# Infinite-ISP Tuning Tool v1.1

**User Guide**

**14th July, 2023**

| Sr. | Description | Version | Date |
|---|---|---|---|
| 1. | Initial draft | v0.1 | 26-05-2023 |
| 2. | Figures and description updated | v0.2 | 29-05-2023 |
| 3. | Formatting and description improvements | v1.0 | 29-05-2023 |
| 4. | New features added:<br>1. Zoom in/out.<br>2. Apply BLC/WB options.<br>3. Generate init_isp.h file.<br>4. Update sensor info. | V1.1 | 14-17-2023 |

# Table of Contents

# List of Figures

# Introduction

## Overview

The Infinite-ISP Tuning Tool is specifically designed to analyze and fine-tune the performance of the modules within the Infinite-ISP_ReferenceModel pipeline. Infinite-ISP_ReferenceModel is a collection of camera pipeline modules implemented at the application level to convert an input RAW image from a sensor to an output RGB image. The overall workflow for this tuning tool is visible in Figure 1.

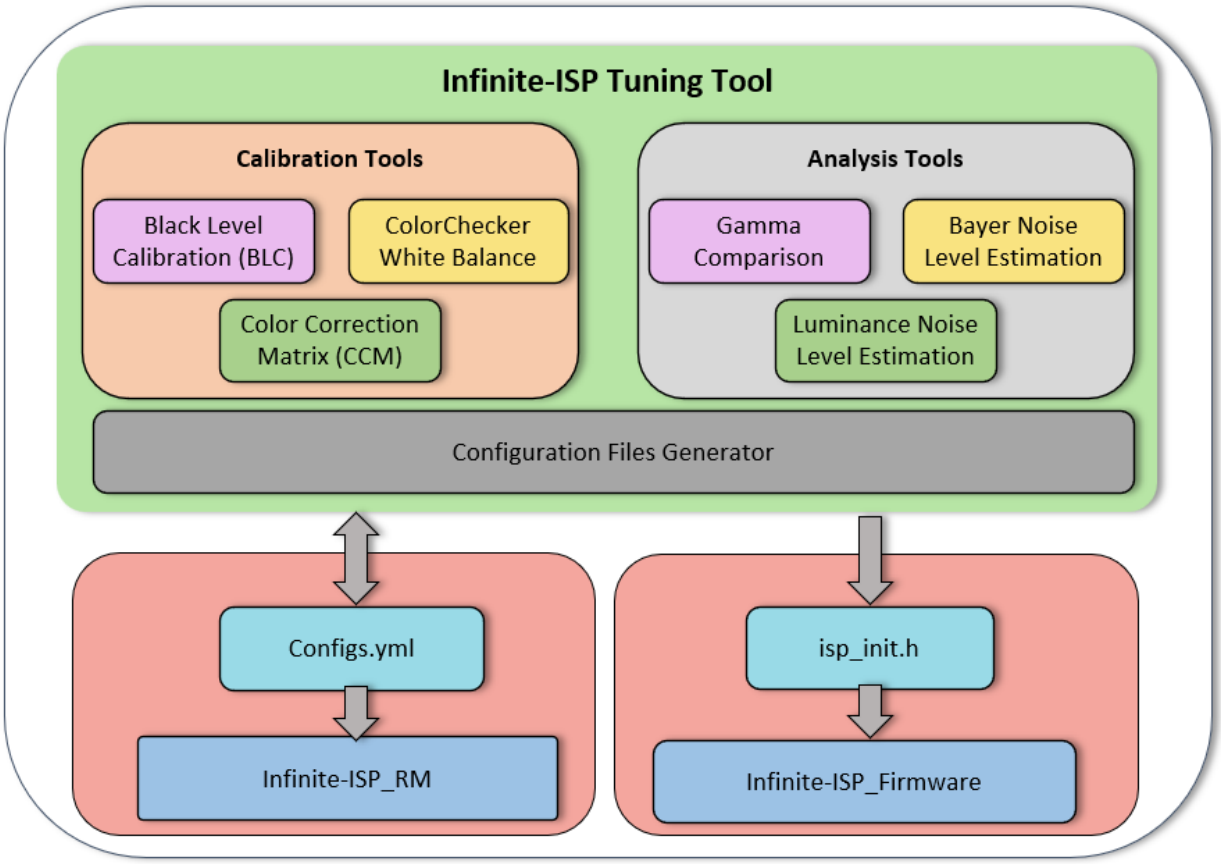


*Figure 1: An overview of the Infinite-ISP Tuning Tool*

This tool can be used separately as a stand-alone application for image quality analysis. This comprehensive toolset not only provides the user with precise control over the calibration modules, but also allows the user to achieve outstanding image quality through the analysis modules. With respect to its usage, it is divided into three categories:

1. Calibration Tools: Used to generate algorithm parameters for modules like black levels, white balance gains, and color correction matrix.
2. Analysis Tools: Used to provide common data and graphs to support analysis like gamma curves, bayer & luma noise levels.

3. Generate Configuration Files: Used to generate baseline configuration files for Infinite-ISP_ReferenceModel pipeline and its FPGA model. Before generating these files, users have the option to update the sensor information like bit depth, Bayer pattern, height, and width.

To make sure the tuning tool and the Infinite-ISP_ReferenceModel pipeline work together seamlessly, they use a special file called default_configs.yml. This file plays a crucial role by specifying the specific input parameters for each module within the pipeline. It serves as the default configuration file and is essential for running the tuning tool. When the tuning tool is started, a duplicate of this file named configs.yml is created, that grants users the flexibility to adjust the parameters according to their unique needs and requirements. For this, the tuning tool will help to fine tune the modules by providing calibration and analysis assistance. By doing so, users can fine-tune the performance of the pipeline, ensuring optimal results.

# Features

Infinite-ISP_ReferenceModel provides a set of camera pipeline modules that needs calibration and analysis for providing quality results. For this, tuning tool provides user with the following features present in Table 1:

*Table 1: Features of the Infinite-ISP Tuning Tool*

| | Modules | Description |
|---|---|---|
| **Calibration Tools** | Black Level Calibration (BLC) | Calculates the black levels of a raw image for each channel (R, Gr, Gb, and B). |
| | ColorChecker White Balance (WB) | Calculates the white balance gains (R gain and B gains) on a ColorChecker RAW or RGB image. |
| | Color Correction Matrix (CCM) | Calculates a 3x3 color correction matrix using a ColorChecker RAW or RGB image. |
| **Analysis Tools** | Gamma Curves (GC) | Compares the user-defined gamma curve with the sRGB color space gamma ≈ 2.2. |
| | Bayer Noise Level Estimation (BNE) | Estimates the noise levels of the six grayscale patches on a ColorChecker RAW image. |
| | Luminance Noise Level Estimation (LNE) | Estimates the luminance noise level of the six grayscale patches on a ColorChecker RAW or RGB image. |
| **Generate Configuration Files** | Update Sensor Info | Updates sensor information such as bit depth, bayer pattern, height and width in the configs.yml. |
| | Generate configs.yml | Generates a configuration file for the Infinite-ISP_ReferenceModel. |
| | Generate isp_init.h | Generates a configuration file for FPGA. |

## Directory Description

Within the base directory, *Infinite-ISP_TuningTool*, the user will find following folders and files that are essential for the functioning of the tuning tool as shown in Figure 2.

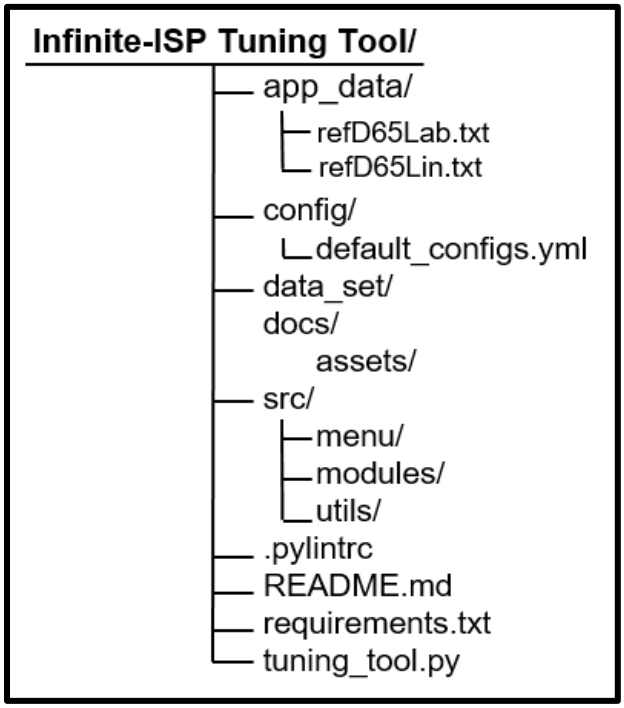


*Figure 2: Directory description*

Here's a description of each folder and file:

1. The folder ‘app_data’ contains essential reference files: refD65Lab and refD65Lin. These files are necessary for the CCM algorithm to determine the correct color correction matrix.
2. The 'config' folder serves as the default location for default_configs.yml file similar to the configuration file present in the Infinite-ISP_ReferenceModel. The default_configs.yml file specifies the input parameters and output settings for each module, ensuring smooth integration between the Tuning Tool and the Infinite-ISP_ReferenceModel. The tuning tool generates a copy of this file (configs.yml), granting users the ability to customize parameters as needed.
3. This folder contains sample images to test the tuning tool. The sample raw images must follow the following naming format.
   **File name format: Name_WxH_Nbits_Bayer.raw**
   For example: ColorChecker_2592x1536_12bits_RGGB.raw
4. tuning_tool.py: This file serves as the main entry point to run the tuning tool.

# Tuning Tool Workflow & Description

The Tuning Tool follows a workflow depicted in Figure 3 to load a default config file, making a copy of it, execute each module, implement its algorithm and save the result. Here's a step-by-step guide on how to use the tool effectively. The key modules of this tuning tool are,

- Calibration Tools
  - Black Level Calculation (BLC)
  - White Balance (WB)
  - Color Correction Matrix (CCM)
- Analysis Tools
  - Gamma Curve (GC)
  - Bayer Noise Estimation (BNE)
  - Luminance Noise Estimation (LNE)
- Generate Configuration Files
  - Update Sensor Information
  - Generate configs.yml
  - Generate isp_init.h

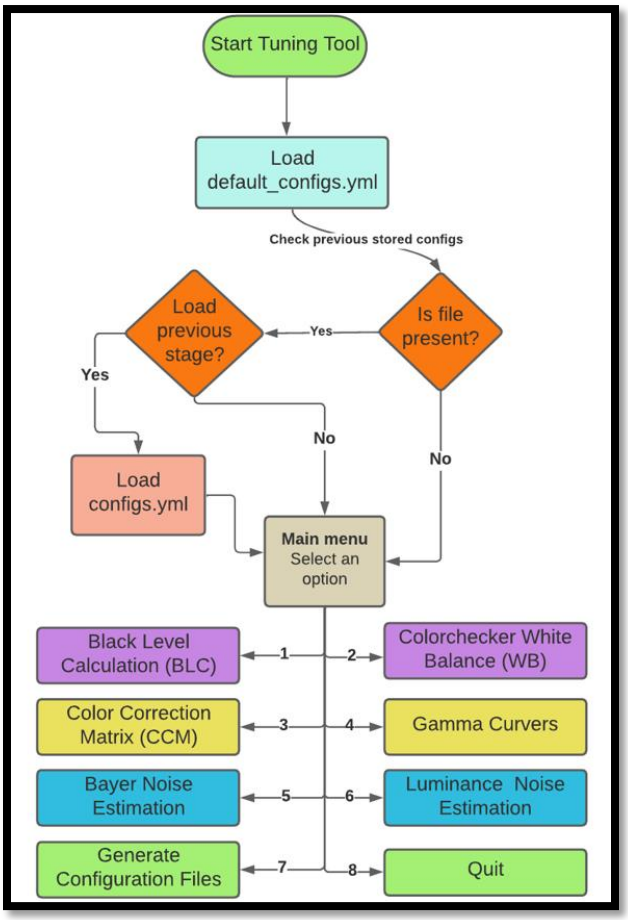


*Figure 3: Workflow for the Infinite-ISP Tuning Tool*

# Getting Started

## Load the Config File

To make use of the default_configs.yml file, it needs to be loaded into the tuning tool associated with the Infinite-ISP_ReferenceModel. Loading the default_configs.yml file to the tuning tool is necessary as this file serves as a central file for configuration parameters and settings that govern the behavior of different modules in the Infinite-ISP_ReferenceModel pipeline.

The default file should not be modified by users. However, a copy of this default file, named configs.yml, will be created in the same folder (Figure 4), providing users with the ability to customize the settings. The contents of the configs.yml file may vary depending on the specific requirements and preferences of the user. However, certain modules within the Infinite-ISP_ReferenceModel pipeline typically require tuning and analysis to achieve desired image processing outcomes. The configs.yml file provides a structured format to define and manage these tuning parameters. It grants users the flexibility to manually adjust these parameters according to their unique needs and requirements.

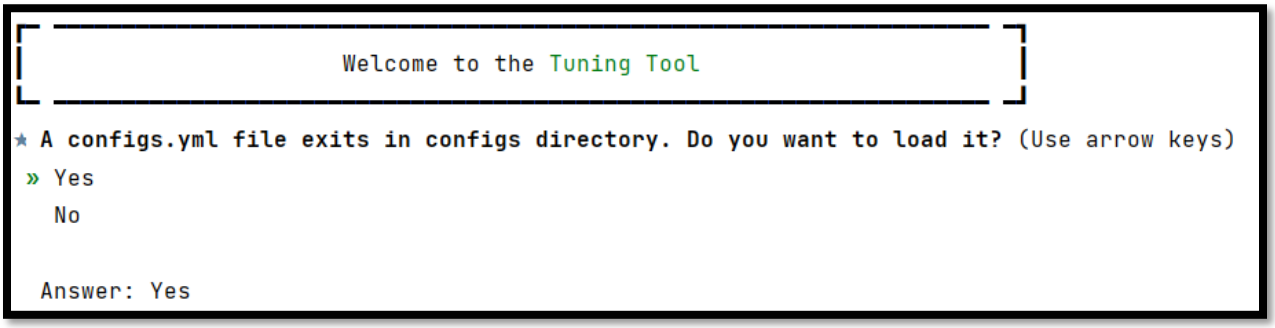


*Figure 4: Asking to load the previous created configs.yml file*

By default, the config folder in the tool's directory is considered the default path for the config file. Ensure that the necessary default config file is present in this location otherwise the tool will not work.

## Main Menu Display

Once the configuration file is selected, the tuning tool will present the following menu to the user as shown in Figure 5. The user can navigate through the menu options by using the up and down arrow keys, and select a desired module by pressing the enter key on the keyboard.

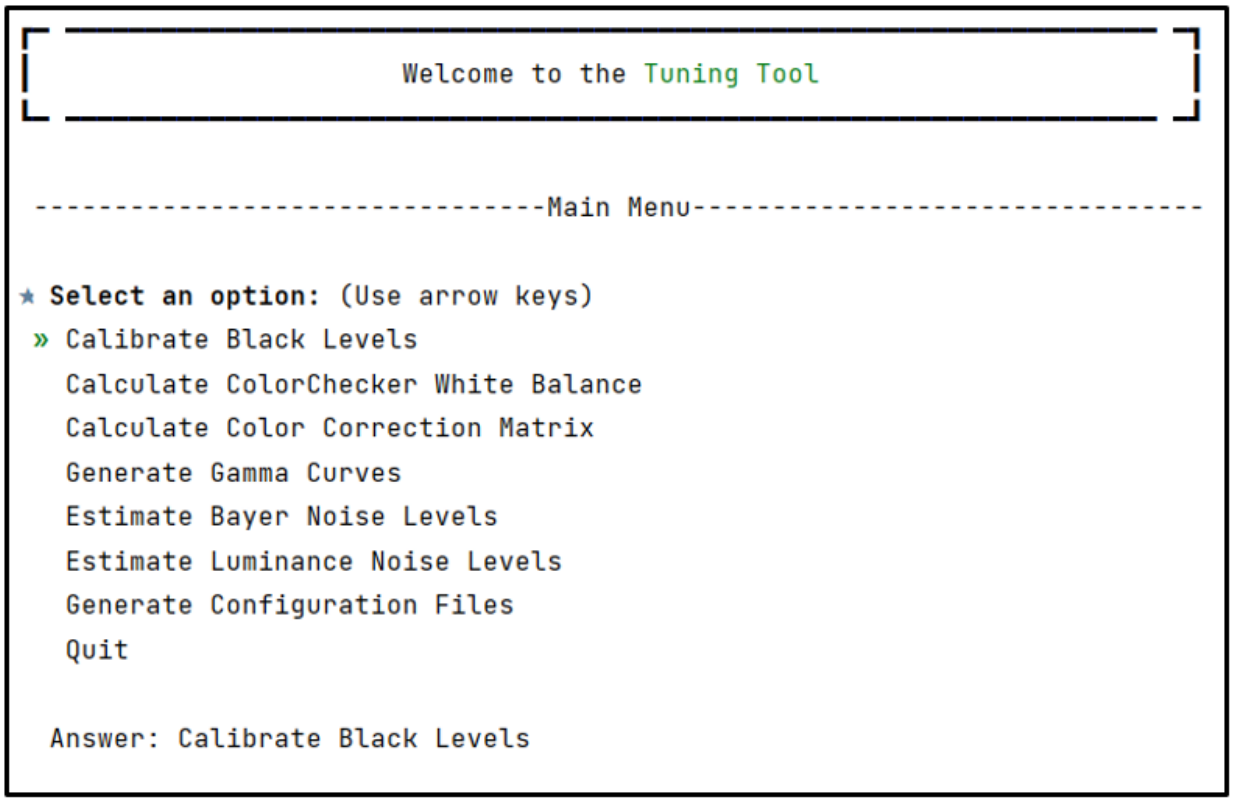


*Figure 5: The Main menu of the Infinite-ISP Tuning Tool*

# Module Workflow & Description

## Common Functionalities

In this section, three essential functions are explained that are common across all modules. These functions provide fundamental capabilities that are frequently utilized in various modules of the tool. Let's take a closer look at each of them.

### Loading an Image

For BLC, WB, CCM, BNE & LNE, loading an image is a must task to run the module.

- BLC: a raw black image for calculation and any raw image for applying black levels.
- WB, CCM, and LNE: a raw or RGB image with ColorChecker.
- BNE: a raw image with ColorChecker.

If the image is not loaded properly, the tool displays an error and will prompt the user to load the file again (Figure 6).

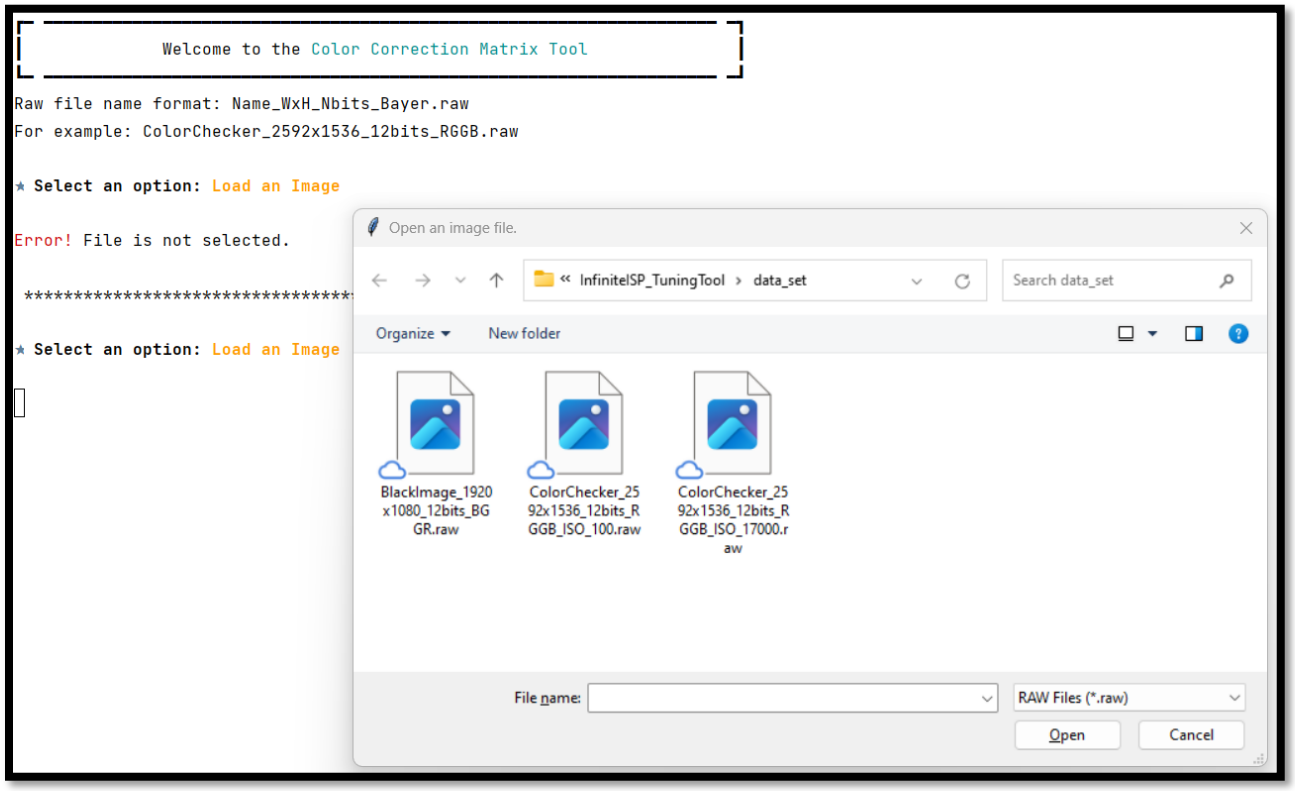


*Figure 6: Loading an image to the module*

### ColorChecker Area Selection

When an image containing a ColorChecker is selected, it will be displayed to the user as depicted in Figure 7.

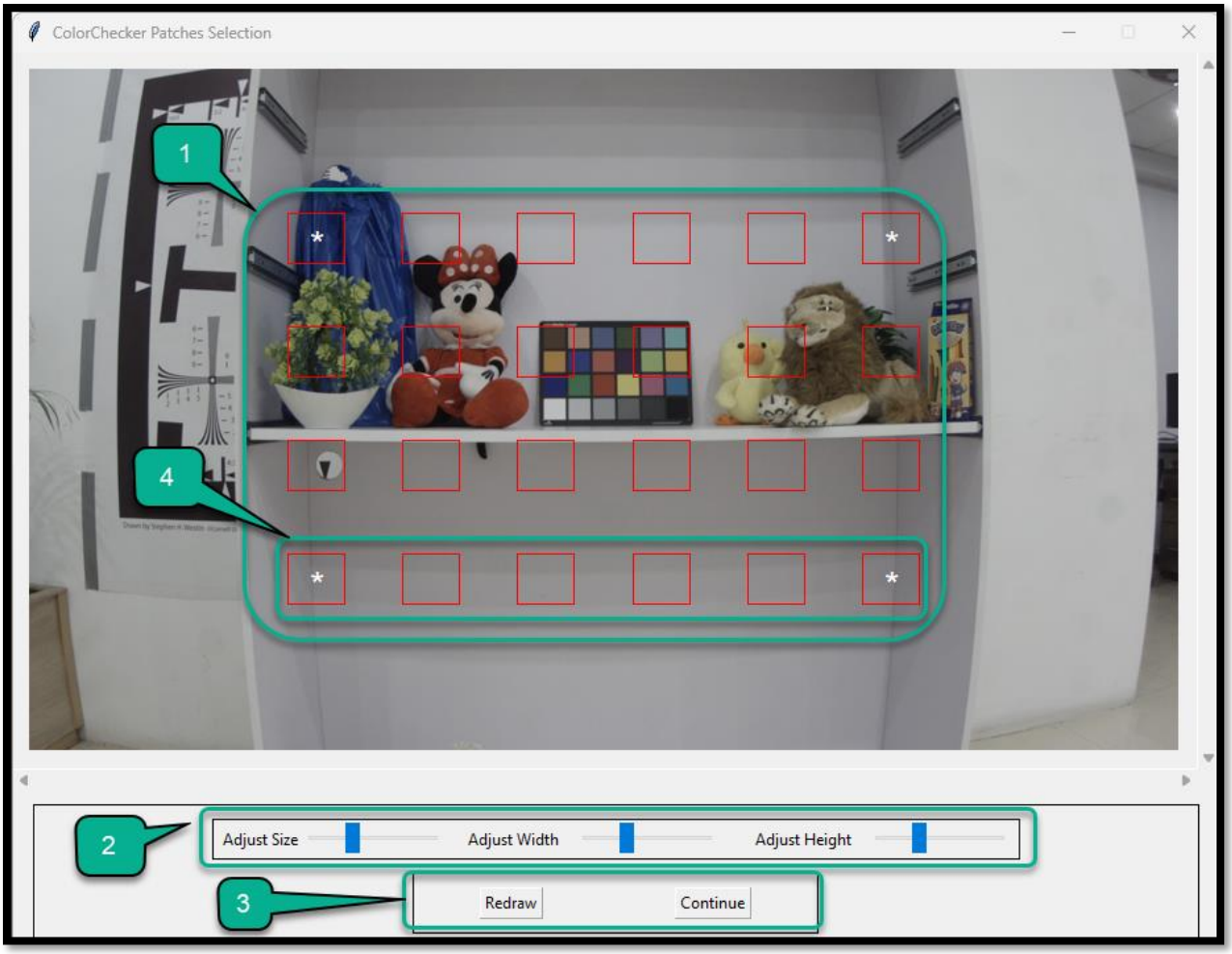


*Figure 7: ColorChecker area selection frame. 1. Drawing for colorchecker patches. Four corner points are marked than can be dragged using mouse to adjust the position of the rectangles. 2. Sliders to adjust size. 3. Buttons to "Redraw" the default ColorChecker rectangles and "Continue" to save patches data. 4. Gray patches (last row of ColorChecker)*

This tool offers following key features:

- **ColorChecker Drawing**: When the image is displayed, a default ColorChecker drawing (1 in Figure 7) is presented. The size of the drawing can be adjusted using sliders located at the bottom of the frame. In addition, using mouse, users can precisely position the drawing by moving any of the four corner points: upper left, upper right, bottom left, and bottom right that are marked in the Figure 8. By moving a corner point, the rectangles will be translated while keeping the other corner points in place. This feature enables users to accurately align the drawing with the ColorChecker area, ensuring precise analysis.


*Figure 8: Moving corner points to adjust the position of the drawing precisely*

- **Zoom In/Out**: Users can zoom in or out of the image using the mouse scroll wheel, and utilize the left and bottom scrollbars to navigate to specific regions of interest within the image. This functionality allows for clear and precise visualization of the ColorChecker present in the image as depicted in Figure 9.

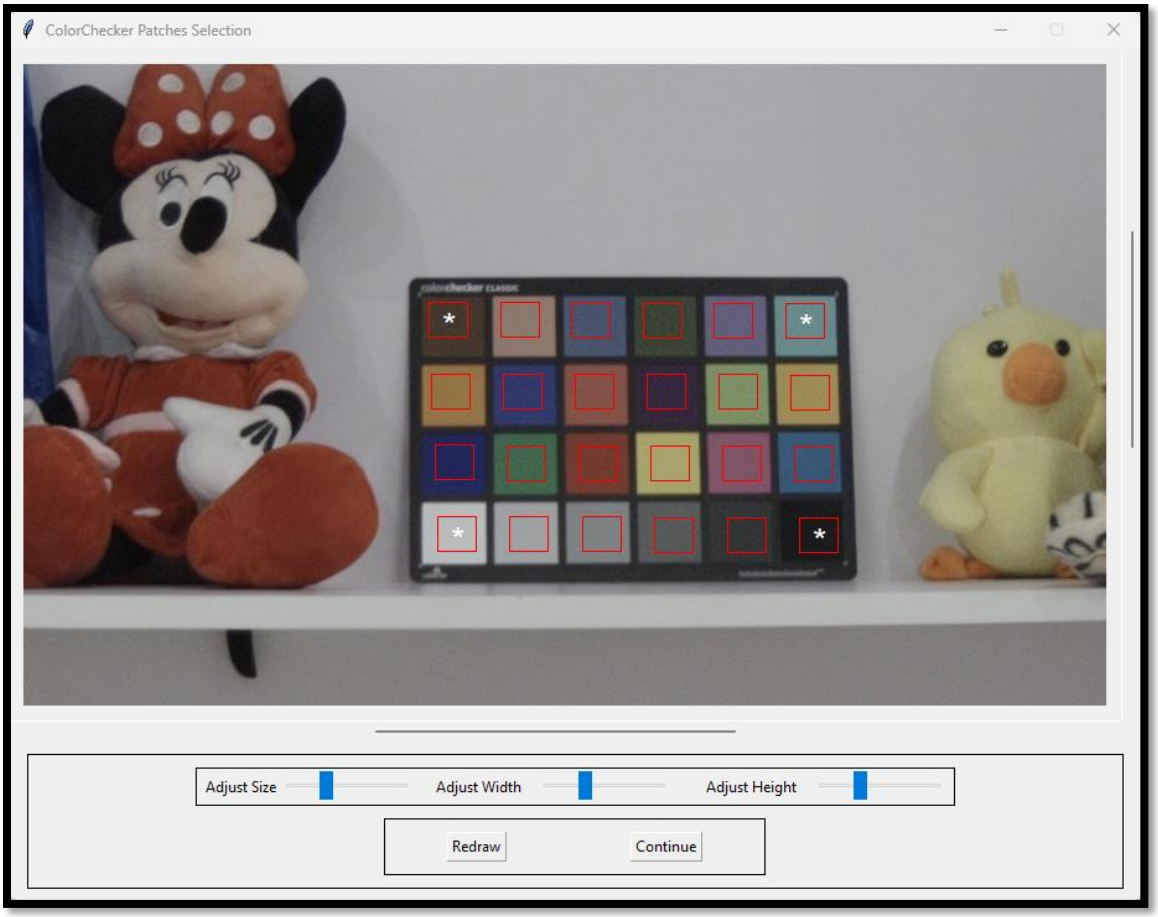


*Figure 9: Zoom in to locate ColorChecker correctly and adjusting drawing on it.*

- **Sliders/Buttons**: The user interface incorporates sliders and buttons to simplify adjustments. By utilizing the Adjust Size, Height, and Width sliders (2 in Figure 7), users can modify the overall size of the rectangles drawn on the ColorChecker. In case the window is resized due to zooming in or out, the “Redraw” button allows users to restore the default drawing. Pressing the "Continue" button will execute the algorithm flow on the image, applying the necessary operations and calculations (3 in Figure 7).

Note that, the last six patches on the ColorChecker are knows as grayscale patches as shown 4 in Figure 7, are used in the white balance and noise estimation modules.

### Saving the Data

Users have the flexibility to save their data in three different ways, based on their preferences:

1. Save to Configuration File: Upon completing a module, users have the option to save the data directly to the configuration file. The "configs.yml" file is located in the default config folder. This file can be easily modified by the user to meet specific requirements.
2. Save to CSV File: By clicking on the "Save" button provided in the graphical user interface (GUI), users can save the data to a CSV file. A convenient dialogue box will appear, allowing users to choose their preferred location for saving the file.

3. Save as PNG Image: Users also have the capability to save the data as a PNG image by utilizing the "Save Images" button on the respective panel.

When the user chooses to save the data, the tool will display a dialogue box, enabling them to select the desired location where the data should be saved. After saving the file, the path of the saved file is shown on the console and end menu reappear.

# Black Level Calibration

## Fundamental Concept

Black level calibration (BLC) is a technique used to compensate for the inherent offset present in sensors, where the 0 value of a pixel does not accurately represent a complete absence of light. This offset can lead to non-zero values being outputted even in the absence of light. BLC corrects this sensor limitation by adjusting the output to align the 0 value with true black. By applying BLC, accurate representation of black or dark areas in an image is achieved, ensuring reliable rendering in various lighting conditions.

## Input Requirements

To calibrate the black levels of a sensor, it is necessary to obtain a uniform black image directly from the sensor. The ideal scenario is capturing this image in complete absence of light. To do this, an effective approach involves capturing the image with the lens left uncovered. This approach allows for establishing the minimum or "black level" of brightness that should be present in any image captured by the sensor or processed by an ISP. For applying black levels raw image is needed.

## Calibration Procedure

The BLC module in tuning tool serves two purposes either to apply or to calculate the black levels.

Users have two options for applying black levels: they can either apply them directly from the configuration file or calculate them first and then apply them to the image. To apply black levels directly from the configuration file, users can choose the "Apply Black Levels" option from the BLC menu. During this process, users will be prompted to specify whether they want to apply linearization or not. If linearization is desired, the saturation values from the configuration files will be utilized for this purpose.

To calculate the Black Levels, follow these steps:

**Step 1:** Load a black image.

**Step 2:** Upon selecting an image, the BLC algorithm executes and calculates black levels for each bayer channel separately and displays them levels as R, Gr, Gb, and B on the console. Now users can either apply these calculated black levels on the desired image and saving the calibrated raw image upon their desired location or they can save these results to the configuration file.

## Results Display/Saving

The calculated black levels, represented as R, Gr, Gb, and B, are then displayed on the console as shown in Figure 10. Following this, an end menu is presented, offering users the option to save the results in a configuration file or apply them to another image.

```
★ Select an option: Calculate Black Levels

★ Select an option: Load an Image


 -------------------------Selected raw image info-------------------------

Width  =  1920
Height =  1080
Bits   =  12
Bayer  =  BGGR


 **************************Calculation Started**************************


 -------------------------Calibrated black levels-------------------------

R Channel  =  3
Gr Channel =  2
Gb Channel =  3
B Channel  =  2


 **************************Calculation Ended**************************

★ Select an option: (Use arrow keys)
 » Save config.yml with Calculated Black Levels
   Apply Calculated Black Levels
   Restart the Black Level Calculation Tool
   Return to the Main Menu
   Quit

  Answer: Save config.yml with Calculated Black Levels
```

*Figure 10: BLC results*

# ColorChecker White Balance Calculation

## Fundamental Concept

White balance gains (WB) are adjustments that can be made to the red, green, and blue channels in an image. These adjustments are used to ensure that colors appear natural and accurate. By adjusting the gains, we can make sure that white objects in the image look truly white, without any unwanted color tints. The goal is to achieve color fidelity and reproduce colors as faithfully as possible.

## Input Requirements

For white balance module, user need to ensure that the selected image (either raw or RGB) contains a ColorChecker, as it is required for the accurate calibration.

## Calibration Procedure

**Step1:** Load the ColorChecker image.

**Step2:** After selecting an image, a panel will open, providing a view of the image as previously shown. Now locate the ColorChecker, as described in the previous section. Fine-tune the position and size of the ColorChecker within the panel. Once the desired alignment is achieved, proceed by pressing the "Continue" button, which initiates the White Balance (WB) algorithm for further processing.

The gray patches present in the last row of the ColorChecker are used to calibrate the wb gains.

## Results Display/Saving

Once the algorithm is executed, the white balance gains will be shown on the console (Figure 11). At this point, the user has several options available. They can choose to save the gains in a configuration file for future use. Alternatively, they can opt to view the results by applying the gains to the input image (Figure 12). Additionally, the user has the ability to save the resulting images for further reference or analysis.

```
Raw file name format: Name_WxH_Nbits_Bayer.raw
For example: ColorChecker_2592x1536_12bits_RGGB.raw

★ Select an option: Load an Image


 ------------------------Selected raw image info------------------------

Width  =  2592
Height =  1536
Bits   =  12
Bayer  =  RGGB


 ----------------------------Calculated Gains----------------------------

R gain =  1.306
B gain =  3.0529

 ************************************************************************

★ Select an option: Apply White Balance on the Input Image


 ************************************************************************

★ Select an option: (Use arrow keys)
 » Apply White Balance on the Input Image
   Save config.yml with the Calculated WB Gains
   Restart the ColorChecker White Balance Tool
   Return to the Main Menu
   Quit

  Answer: Apply White Balance on the Input Image
```

*Figure 11: White balance gains results*


*Figure 12: Calibrated WB gains comparison on input image*

# Color Correction Matrix Calculation

## Fundamental Concept

Calculating the Color Correction Matrix (CCM) is an important module in the tuning tool for an Image Signal Processor (ISP). Color correction matrix (CCM) enables precise color adjustments by manipulating the transformation matrix that maps the camera's captured colors to the desired output colors.

The Color Correction Matrix is typically a 3x3 matrix that defines the transformation from the camera's captured colors to the target colors. Each element in the matrix represents the contribution of the corresponding color channel (Red, Green, and Blue) to the final output color. The CCM is used to correct color imbalances, adjust saturation levels, and achieve accurate color representation.

## Input Requirements

For CCM module, user need to ensure that the selected raw image or an RGB image (.png, .jpg or .jpeg) contains a color checker, as it is required for the accurate calibration.

## Calibration Procedure

**Step1:** Load the ColorChecker image.

**Step2:** After selecting an image, a panel will open, providing a view of the image as previously shown. Now locate the ColorChecker, as described in the previous section. Fine-tune the position and size of the ColorChecker within the panel. Once the desired alignment is achieved, proceed by pressing the "Continue" button, which initiates the color matrix calibration process.

**Step2:** After ColorChecker area selection, the user will be prompted to decide whether to apply white balance in the image or not.

**Step4:** The user will be asked to choose between provided Error Matrix $\Delta E_{ab}00$ and $\Delta C_{ab}00$ . The reference files for both the algorithms are placed in the AppData folder.

**Step5:** Then, the user will be prompted to decide whether to maintain white balance in the image or not.

Depending on the user's choice, the corresponding algorithm will be executed. The color correction matrix is designed to minimize the $\Delta E_{ab}$ or $\Delta C_{ab}$ values between the original and corrected colors. It is common for color correction matrices to follow the constraint of the sum of each row equaling one. This constraint ensures that the color correction matrix maintains color balance and doesn't introduce any unintended color shifts during the correction process.

## Results Display/Saving

After algorithm executes, the corrected red, green, and blue rows will be displayed on the console. The calibrated CCM will be applied on the input image and is displayed as Figure 14. Two type of matrices will be displayed as depicted in Figure 13, one is in floating-point notation and the other one is the integer type for make it hardware friendly.

After displaying corrected CCM on the console, an end menu will appear, offering user the options to save results.

```
 ------------------------------------------------------------------------

★ Apply White Balance? Yes

★ Error Matrix? ΔC 00

ΔC 00 is selected.

★ Maintain White Balance?  Yes

White balance is maintained.

 --------------------------Algorithm is running---------------------------


 *********************************Result************************************

Calculated Color Correction Matrix with following parameters:
1. Error matrix: Delta Cab
2. White balance: Maintained

 -------------------------------Integer CCM--------------------------------

Corrected red   =  [2116, -988, -104]
Corrected green =  [-393, 1533, -117]
Corrected blue  =  [258, -1084, 1850]

 --------------------------Floating-point CCM----------------------------

Corrected red   =  [ 2.0668 -0.9652 -0.1016]
Corrected green =  [-0.3835  1.4974 -0.1139]
Corrected blue  =  [ 0.252  -1.0582  1.8062]
```

*Figure 13: CCM results*


*Figure 14: Calibrated CCM comparison on input image*

# Gamma Analysis

The Gamma module in the Tuning Tool allows user to visualize the gamma curves.

## Input Requirements

For gamma module, the user-defined gamma curve is loaded from the loaded configuration file and is compared with the usual gamma created at $\gamma = 2.2$.

## Analysis Procedure

Choosing compare gamma curves from Gamma module menu will display the plots for two types of gamma curves: user-defined and the common gamma curve generated $\gamma = 2.2$ that can be seen here in Figure 15.

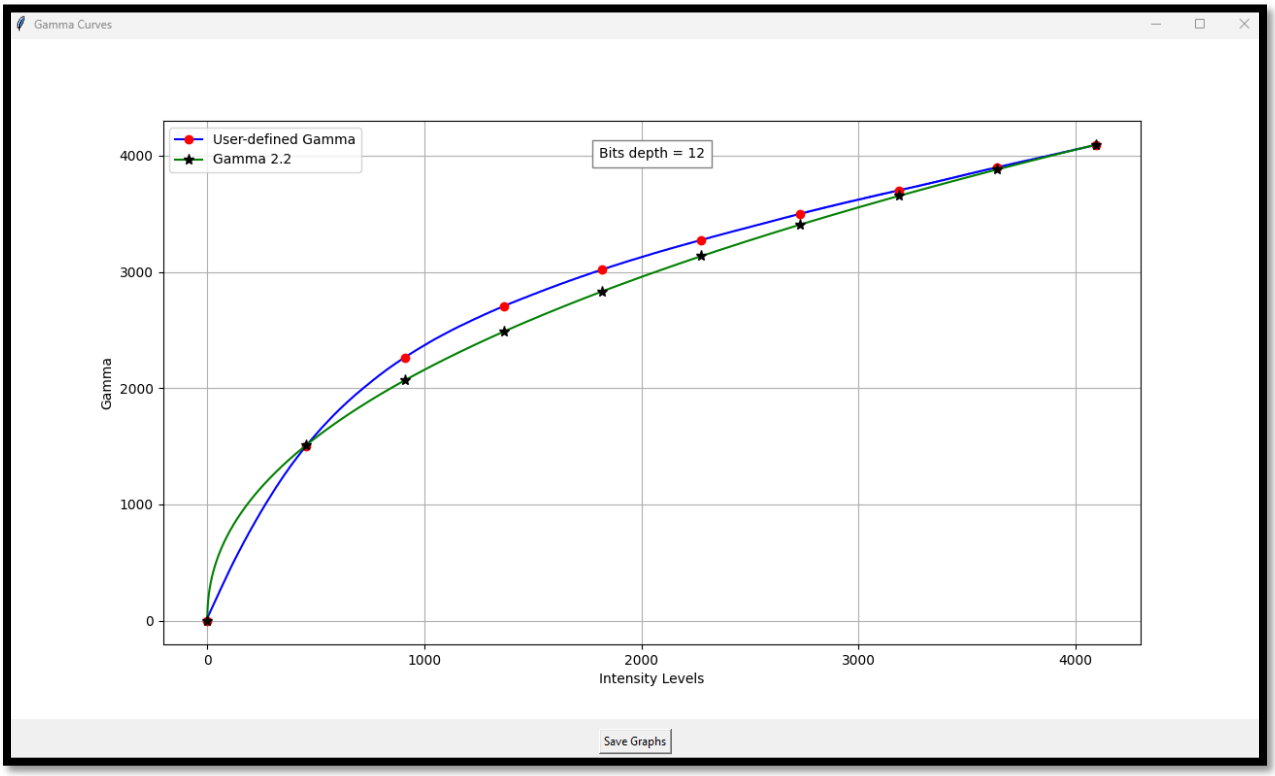


*Figure 15: Gamma comparison curves*

Within the panel displaying the gamma plots, users will see a "Save" button, allowing them to save the results if they wish to do so.

# Bayer Noise Estimation

## Fundamental Concept

The Bayer Noise Estimation module is designed to assist users in estimating noise levels in a Bayer pattern by analyzing raw images and locating a ColorChecker. The Bayer image, composed of the red (R), green (G), and blue (B) channels, is then split into these individual channels. By leveraging gray patches within each channel, the algorithm performs noise estimation. Since the gray patches ideally contain uniform color values, any deviations from uniformity can be attributed to noise.

By following the steps outlined in this guide, user can accurately measure the Bayer noise in the image and obtain valuable information for further tuning and processing.

## Input Requirements

For BNE module, user need to ensure that the selected image should be raw that contains a color checker.

## Analysis Procedure

The algorithm is initiated when the patches from the ColorChecker image are chosen, triggering the estimation of noise levels in the bayer domain. The bayer image is then divided into three bayer channels: R, G (Gr and Gb), and B. By utilizing gray patches within these channels, the algorithm proceeds to calculate the standard deviation of each of the gray patch present in the ColorChecker.

## Results Display/Saving

A GUI will appear, showing a table with the standard deviations of last 6 grey patches (as mentioned as 2 in Figure 7) from each R, G (Gr & Gb), and B raw channels. This is clearly shown in Figure 16 below.

If desired, click the "Save" button on the GUI to save the table data to a CSV file at his preferred location. Cancel to proceed further.

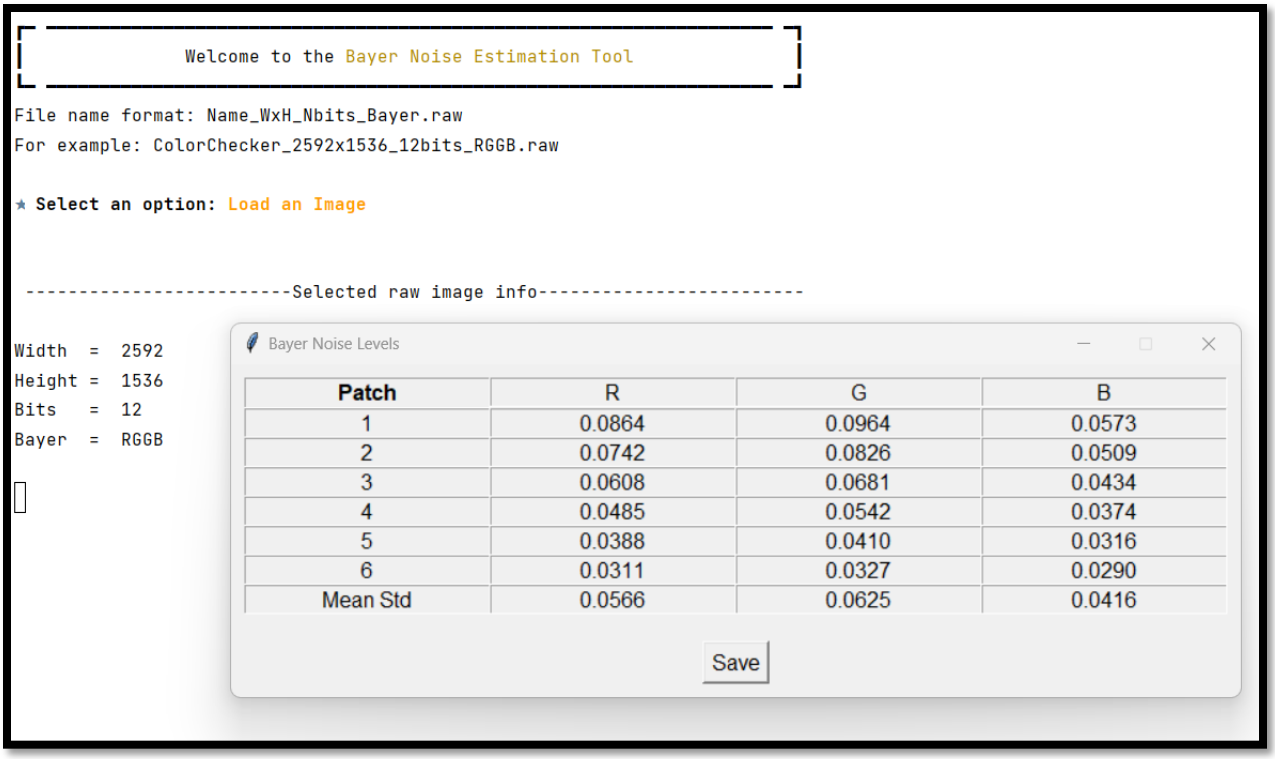


*Figure 16: BNE results*

# Luminance Noise Estimation

## Fundamental Concept

The Luminance Noise Estimation module is designed to assist users in estimating luminance noise levels in an RGB image by locating a ColorChecker. The analysis involves evaluating the statistical characteristics of the pixel values within the gray patches. By examining the variances or standard deviations of these pixel values, the algorithm gains insights into the degree of noise affecting the luminance channel. By following the steps outlined in this guide, user can accurately measure the noise in the image and obtain valuable information for further tuning and processing.

## Input Requirements

User can select raw image or an RGB image (.png, .jpg or .jpeg) that must contain a ColorChecker for accurate analysis.

## Analysis Procedure

On loading the image and after carefully choosing the ColorChecker patches users will be prompted with an option to apply white balance on the selected image. The algorithm will then be executed to calculate the standard deviations of each of the gray patch (last row of ColorChecker) present in the ColorChecker.

## Results Display/Saving

A GUI will appear, showing a table with the standard deviations of last 6 grey patches from the luminance channel of the image along with the patches images as shown in Figure 17.

If desired, click the "Save" button on the GUI to save the table data to a CSV file at user's preferred location. Cancel to proceed further.

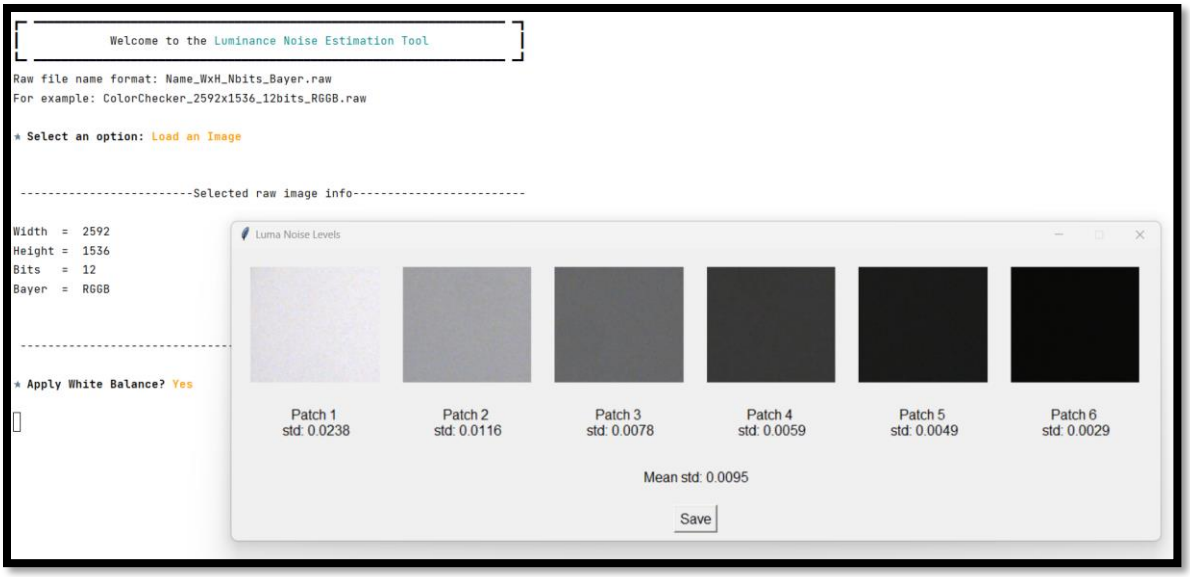


*Figure 17: LNE results*

# Generate Configuration Files

The menu for this feature of the tuning tool is displayed in Figure 18. There are two types of files generated by the Infinite-ISP Tuning Tool.

1. configs.yml
2. isp_init.h

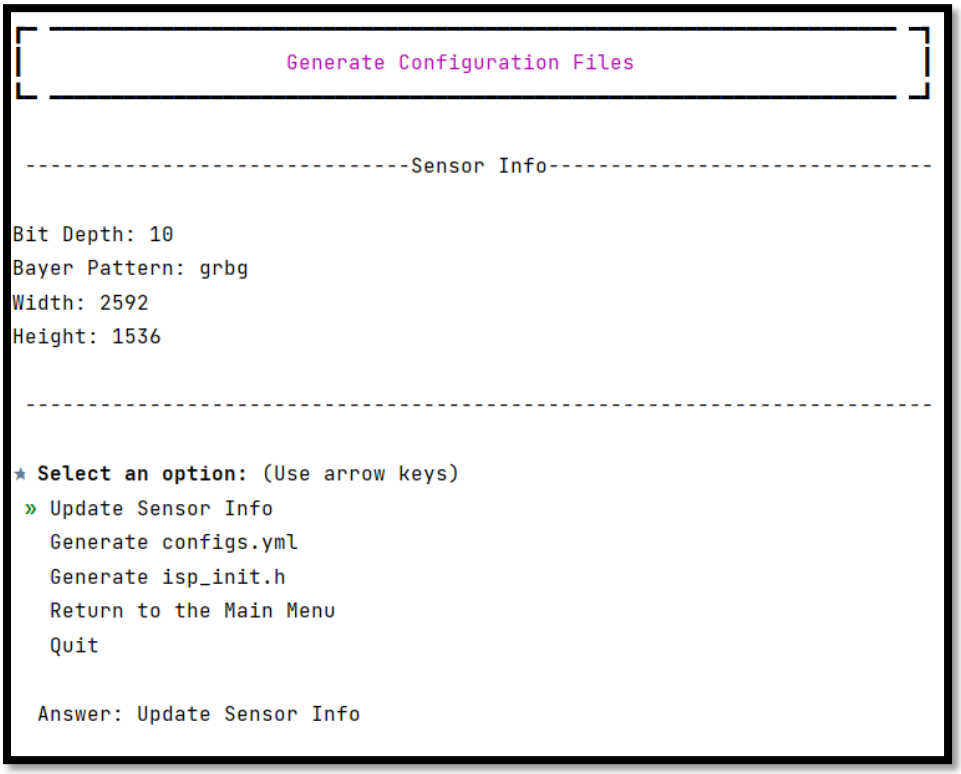


*Figure 18: Generate configuration files menu*

## File configs.yml

The tuning tool offers users the ability to generate a configuration file called configs.yml. This file is similar to the Infinite-ISP_ReferenceModel pipeline configuration. After completing the calibration and analysis process, users can save the data to the initial configuration file that was created at the start of the tuning tool. The configs.yml file serves as a valuable resource for further customization and fine-tuning.

## File isp_init.h

In addition to the configs.yml file, the tuning tool allows users to generate an isp_init.h file. This file serves as a baseline start configuration specifically designed for FPGA models. It provides a solid foundation for FPGA implementation. By utilizing the data from the configs.yml file, the isp_init.h file can be generated, ensuring compatibility and efficient utilization of the FPGA resources.

## Updating Sensor Info

Before generating these files, users have the option to update the sensor information. This functionality allows users to modify parameters such as the bit depth, Bayer pattern, height, and width (Figure 19).

Once the sensor info is updated, users can proceed to generate the configuration file of their choice. This process takes into account the modified sensor info and user-defined configurations, resulting in a tailored configuration file that aligns with their specific needs.

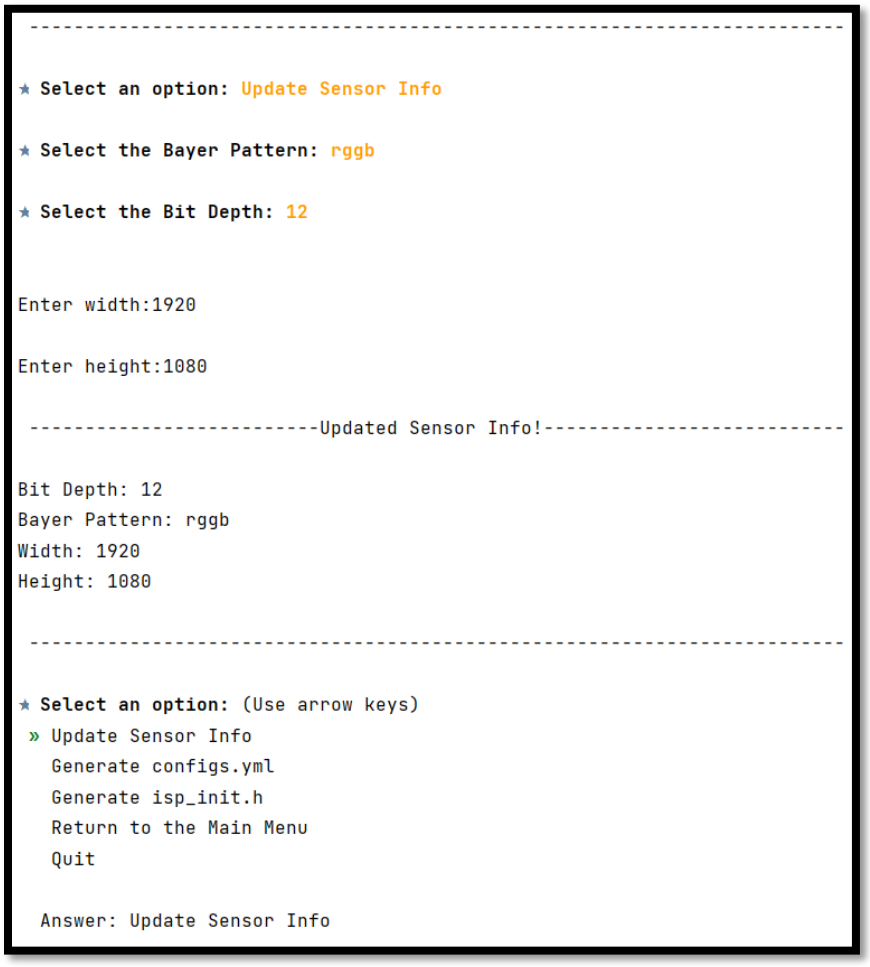


*Figure 19: Updating sensor info*