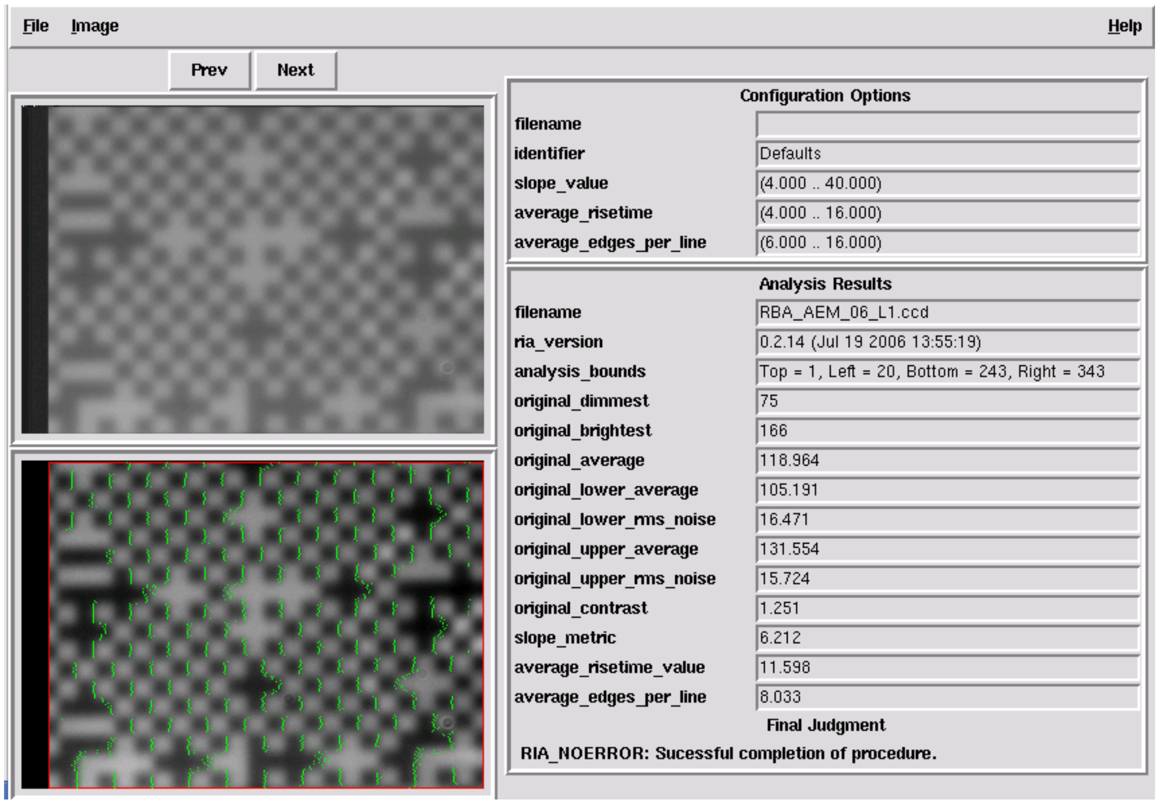
filename	BRD_CEM_06_R2_R1_1.ccd
bia_version	0.2.9 (Jul 18 2006 11:35:40)
analysis_bounds	Top = 101, Left = 144, Bottom = 141, Right = 184
max_pixel_value	135
max_pixel_x	163
max_pixel_y	121
pixels_in_spot	10
threshold_value	63
background_value	40.173
rms_noise_value	0.487
contrast_value	2.143

Final Judgment

BIA_NOERROR: Sucessful completion of procedure.

Figure 7: Screen Shot of BIA Offline Tool

[3] The LWDAQ is also written in Tcl/Tk and so the mechanism and analysis is identical in the online and offline cases.



Configuration Options

filename	
identifier	Defaults
slope_value	(4.000 .. 40.000)
average_risetime	(4.000 .. 16.000)
average_edges_per_line	(6.000 .. 16.000)

Analysis Results

filename	RBA_AEM_06_L1.ccd
ria_version	0.2.14 (Jul 19 2006 13:55:19)
analysis_bounds	Top = 1, Left = 20, Bottom = 243, Right = 343
original_dimmest	75
original_brightest	166
original_average	118.964
original_lower_average	105.191
original_lower_rms_noise	16.471
original_upper_average	131.554
original_upper_rms_noise	15.724
original_contrast	1.251
slope_metric	6.212
average_risetime_value	11.598
average_edges_per_line	8.033

Final Judgment

RIA_NOERROR: Sucessful completion of procedure.

Figure 8: Screen Shot of RIA Offline Tool

The two tools are similar in appearance and operation. Once a file of the appropriate type (.ccd) is opened, the "Prev" and "Next" buttons may be used to navigate through other images in the current directory. The original image is shown in the upper left window, and a processed and annotated image is shown in the lower left. Any configuration options are shown in a window on the upper right and the image analysis results are shown in a window on the lower right. The "Final Judgment" in the lower right corresponds to a decoded version of the result code given to the LWDAQ online system.

To run the tools, the LD_LIBRARY_PATH of your Linux shell must include the directory where the bia.so and ria.so libraries reside. As mentioned above, the released versions of the tools and the libraries may be found in,

~align/software_dev/release

Additionally, an X client must be running, such as Hummingbird Exceed, and the DISPLAY environment variable must be set correctly. The details on accomplishing this vary depending on how you will access the lxplus system.

Once the environment is set, the BIA offline tool may be run by typing "bia.tcl" at the Linux prompt, and the RIA offline tool may be run by typing "ria.tcl" at the linux prompt as in the following example.

```
[lxplus073] ~/software_dev/release > ria.tcl
```

1.2.3 Offline Usage in the Standalone Tools

In order to provide an automated environment for analysis of large numbers of images, batch versions of the analysis tools have been developed. These are called bia and ria, and are run directly from the Linux prompt. Typically, these tools are called from shell scripts that iterate through a directory and perform the analysis on many files. A typical shell script is shown below.

```
#! /bin/tcsh
rm -f bloop.txt
foreach f (afsim/B*.ccd)
./bia $f >>! bloop.txt
end
```

This particular script loops through all of the files in the `afsim` directory that start with the letter 'B' and end with the suffix ".ccd" and calls the offline batch analysis tool (bia) for each of those files. The output of each iteration of the analysis (the result of each file analysis) is saved to the text file, "bloop.txt".

The batch tools are typically used to generate tables of image characteristics that are analyzed graphically using ROOT or Excel. Examples of shell scripts using both the RIA and BIA tools can be found in,

~align/software_dev/release

as `rloop.sh` and `bloop.sh` respectively.

2. The Algorithms

2.1 The BCAM Image Analysis Algorithm

The BCAM Image Analysis Library (bia.so) exports a Tck/Tk command called `Bia_Analyze` that actually performs the image analysis. The BCAM analysis proceeds in several steps. Each of the headings below corresponds to a step in the analysis and a function implemented in the bia.so library.

An image produced by a BCAM contains a region within the image that contains useful information. The size and position of this region is determined by the LWDAQ Acquisifier tool that captured the image, and is stored in the first few bytes of the image file. All of the following functions use these provided analysis bounds.

2.1.1 FindSpot

The function `FindSpot` locates the pixel with the highest intensity located inside the analysis bounds. We assume that this is the brightest point in the BCAM spot and save the x and y coordinates. This point is used as the center of a *restricted* analysis area that separates a small foreground area that contains the brightest spot from the background area described by the analysis bounds.

The values found are placed in global variables that are available to a Tcl/Tk program,

```
Bia_max_pixel_value
Bia_max_pixel_x
Bia_max_pixel_y
```

2.1.2 ValidateAnalysisBounds

`ValidateAnalysisBounds` checks to make sure that the analysis bounds defined by the image bounds, the image valid region, the analysis bounds and the restricted analysis bounds all make sense.

As mentioned above, the image has an active area outside which no valid data is found. It is required that the analysis bounds be entirely contained within this area. It is also required that the restricted analysis bounds be entirely contained within the analysis bounds area. This, in turn, means that brightest point in the image must not be located closer than half of the size of the restricted analysis bounds size to the edge of the analysis bounds.

If the analysis bounds hierarchy is violated, an error code is returned and the image analysis does not proceed.

2.1.3 Background

The function `Background` calculates the average value of the intensities (encoded as bytes with values from 0 to 255) within the analysis bounds section of the array, excluding the restricted analysis bounds.

The calculated value is placed in a global variable that is available to Tcl/Tk

```
Bia_background_value
```

2.1.4 RmsNoise

`RmsNoise` is a function that calculates the Root Mean Square noise value of the bytes within the analysis bounds section of the array, again excluding the restricted analysis bounds section. The RMS noise is the standard deviation of the image intensities, and so reflects the deviations from the background value. Large deviations, and therefore RMS noise values can indicate exposure problems (see Figure 2).

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Bia_rms_noise_value
```

2.1.5 SizeSpot

The function `SizeSpot` finds the number of pixels above a specified threshold around the brightest spot in the image as reported by `FindSpot`. The threshold value is computed according to the '#' method specified in the LWDAQ manual.

```
Bia_threshold_value = (int) (
    Bia_background_value +
    ((double)Bia_max_pixel_value - Bia_background_value) *
    ((double)Bia_threshold_percent / 100.0));
```

No effort is made to ensure that there is connectivity between all of the bright pixels identified. The number of bright (above the threshold) pixels found in the restricted analysis bounds region is placed in a global variable that is available to Tcl/Tk.

```
Bia_pixels_in_spot
```

2.1.6 Contrast

`Contrast` calculates the ratio of the brightest pixel found by `FindSpot` to the calculated threshold value. The threshold calculation is shown in the `SizeSpot` section. This is a measure of the dynamic range of values present in the interesting parts of the image.

The calculated contrast value is placed in a global variable that is available to Tcl/Tk.

```
Bia_contrast_value
```

2.1.7 Cut

The final cut used to determine pass or fail of a given BCAM image is made on the strength of only two measurements:

- The number of pixels in the restricted analysis bounds as reported by `FindSpot` `(Bia_pixels_in_spot);`
- The `Contrast (Bia_contrast_value)` – the ratio of the intensities of brightest spot in the image and the threshold (which is in turn a function of the background).

Default maximum and minimum values for both of these measurements are hard-coded into the bia.so library, but may be modified using configuration files. If an error is detected, a library specific negative error code is returned from `Bia_Analyze`. If the image passes the analysis, a success code (zero) is returned.

2.2 The RasNik Image Analysis Algorithm

The RasNiK Image Analysis Library (ria.so) exports a Tck/Tk command called `Ria_Analyze` that actually performs the image analysis. The RasNiK analysis proceeds in several steps. Each of the headings below corresponds to a step in the analysis and a function implemented in the ia.so library.

An image produced by a RasNiK contains a region within the image that contains useful information. The size and position of this region is determined by the LWDAQ Acquisifier tool that captured the image, and is stored in the first few bytes of the image file. All of the following functions use these provided analysis bounds.

The RasNiK analysis operates on two versions of the image data. The original image and a contrast stretched version. The LWDAQ Acquisifier Tool does a similar stretching in order to extract data from images that are almost imperceptibly varying in intensity in their original form.

2.2.1 Average

The function `Average` calculates the average value of the intensities (encoded as bytes with values from 0 to 255) within the analysis bounds section of the original data array. This provides a value that separates values interpreted as light squares in the RasNiK chessboard from dark square values.

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_average_value
```

2.2.2 UpperAverage

The function `UpperAverage` uses the calculated `Ria_average_value` of the original array to differentiate between bright and dark areas. Bright values (upper values) are those above the average. This function calculates the average value of all pixels with intensities greater than the average value. The resulting number is interpreted as the average value of the light areas of the RasNiK image.

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_upper_average_value
```

2.2.3 LowerAverage

Just as the function `UpperAverage` calculated the average value of the light areas of the RasNiK image, `LowerAverage` calculates the average value of the dark areas. Dark values (lower values) are defined as those below the average value calculated in `Average`

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_lower_average_value
```

2.2.4 UpperRmsNoise

`UpperRmsNoise` is a function that calculates the Root Mean Square noise value of the bytes within the analysis bounds section of the array that are above the average value. The RMS noise is the standard deviation of the image intensities, and so reflects the deviations from the average bright value calculated above.

Large deviations, and therefore RMS noise values can indicate problems such as dust donuts or hairs on the image (see Figure 5). Diffraction effects result in these imperfections being represented in variations in the bright and dark values and therefore contribute to higher than normal RMS noise values.

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_upper_rms_noise_value
```

2.2.5 LowerRmsNoise

`Just as` `UpperRmsNoise` calculates the Root Mean Square noise value of the light areas, `LowerRmsNoise` calculates the RMS noise of the dark areas

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_lower_rms_noise_value
```

2.2.6 ContrastStretchImage

`ContrastStretchImage` takes the dimmest pixel value and the brightest pixel value found in the original image and linearly maps those values to the maximum dynamic range available (0 to 255. This has the effect of increasing the contrast of the image and “bringing out detail.”

2.2.7 Average

The function `Average` is used again to calculate the average value of the bytes within the *contrast stretched* data array. This provides a value that separates values interpreted as light squares in the RasNiK chessboard from dark square values in the contrast stretched image

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_stretched_average_value
```

2.2.8 UpperAverage

The function `UpperAverage`, this time operating on the contrast stretched array, uses the calculated `Ria_stretched_average_value` to differentiate between bright and dark values. This function calculates the average value of all pixels with intensities greater than the average value. This number is interpreted as the average value of the light areas of the contrast-stretched RasNiK image.

 The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_stretched_upper_average_value
```

2.2.9 LowerAverage

`LowerAverage` is used again to calculate the average value of the dark areas in the contrast-stretched image.

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_stretched-lower_average_value
```

2.2.10 UpperRmsNoise

`UpperRmsNoise` is used to calculate.the Root Mean Square noise value of the bright areas of the contrast-stretched array.

Large deviations, and therefore RMS noise values can indicate problems such as dust donuts or hairs on the image (see Figure 5). Diffraction effects result in these imperfections being represented in variations in the bright and dark values. Contrast stretching enhances these imperfections.

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_stretched_upper_rms_noise_value
```

2.2.11 LowerRmsNoise

`Just as UpperRmsNoise` calculates the Root Mean Square noise value of the light areas, `LowerRmsNoise` calculates the RMS noise of the dark areas; this time in the contrast-stretched array.

The calculated value is placed in a global variable that is available to Tcl/Tk.

```
Ria_stretched_lower_rms_noise_value
```

2.2.12 DerivativeX

Although this sounds like a high-powered and time consuming operation, it is actually fairly lightweight – a better name may have been DifferenceX. `DerivativeX` calculates the difference (in the X direction) between adjacent pixels in the image and saves these values in a new derivative array. Just as the first derivative of a function is largest when the rate of change is highest; the values in the derivative array are largest when the rate of change of intensity in the contrast-stretched image array is highest.

The result of this operation is an array that locates the edges in the X direction found in the contrast-stretched array.

2.2.13 CharacterizeEdges

The `CharacterizeEdges` function is the heart of the RasNiK image analysis code. It takes the output of `DerivativeX` as its input. It scans the derivative array looking for rising edges.[4] For each rising edge in the derivative array, the corresponding feature in the original data array is examined.

A slope is calculated. The values of the original data array are taken as points and a fit is done on the pixels around the point identified as the center of the edge. This is the slope of the edge and can be used to estimate a goodness of focus metric in the original image. The average value of this measurement is remembered and is saved in a global variable that is available to Tcl/Tk.

```
Ria_slope_value
```

The rise time of the edge is calculated. The rise time is the number of horizontal pixels it takes for the edge to rise from ten percent above the local average low value to ten percent below the local average high value of the edge. The rise time is also related to the focus of the original image and is a normalized version of the slope – that is, it is not affected by the amplitude of the signal. The average value of this measurement is remembered and is saved in a global variable that is available to Tcl/Tk.

```
Ria_average_risetime_value
```

The average number of low-to-high transitions in each line of the contrast-stretched array is remembered. If a reduced analysis bound is in effect, the number of low-to-high transitions is normalized to what may have been expected if the whole array was used in the calculation.

This value is placed in a global variables that is available to Tcl/Tk.

```
Ria_average_edges_per_line
```

2.2.14 Cut

The final cut used to determine pass or fail of a given image is made on the strength of only one measurement:

- The average number of dark to light transitions (edges) found in each *horizontal* line of the image as reported by `CharacterizeEdges` and found in the corresponding variable `(Ria_average_edges_per_line);`

[4] We have assumed that falling edges will have roughly the same characteristics as rising edges; and that edges in the Y direction will have the same rough characteristics as the edges in the X direction

Default maximum and minimum values for this measurement are hard-coded into the ria.so library, but may be modified using configuration files. If an error is detected, a library specific negative error code is returned from `Ria_Analyze`. If the image passes the analysis, a success code (zero) is returned.

3. Accessing the Shared Libraries

The image analysis libraries are implemented in C as Linux shared libraries. Shared libraries in Linux correspond to Dyncamically Loaded Libraries (DLLs) in the Windows system. They are loaded when they are needed by a program which needs to access them.

In most cases, the programs which make use of the image analysis libraries are Tcl/Tk scripts, and so a mechanism for exporting the functions and variables in the shared library to Tcl/Tk must be implemented. This is a tedious process if done manually, and so the mechanism has been automated using a tool called Simplified Wrapper and Interface Generator (SWIG).[5]

Technically, SWIG is an Interface Compiler. It takes declarations of the functions and variables in the shared library C code, and uses them to generate "wrapper" functions that Tcl/Tk uses to access the library. The result is that, although the image analysis library is implemented as a set of functions written in C; once loaded into a Tcl/Tk script, it appears there as a set of commands just as if they were written in Tcl.

This means that one can access the libraries interactively using the Tcl shell. For example,

```
[lxplus063] ~/software_dev/release > tclsh
% load bia.so Bia
% puts [Bia_Version]
0.2.9 (Jul 18 2006 11:35:40)
%
```

If you type the commands in **`bold`** (or in **`bold blue`** if you have the color version of this document) you can find the version of the bia.so library.

- The command **`tclsh`** loads the Tcl shell (the interpreter);
- The command **`load bia.so Bia`** loads the BCAM Image Analysis Library;
- The command **`puts [Bia_Version]`** runs the library function `Bia_Version` and prints the returned string.

The version of bia.so in this example is 0.2.9 that was compiled on July 18, 2006 at 11:35:40 AM.

[5] See http://www.swig.org/ for details.

All of the interaction between the library and the online system and the offline tools is done in a similar way.

3.1 The BCAM Image Analysis Library Functions

The functions made available to Tck/Tk from the BCAM Image Analysis library can be found in the file bia.i – the interface file used by SWIG to create the wrapper functions. The working directory that holds the source code for the BCAM library is,

~align/software_dev/bia

Since the information in this file ultimately ends up being compiled by the C compiler when the library is made, the syntax is C. In bia.i you will find declarations like,

```
extern char *Bia_Version(void);
```

This is the declaration of the version command executed in the previous section. This declaration tells the SWIG compiler that there is a function called `Bia_Version` in the library that takes no parameters and returns a pointer to a char. SWIG then generates the wrapper code to enable the export to and access by Tcl/Tk scripts.

The interface declarations are split into several sections.

3.1.1 Configuration Variables

The configuration variables control the basic operation of the library functions. They have compiled-in default values and may be set directly using Tcl/Tk or in some cases by asking the library to load and parse an rc file (rc stands for run command – a run-time configuration file in Unix parlance).

```
extern int Bia_verbose;
extern int Bia_do_color;
extern int Bia_threshold_percent;
extern int Bia_restricted_analysis_bound;
```

3.1.2 Cut Values

Cut values specify exactly how the pass-fail decision is to me made. They have compiled-in default values and may be set directly using Tcl/Tk or by asking the library to load and parse an rc file (rc stands for run command – a run-time configuration file in Unix parlance).

```
extern int    Bia_pixel_max_low;
extern int    Bia_pixel_max_high;
extern double Bia_contrast_low;
extern double Bia_contrast_high;
extern int    Bia_spot_count_low;
extern int    Bia_spot_count_high;
```

3.1.3 Functions

These are the functions that are exported from the library to Tck.Tk.

```
extern char *Bia_TranslateError(int);
extern char *Bia_Version(void);
extern int   Bia_LoadRcFile(char *pFilename);
extern char *Bia_RcIdentifier(void);
extern int   Bia_LoadCcdArray(char *pFilename);
extern int   Bia_Analyze(void);
```

3.1.3 Results

These are the variables that hold the results of the various measurements that were made during the analysis. The Cut Values specify acceptable ranges for the results that are used in making the pass-fail decision.

```
extern double Bia_background_value;
extern int    Bia_max_pixel_value;
extern int    Bia_max_pixel_x;
extern int    Bia_max_pixel_y;
extern int    Bia_pixels_in_spot;
extern int    Bia_threshold_value;
extern double Bia_rms_noise_value;
extern double Bia_contrast_value;
extern int    Bia_analysis_bounds_top;
extern int    Bia_analysis_bounds_left;
extern int    Bia_analysis_bounds_bottom;
extern int    Bia_analysis_bounds_right;
extern char  *Bia_results_string;
```

3.1.4 Offline Tool Interfaces

This section contains functions that are not part of the mainline image analysis, but support the offline tool.

```
extern int Bia_UploadImage(char *pImagename,
    char *pSource);
```

3.2 The RasNiK Image Analysis Library Functions

The functions made available to Tck/Tk from the RasNiK Image Analysis library can be found in the file ria.i – the interface file used by SWIG to create the wrapper functions. The working directory that holds the source code for the BCAM library is,

~align/software_dev/ria

Since the information in this file ultimately ends up being compiled by the C compiler when the library is made, the syntax is C. In bia.i you will find declarations like,

```
extern char *Ria_Version(void);
```

This declaration tells the SWIG compiler that there is a function called `Ria_Version` in the library that takes no parameters and returns a pointer to a char. SWIG then generates the wrapper code to enable the export to and access by Tcl/Tk scripts.

The interface declarations are split into several sections.

3.2.1 Configuration Variables

The configuration variables control the basic operation of the library functions. They have compiled-in default values and may be set directly using Tcl/Tk.

```
extern int Ria_verbose;
extern int Ria_do_color;
```

3.2.2 Cut Values

Cut values specify exactly how the pass-fail decision is to me made. They have compiled-in default values and may be set directly using Tcl/Tk or by asking the library to load and parse an rc file (rc stands for *r*un command – a run-time configuration file in Unix parlance).

```
xxtern double  Ria_slope_value_low;
extern double  Ria_slope_value_high;
extern double  Ria_average_risetime_low;
extern double  Ria_average_risetime_high;
extern double  Ria_average_edges_per_line_low;
extern double  Ria_average_edges_per_line_high;
```

3.2.3 Functions

These are the functions that are exported from the library to Tck.Tk.

```
extern char *Ria_TranslateError(int);
extern char *Ria_Version(void);
extern int   Ria_LoadRcFile(char *pFilename);
extern char *Ria_RcIdentifier(void);
extern int   Ria_LoadCcdArray(char *pFilename);
extern int   Ria_Analyze(void);
```

3.2.3 Results

These are the variables that hold the results of the various measurements that were made during the analysis. The Cut Values specify acceptable ranges for the results that are used in making the pass-fail decision.

```
extern int    Ria_brightest_pixel_value;
extern int    Ria_dimmest_pixel_value;
extern double Ria_average_value;
extern double Ria_upper_average_value;
extern double Ria_lower_average_value;
extern double Ria_upper_rms_noise_value;
extern double Ria_lower_rms_noise_value;
extern double Ria_contrast_value;
extern int    Ria_analysis_bounds_top;
extern int    Ria_analysis_bounds_left;
extern int    Ria_analysis_bounds_bottom;
extern int    Ria_analysis_bounds_right;
extern char  *Ria_results_string;
extern double Ria_slope_value;
extern double Ria_average_risetime_value;
extern double Ria_average_edges_per_line;
```

3.2.4 Offline Tool Interfaces

This section contains functions that are not part of the mainline image analysis, but support the offline tool.

```
extern int Ria_UploadImage(char *pImagename,
    char *pSource);
```

4. Specifying the Cut Values Via Configuration Files

As mentioned above, the purpose of the image analysis system is to ensure that images captured by the online system are of adequate quality. Both the BCAM Image Analysis library (bia.so) and the RasNiK Image Analysis library (ria.so) make a number of measurements and condense those measurements into one or two variables that describe the quality of the image.

In order to make the final decision on the adequacy of the images, we define certain ranges of the image variables as representing characteristics of images that make them acceptable.

The values for the cuts can be specified in two ways: either through hard-coded default values specified in the library source code, or dynamically with run-time loadable configuration files (known as run command files files in Unix).[6]

The configuration files are loaded by the libraries when the corresponding Tcl/Tk command is executed. For the BCAM Image Analysis library the command is

[6] If you're familiar with Unix shells, the files .tcshrc or .bashrc are examples of run command files. These are usually hidden files (beginning with the dot character).

Bia_LoadRcFile. An interactive example would look something like the following script,

```
set fn [tk_getOpenFile]
Bia_LoadRcFile $fn
```

The actual rc files are text files that contain a list of names and initialization values for certain configuration and cut parameters. An example rc file for the bia.so library is shown in the file,

~align/software_dev/bia/bia.rc

This file contains the following declarations,

```
[BIA]
Bia_rc_identifier = "Thu Jul  6 13:55:40 CEST 2006 cdowell"
Bia_threshold_percent = 25
Bia_pixel_max = (80 .. 190)
Bia_contrast = (1.5 .. 256.0)
Bia_spot_count = (5 .. 20)
Bia_analysis_bound = 10
```

The line containing "[BIA]" is a section identifier. It is possible to group configurations for the BIA and RIA libraries into a single file. This section identifier tells the program that the following section contains declarations for the BCAM Image Analysis library.

The syntax[7] of a section is very simple:

```
<variable name> = <value> | <range>
```

The <variable name> is an internal variable in the library. You cannot pick any variable and expect it to work however. The library must have code written to recognize the string <variable name> and store the <value> or the <range>.

The last entry in the example above,

```
Bia_analysis_bound = 10
```

ultimately causes the bia.so variable Bia_analysis_bound to be set to the value of 10 (decimal).

The entry,

[7] This isn't intended to be a BNF description, just enough information to understand what the file means.

```
Bia_spot_count = (5 .. 20)
```

specifies a range concerning the library variable `Bia_spot_count.` The implementation actually sets two variables in the library in the following way,

```
Bia_spot_count_low = 5
Bia_spot_count_high = 20
```

The following line shows that a range can include floating point numbers;

```
Bia_contrast = (1.5 .. 256.0)
```

And the line,

```
Bia_rc_identifier = "Thu Jul  6 13:55:40 CEST 2006 cdowell"
```

is a string assignment.

4.1 The BCAM Library Cut Values

As described above, the final cut used to determine passing or failing a given BCAM image is made on the strength two variables:

- `Bia_pixels_in_spot`;
- `Bia_contrast_value`.

The meaning of these two variables is given in the library functions section above. The default maximum and minimum values for both of these measurements may be found in the header file bia.h which, in turn, is located on AFS in the directory,

~align/software_dev/bia

The relevant declarations are,

```
#define BIA_CONTRAST_LOW_DEFAULT                (1.5)
#define BIA_CONTRAST_HIGH_DEFAULT               (256.0)

#define BIA_SPOT_COUNT_LOW_DEFAULT              (5)
#define BIA_SPOT_COUNT_HIGH_DEFAULT             (20)
```

In the default case, the size of a spot in a BCAM image must then be from 5 to 20 pixels, inclusive to be declared adequate and therefore pass the test. The contrast value (recall that this is the ratio of the intensity of the brightest pixel divided by the threshold) must be from 1.5 to 256.0, inclusive.

As described above, these limits may be changed dynamically by loading an “rc” file. The provided example file, bia.rc, has declarations that cause the default values to be set. This file may be found in the AFS in the directory,

~align/software_dev/bia

The relevant declarations are

```
Bia_contrast = (1.5 .. 256.0)
Bia_spot_count = (5 .. 20)
```

If it were decided that the spot count cut was too generous, and numbers describing adequate images must be from 5 to 16, just change the number 20 to 16.

Other configuration items that can be set at run time are:

- `Bia_threshold_percent` – the threshold described in the calculation illustrated in the `SizeSpot` function, above.
- `Bia_analysis_bound` – determines the size of the restricted analysis bounds. The resulting area has sides of length 1 + 2 * `Bia_analysis_bound`.

4.2 The RasNiK Library Cut Values

The final cut used to determine pass or fail of a given image is made on the strength of only one measurement:

- The average number of dark to light transitions (edges) found in each *horizontal* line of the image as reported by `CharacterizeEdges` and found in the corresponding variable `(Ria_average_edges_per_line);`

The meaning of this variable may be found in the library function section. The default maximum and minimum value may be found in the header file ria.h which, in turn, is located on AFS in the directory,

~align/software_dev/ria

The relevant declarations are,

```
#define RIA_AVERAGE_EDGES_PER_LINE_LOW_DEFAULT  (6.0)
#define RIA_AVERAGE_EDGES_PER_LINE_HIGH_DEFAULT (16.0)
```

In the default case, the average number of rising edges in the X direction per line of a RasNiK image must then be from 6.0 to 16.0, inclusive, to be declared adequate and therefore pass the test.

As described above, these limits may be changed dynamically by loading an “rc” file. The provided example file, ria.rc, has declarations that cause the default values to be set. This file may be found in the AFS in the directory,

~align/software_dev/ria

The relevant declaration is

```
Ria_average_edges_per_line = (6.0 .. 16.0)
```

If it were decided that the edge count cut was too generous, and numbers describing adequate images must be from 6.0 to 12.0, just change the floating point number 16.0 to 12.0.

Other configuration items that can be set at run time are:

- `Ria_slope_value` – a currently unused cut range on the value of the slope of the edges (a focus metric).
- `Ria_average_risetime` – a currently unused cut range on the value o the rise time o the edes (another focus metric).

5. LWDAQ Acquisifier Script Modifications

In order to use the image analysis library functions the LWDAQ Acquisifier script controlling the acquisition of the image data must be modified. A generic script header has been developed that includes the changes required to invoke and report image analysis and quality. This header may be found on AFS in the directory,

~align/lwdaq

This header file, `script_header.txt`, must be used as a basis for any scripts that intend to use the analysis libraries and report results through the “step results” mechanism to the LWDAQ Controller (lc) and to the alignment results database. In order to create a new script, the file `script_header.txt` is typically copied to a new file, `SEC_AEM_11.acq`, for example. A file containing the “`acquire:`” commands corresponding to a list of devices to be measured are then concatenated to the end of this file, yielding a new, complete acquisifier script.

The modifications to the acquisifier script (located in `script_header.txt`) that were required to add image analysis are found in the `default_post_processing` sections.

5.1 The BCAM Default Post-Processing Changes

The following section of code actually performs the load o the BCAM Image Analysis library. The version is extracted and printed on the LWDAQ acquisifier console. The commands mapped by SWIG into image analysis library calls are shown in **bold** (or **bold blue** if you have the color version of this document).

```
#
# Load the BCAM Image Analysis library and print the version string.
#
    load /afs/cern.ch/user/a/align/software_dev/release/bia.so Bia
    set output [format "Loaded BIA version %s" [Bia_Version]]
    Acquisifier_print $output purple
```

The next section is where the actual image data is written to a file which can examined later during offline analysis.

```
#
# Write the BCAM image out to the results directory so the analysis
program
# can get to it.
#
    set fn [file join [file dirname $config(run_results)] $name\.ccd]
    LWDAQ_write_image_file $iconfig(memory_name) $fn
```

The actual image analysis functions are called next. An error code is returned from the library indicating that the image has passed or failed the analysis.

```
#
# Perform the image analysis.
#
    Acquisifier_print "Analyzing Image ..." purple
    set ::Bia_verbose 0
    Bia_LoadCcdArray $fn
    set error_code [Bia_Analyze]
    set msg_string [Bia_TranslateError $error_code]

    if {$error_code == 0} {
        Acquisifier_print "Image okay" purple
    } else {
        Acquisifier_print $msg_string red
    }
```

In addition to the error code, we retrieve the values on which the cut was made, and some other values that might be useful in the analysis of problems.

```
#
# Pull out the results that were used to make the cut.
#
    set pixels_in_spot $::Bia_pixels_in_spot
    set contrast $::Bia_contrast_value
#
# Pull out other possibly interesting / useful results.
#
    set background $::Bia_background_value
    set background_noise $::Bia_rms_noise_value
```

These values are appended to the normal printout of results according to the key shown in the box below. The first column is a `printf` format type. The type "%d" indicates an integer value will be printed. The type "%1.3f" indicates that a floating point value at least one character long, but including three digits to the right of the decimal (e.g., 0.135). The two letter code following the format type is also printed. For example, if an error code of zero was returned by the library, the results would show the string "0 ec" appended to the string.

```
#     %d ec         -- the error code from Bia_Analyze
#     %d np         -- the number of pixels in the spot from
                       Bia_pixels_in_spot
#     %1.3f co      -- the contrast value from Bia_contrast_value
#     %1.3f bg      -- the average intensity of the background pixels
#     %1.3f rn      -- the rms noise of the background pixels
```

The following box shows an example of a BCAM result string. The fields reporting the image analysis results are shown in a **`larger bold font`** (colored **`blue`** if you have the color version of this document).

```
BAZ_AEM_10_08_C4_1 2.79500 0.30500 0.00000 1154593265
#DQMB BAZ_AEM_10_08_C4_1 1 px 53 in 0.000 se 52 th 56689 ex 51 \
av 0 st 53 mx 49 mn -3010 ec 0 np 0.000 co 0.000 bg 0.000 rn
```

5.2 The RasNiK Default Post-Processing Changes

The following section of code actually performs the load o the BCAM Image Analysis library. The version is extracted and printed on the LWDAQ acquisifier console. The commands mapped by SWIG into image analysis library calls are shown in **`bold`** (or **`bold blue`** if you have the color version of this document).

```
#
# Load the RasNIK Image Analysis library and print the version string.
#
    load /afs/cern.ch/user/a/align/software_dev/release/ria.so Ria
    set output [format "Loaded RIA version %s" [Ria_Version]]
    Acquisifier_print $output purple
```

The next section is where the actual image data is written to a file which can examined later during offline analysis.

```
#
# Write the RasNIK image out to the results directory so the analysis
program
# can get to it.
#
    set fn [file join [file dirname $config(run_results)] $name\.ccd]
    LWDAQ_write_image_file $iconfig(memory_name) $fn
```

The actual image analysis functions are called next. An error code is returned from the library indicating that the image has passed or failed the analysis.

```
#
# Perform the image analysis.
#
    Acquisifier_print "Analyzing Image ..." purple
    set ::Ria_verbose 0
    Ria_LoadCcdArray $fn
    set error_code [Ria_Analyze]
    set msg_string [Ria_TranslateError $error_code]
    if {$error_code == 0} {
        Acquisifier_print "Image okay" purple
    } else {
        Acquisifier_print $msg_string red
    }
```

In addition to the error code, we retrieve the values on which the cut was made, and some other values that might be useful in the analysis of problems.

```
#
# Pull out the results that were used to make the RIA cut.
#
    set edges_per_line $::Ria_average_edges_per_line
#
# Pull out other possibly interesting / useful results.
#
    set contrast $::Ria_contrast_value
    set sharpness $::Ria_average_risetime_value
    set bright_noise $::Ria_upper_rms_noise_value
    set dim_noise $::Ria_lower_rms_noise_value
```

These values are appended to the normal printout of results according to the key shown in the box below. The first column is a `printf` format type. The type “%d” indicates an integer value will be printed. The type “%1.3f” indicates that a floating point value at least one character long, but including three digits to the right of the decimal (e.g., 0.135). The two letter code following the format type is also printed. For example, if an error code of zero was returned by the library, the results would show the string “0 ec” appended to the string.

```
#     %d ec         -- the error code from Bia_Analyze
#     %1.3f el      -- the average number of edges per line
#     %1.3f co      -- the contrast value; diff betw avg bright and
#                      dim areas
#     %1.3f ln      -- the rms noise of the light pixels
#     %1.3f dn      -- the rms noise of the dark pixels
#     %1.3f sh      -- the sharpness metric (risetime of bright /
#                      dim changes)
```

The following box shows an example of a RasNiK result string. The fields reporting the image analysis results are shown in a **larger bold font** (colored **blue** if you have the color version of this document).

```
RPL_AEM_11_C2_W_W 68.7160 29.2584 0.999957 1154593303
#DQMR RPL_AEM_11_C2_W_W -0.002089 dm -4.830 ro 0.13 dx \
120 sq 2 or 2 rf 21749 ex 86 av 24 st 137 mx 40 mn \
0 ec 11.062 el 1.660 co 23.453 ln 24.248 dn 8.703 sh
```

6. Return Codes

Each of the modules composing the image analysis libraries has been assigned a range of error codes. These are found in

~align/software_dev/error_codes.h

The assignments to the components that could produce error codes in the image analysis results are the following,

```
// Run command file parser
#define RC_ERROR_BASE                    (-2000)
// BCAM image analysis library
#define BIA_ERROR_BASE                   (-3000)
// Rasnik image analysis library
#define RIA_ERROR_BASE                   (-4000)
```

The errors from the command file parser are returned during the calls to `Bia_LoadRcFile` or `Ria_LoadRcFile`. These errors could be in the range -2000 to -2999.

Errors from the BCAM image analysis may be in the range -3000 to -3999, and errors from the RasNiK image analysis may be in the range -4000 to -4999.

In all cases, a return code of 0 indicates successful completion of a given procedure.

6.1 The BCAM Return Codes

The return values that the BCAM Image Analysis library (bia.so) uses are found in the header file for the C language implementation of the library. This file may be found in,

~align/software_dev/bia/bia.h

The values are shown in the following preprocessor definitions,

```
#define BIA_ERROR                          (BIA_ERROR_BASE)
#define BIA_ERROR_FOPEN                    (BIA_ERROR_BASE - 1)
#define BIA_ERROR_FILE_LENGTH              (BIA_ERROR_BASE - 2)
#define BIA_ERROR_CCD_FILE_NOT_LOADED      (BIA_ERROR_BASE - 3)
#define BIA_ERROR_SPOT_AT_EDGE             (BIA_ERROR_BASE - 4)
#define BIA_ERROR_PIXEL_MAX                (BIA_ERROR_BASE - 5)
#define BIA_ERROR_SPOT_SIZE                (BIA_ERROR_BASE - 6)
#define BIA_ERROR_CONTRAST                 (BIA_ERROR_BASE - 7)
#define BIA_ERROR_INVALID_BOUNDS           (BIA_ERROR_BASE - 8)
#define BIA_ERROR_BOUNDS_SIZE              (BIA_ERROR_BASE - 9)
#define BIA_ERROR_BOUNDS_INTERSECTION      (BIA_ERROR_BASE - 10)
#define BIA_ERROR_REQUIRES_SPOT            (BIA_ERROR_BASE - 11)
```

Recall that `BIA_ERROR_BASE` is -3000, and so error number -3001 corresponds to `BIA_ERROR_FOPEN`.

A function in the library is provided to translate these error codes into a more easily understood form. The following error description may appear on the acquisifier console or in the “final judgment” section of the BCAM version of the offline tool. This translation is not made in the step results output.

"BIA_NOERROR: Successful completion of procedure."

"BIA_ERROR: General unspecified error condition."

"BIA_ERROR_FOPEN: Error opening a file (config or CCD)."

"BIA_ERROR_FILE_LENGTH: The CCD file is not long enough to be a CCD array."

"BIA_ERROR_CCD_FILE_NOT_LOADED: An attempt to perform an analysis without loading a CCD file was made."

"BIA_ERROR_SPOT_AT_EDGE: The BCAM spot is too close to the edge of the CCD array to perform the analysis."

"BIA_ERROR_PIXEL_MAX: The intensity of the brightest pixel in the spot was found to be outside the specified limits."

"BIA_ERROR_SPOT_SIZE: The number of pixels (located within the restricted analysis bounds) with intensities greater than the calculated threshold was found to be outside the specified limits."

"BIA_ERROR_CONTRAST: The contrast metric of the image was found to be outside the specified limits, "BIA_ERROR_INVALID_BOUNDS: The archived analysis bounds stored in the image do not define a rectangle within the valid region of the image."

"BIA_ERROR_BOUNDS_SIZE: The archived analysis bounds stored in the image do not define a rectangle that can contain any restricted analysis bounds region of the configured size (the archived bounds are too small or the restricted bounds are too large)."

"BIA_ERROR_BOUNDS_INTERSECTION: The archived analysis bounds stored in the image do not define a rectangle that completely contains the calculated restricted analysis bounds region (the archived bounds are too small or the restricted bounds are too large; or the brightest spot is too close to the edge of the archived bounds)."},

6.2 The RasNiK Return Codes

The return values that the RasNiK Image Analysis library (ria.so) uses are found in the header file for the C language implementation of the library. This file may be found in,

~align/software_dev/ria/ria.h

The values are shown in the following preprocessor definitions,

```
#define RIA_ERROR                          (RIA_ERROR_BASE)
#define RIA_ERROR_FOPEN                    (RIA_ERROR_BASE - 1)
#define RIA_ERROR_FILE_LENGTH              (RIA_ERROR_BASE - 2)
#define RIA_ERROR_CCD_FILE_NOT_LOADED      (RIA_ERROR_BASE - 3)
#define RIA_ERROR_REQUIRES_ANALYSIS        (RIA_ERROR_BASE - 4)
#define RIA_ERROR_RISETIME_VALUE           (RIA_ERROR_BASE - 5)
#define RIA_ERROR_EDGES_VALUE              (RIA_ERROR_BASE - 6)
#define RIA_ERROR_SLOPE_VALUE              (RIA_ERROR_BASE - 7)
```

Recall that `RIA_ERROR_BASE` is -4000, and so error number -4001 corresponds to `RIA_ERROR_FOPEN`.

A function in the library is provided to translate these error codes into a more easily understood form. The following strings may appear on the acquisifier console or in the "final judgment" section of the BCAM version of the offline tool. This translation is not made in the step results output.

"RIA_NOERROR: Sucessful completion of procedure."

"RIA_ERROR: General unspecified error condition."

"RIA_ERROR_FOPEN: Error opening a file (config or CCD)."

"RIA_ERROR_FILE_LENGTH: The CCD file is not long enough to be a CCD array."

"RIA_ERROR_CCD_FILE_NOT_LOADED: An attempt to perform an analysis without loading a CCD file was made."

"RIA_ERROR_REQUIRES_ANALYSIS: An attempt was made to execute a function that depends on analysis values without previously performing an analysis."

"RIA_ERROR_RISETIME_VALUE: The average value of the risetime (in pixels) of the image was found to be outside acceptable bounds."

"RIA_ERROR_SLOPE_VALUE: The average value of the slope of the edges of the image was found to be outside acceptable bounds."

"RIA_ERROR_EDGES_VALUE: The average value of the number of edges per line of the image was found to be outside acceptable bounds."