infinite-isp

Algorithms Description

v1.0

Contents

List of Figures

List of Tables

Introduction

infinite-isp 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. infinite-ISP offers simple to complex algorithms at a modular level to stimulate the whole pipeline with hardware-based tuning functions.

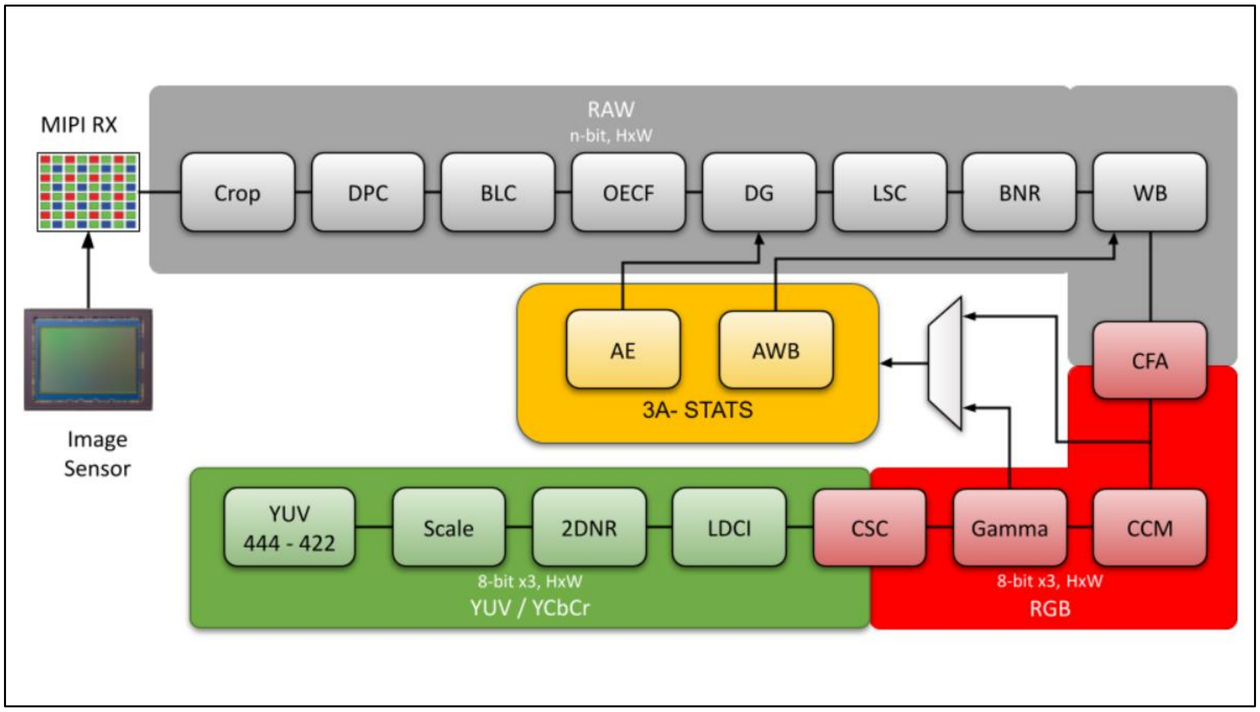


Figure 1: infinite-ISP architecture.

Motivation

The real motivation behind infinite-isp is to streamline procedures, enhance current approaches, and advance algorithms in the specialized ISP and computational photography field. This project provides a platform for relevant developers to work on the most exciting and challenging hardware-related issues that an ISP engineer encounters. The "open innovation" approach, which enables anyone with the right skills, time, and interest to contribute generously to getting the ISP framework executed from the software to the hardware level, is the key upon which infinite-isp seeks to achieve success.

Modules Description

The infinite-isp pipeline consists of the following modules,

- Crop,
- Dead pixel correction (DPC),
- Black level compensation (BLC),
- Optoelectronic conversion function (OECF),
- Auto white balance gain control (AWB),
- Auto exposure control (AE),
- Digital gain,
- Color filter array interpolation (CFA),
- Gamma correction (GC),
- Color correction matrix (CCM),
- Bayer noise reduction (BNR),
- Color space conversion (CSC)),
- Local dynamic contrast improvement (LDCI),
- 2d noise reduction (2DNR),
- Scale and YUV format (4:4:4 & 4:2:2).

Some functions like Lens shading correction (LSC), HDR stitching (W/HDR), tone mapping (TM) and sharpening or edge enhancement (EE) will be implemented furthermore.

Dead Pixel Correction

Introduction

Image sensors sometimes come with manufacturing flaws that result in some dead pixels in the pictures. Despite an advanced manufacturing technique, photo sensor arrays like CCD and CMOS arrays sometimes include a few defective pixels because of noise, dust, or fabrication flaws. So, identifying the correct dead pixels in the images and then replacing them with the corrected values are two important phases of the DPC module.

Defective pixels can be categorized into three groups based on their defect values:

1. Dead (consistently low)
2. Hot (consistently high)
3. Stuck (always a value between low and high).

Defective pixels are either static dead pixels (constant) or dynamic dead pixels, depending on their behavior that gets changed by some conditions like exposure or temperature.

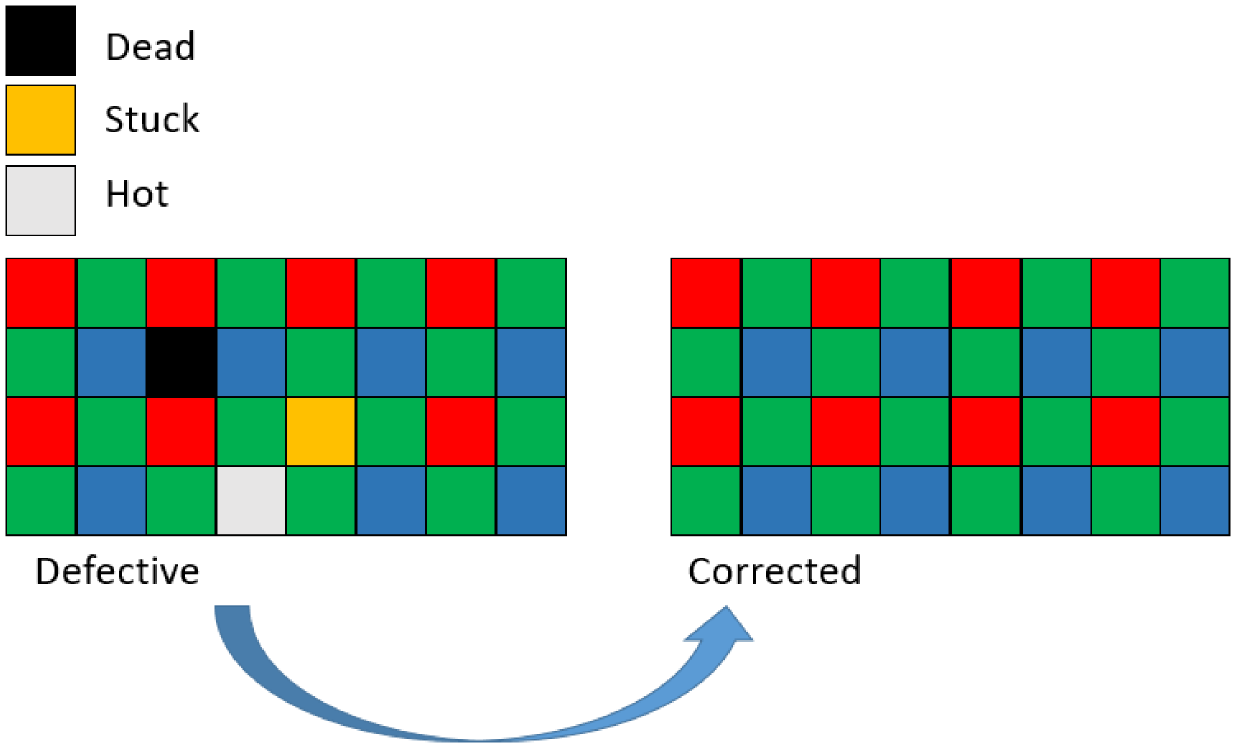


Figure 2:Defective and corrected pixels

Algorithm Explanation

Each pixel of the image is tested while the DPC algorithm runs. If a particular threshold is crossed, potential checks for each pixel ensure the detection of damaged pixels. To find defective pixels, take a 3×3 neighborhood of the same color channel surrounding each pixel being tested in a 5×5 window. A pixel is considered defective if it passes through the following checks,

1. Pixel value should be less than the minimum pixel value in the 3×3 neighborhood or greater than the maximum pixel value in the 3×3 neighborhood.
2. Difference between the pixel under test and each one of the neighboring pixels must be greater than the specified threshold.

If the pixel is identified as defective, the other phase of the module will replace these corrupted pixels with the corrected values by computing gradients with horizontal, vertical, left diagonal, and right diagonal. The corrected value is computed as the average of the two neighbors in the minimum gradient direction.

Configuration Parameters

Parameters	Details
$isEnable$	When enabled, apply the DPC algorithm. False: Disable True: Enable
$isDebug$	Flag to output module debugs logs.
$dp\_threshold$	The threshold for tuning the DPC module. The lower the threshold more are the chances of pixels being detected as dead and hence corrected.

Table 1: DPC configuration parameters

Black Level Correction

Introduction

This module is responsible for setting the image's pure black color. According to the sensor's characterization and layout, the black level voltage, which cannot be regarded as 0, is its lowest output voltage. Setting the exposure time and every gain (analog and digital) to its minimum values is one technique to create a pure black image; however, even under these conditions, the image won't be entirely black; instead, the black level voltage needs to be rectified.

Algorithm Explanation

To rectify the black level voltage, subtract the respective offsets from the channels as below. For RGGB Bayer,

$$R' = R - R_{offset}$$

$$Gr' = Gr - Gr_{offset}$$

$$Gb' = Gb - Gb_{\ offset}$$

$$B' = B - B_{offset}$$

Configuration Parameters

Parameters	Details
$isEnable$	When enabled, apply the BLC. False: Disable True: Enable
r_offset	Red channel offset.
gr_offset	Gr channel offset.
gb_offset	Gb channel offset.
b_offset	Blue channel offset.
$isLinear$	Enables or disables linearization. When enabled, the BLC offset maps to zero, and saturation maps to the highest possible bit range given by the user.
r_sat	The red channel saturation level
gr_sat	Gr channel saturation level
gb_sat	Gb channel saturation level
b_sat	Blue channel saturation level

Table 2: BLC configuration parameters

Opto-Electronic Conversion Function—OECF

Introduction

The relationship between the camera output (photocell voltages) and incident light lux is referred to as the optoelectronic conversion function (OECF). The sensor usually has an approximately linear response in that the detected signal is proportional to the scene exposure as imaged by the lens.

The camera should be set up for sensors with poor linear response to achieve a linear relationship between the digital output and the measured luminance. Linearity is essential for achieving good white balance and color accuracy. The OECF module implemented lookup curves for sensor signal re-mapping. There are four curves and one for each color channel.

Algorithm Explanation

The OECF module corrects the nonlinearity of the sensor response. To fix this nonlinearity, there are standards. ISO 14524 defines a standard procedure for this purpose. It uses a digital reflective camera contrast chart to plot density response vs. pixel values. Ideally, this response should be linear but when it's not OECF module chimes in and maintains the linearity.

Configuration Parameters

Parameters	Details
$isEnable$	When enabled, applies the OECF curve. False: Disable True: Enable
r_lut	The look-up table for the OECF curve. This curve is mostly sensor dependent and is found by calibration using some standard technique.

Table 3: OECF configuration parameters

Lens Shading Correction

To be implemented.

Bayer Noise Reduction

Introduction

The Bayer Noise Reduction (BNR) block suppresses noise in the Bayer domain. As a noise reduction block in the ISP pipeline, it is primarily tasked with reducing noise without effectively blurring the image details before demosaicing.

Algorithm Explanation

For the BNR block, a Joint/Cross Bilateral Filter (JBF) has been selected as the appropriate candidate for RAW image denoising.

1. Green channel interpolation is performed on the input Bayer image to obtain a complete image of the green pixels using green interpolation kernels from Malwar He Cutler.
2. Input Bayer image is deconstructed into R and B sub-images of size ½ compared to the original Bayer image.
3. Interpolated G image is deconstructed into G-at-R-locations and G-at-B-locations sub-images to serve as guide images for R and B sub-images, respectively.
4. JBF is applied on the R and B sub-images using their respective guide images, and JBF is applied on the interpolated G image using itself as the guide image.
5. The output Bayer image is reconstructed by joining the R, G, and B pixels from the JBF outputs of the respective color images.

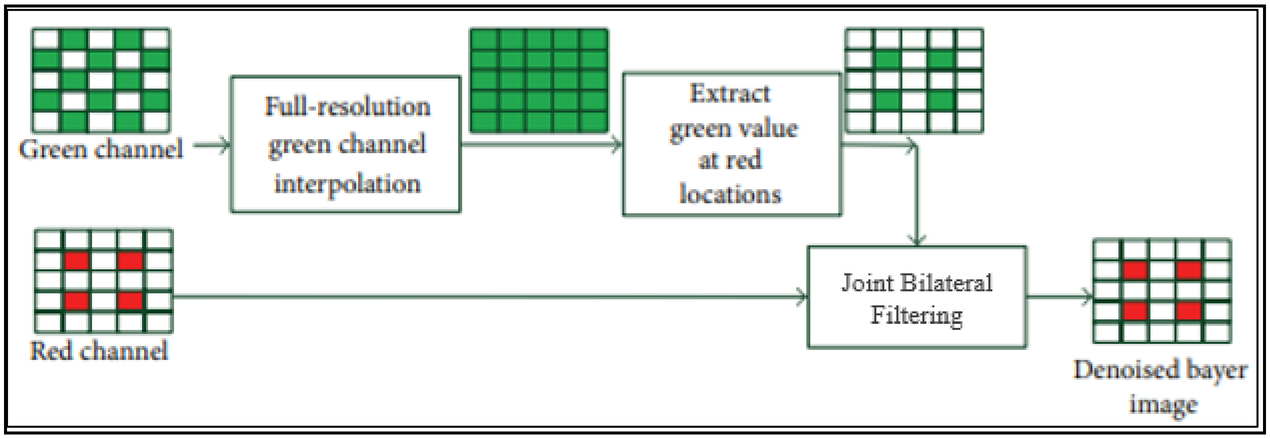

Green channel
Full-resolution green channel interpolation
Extract green value at red locations
Red channel
Joint Bilateral Filtering
Denoised bayer image

Figure 3: BNR algorithm

Configuration Parameters

Parameters	Details
$isEnable$	When enabled, applies the BNR algorithm. False: Disable True: Enable
$file\_window$	It should be an odd window size.
$r\_stdDevS$	Red channel Gaussian kernel strength. The more strength the, more the blurring. It cannot be zero.
$r\_stdDevR$	Red channel range kernel strength. The more the strength, the more the edges are preserved. Cannot be zero.
$g\_stdDevS$	Gr and Gb Gaussian kernel strength.
$g\_stdDevR$	Gr and Gb range kernel strength.
$b\_stdDevS$	Blue channel Gaussian kernel strength.
$b\_stdDevR$	Blue channel range kernel strength.

Table 4: BNR configuration parameters

White Balance

Introduction

This module applies the white balance gains provided by the user. If gains are unavailable, then the auto-white balance should be used.

Configuration Parameters

Parameters	Details
$isEnable$	Applies user-given white balance gains when enabled.
$isAuto$	When true enables the 3A - AWB and doesn’t use the user-given WB gains.
r_gain	Red channel gain.
b_gain	Blue channel gain.

Table 5: WB configuration parameters

Digital Gain

Introduction

Digital gain works with the auto exposure module to choose an appropriate constant value to be multiplied by the channels.

Algorithm Explanation

The algorithm will be better understood in the auto-exposure module.

Configuration Parameters

Parameters	Details
$isEnable$	This is an essential module and cannot be disabled.
$isDebug$	Flag to output module debug logs.
$gain\_array$	Gains array. User can select any of the gains listed here. This module works together with the AE module.
$current\_gain$	Index for the current gain starting from zero.

Table 6: DG configuration parameters

3A STATS

3A STATS module refers to the three important As in the pipeline: Auto White Balance, Auto Focus, and Auto Exposure.

Auto White Balance

Introduction

AWB calculates gains for white balance using a different algorithm selected by mentioning the algorithm name in the ISP pipeline config file. AWB module is separate from WB (White balance) module, where gains are applied to a raw image. AWB module resides between demosaicing and color correction matrix (CCM).

Algorithm Explanation

The AWB module corrects the color mixture of an image by applying gains to the image to restore the color of grey pixels. After researching a variety of algorithms, the following algorithms were selected based on their performance and computational complexity.

- Grey World Algorithm
- Norm-2 Grey World
- PCA Illuminant Estimation

Grey World Algorithm

The Gray World is a white balance method that assumes that your scene, on average, is a neutral gray. Gray-world assumption hold if we have a good distribution of colors in the scene. Assuming that we have a good distribution of colors in our scene, the average reflected color is considered the color of the light. Therefore, we can estimate the illumination color cast by looking at the average color and comparing it to gray. The gray world algorithm produces an illumination estimate by computing the mean of each image channel $I = I_R, I_G, I_B$

$$C_{avg} = \frac{\sum_{n=0}^{n} I_c(n)}{N}$$

$C \in \{R, G. B\}$, White balance gains for red and blue channels are calculated, which levels the intensities for all channels for grey pixels.

$$R_{gain} = \frac{G_{avg}}{R_{avg}}$$

$$B_{gain} = \frac{G_{avg}}{B_{avg}}$$

Norm-2 Gray World

Norm-2 grey world works on the same assumption as the grey world but uses L2-norm to calculate channel averages for gain calculations.

$$C_{avg} = \frac{\sum_{n=0}^{n} (I_c(n)^p)^{1/p}}{N}$$

Where $p = 2$ for L2-Norm. The gain calculation is the same as 2 and 3. Norm-2 Grey performs better than grey world white balance, especially in outdoor scenes and images with larger green sections; it produces less color cast.

PCA Illuminant Estimation

PCA illuminant estimation is one of the best conventional algorithms for white balance. It shows that spatial information only provides additional information that can be obtained directly from the color distributions.

The defined algorithm is an efficient illumination estimation method that chooses bright and dark pixels using a projection distance in the color distribution and then applies PCA to estimate the illumination direction, as shown in the figure below.

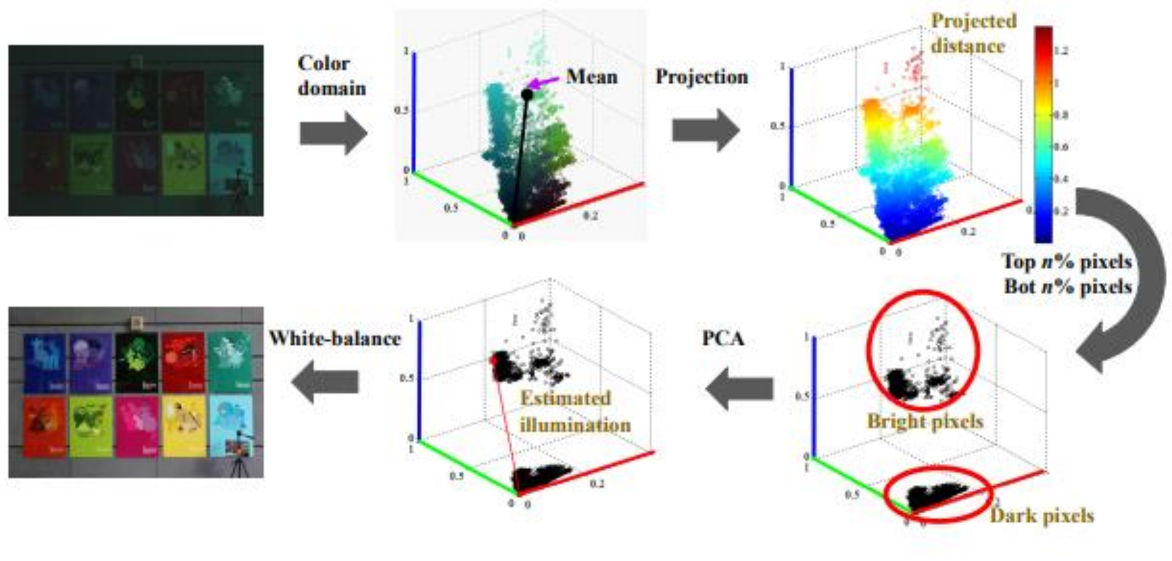


Figure 4: PCA illuminant estimation

This method gives state-of-the-art results on existing general illumination datasets. The drawback of this method is its computational complexity.

Configuration Parameters

Parameters	Details
$algorithm$	Can select one of the following algorithms - $grey\_world$ - $norm\_2$ - pca
$percentage$	[0 - 100] - Parameter to select dark-light pixels percentage for PCA algorithm

Table 7: AWB configuration parameters

Auto Exposure

Introduction

AE is one of the three modules of 3A Control (along with AWB and AF). In consumer digital cameras, one of the image pipeline tasks includes auto-exposure (AE). This module provides the right amount of exposure by adjusting the camera exposure automatically. For auto-adjusting of the camera exposure, we need to make a feedback system that can update the camera exposure according to an evaluation of the current exposure. Therefore, the exposure evaluation method is a key point in the auto-exposure system.

In-camera ISP, AE is implemented as a complete loop with AE-Stats feedback and AE control setting.

AE Controls: these are the sensor's settings that are controlled based on AE feedback. The most commonly used AE controls are:

1. Shutter speed
2. Aperture
3. Gains (Analog, Digital, and ISP Gain)

The high-end camera mainly controls the scene brightness using exposure, aperture, and analog gain. Due to the required frame rate, video cameras have a lower exposure limit.

Also, due to limited sensor control in economical cameras, AE control settings are primarily done through gains (Analog, Digital, and ISP gain that all add up to the scene's ISO).

AE Stats Feedback:
The primary AE stat used to determine scene brightness is the scene's luminance histogram. Histogram collection can be done in various ways:

1. One Histogram for the complete image
2. Divide the window into a grid with one luminance histogram per cell.

3. Divide the frame into sections of unequal size depending on which image section you want to give the highest priority for scene brightness estimation

Algorithm Explanation

The skewness of the luminance histogram around a central luminous gives us an idea about whether the image is underexposed or overexposed.

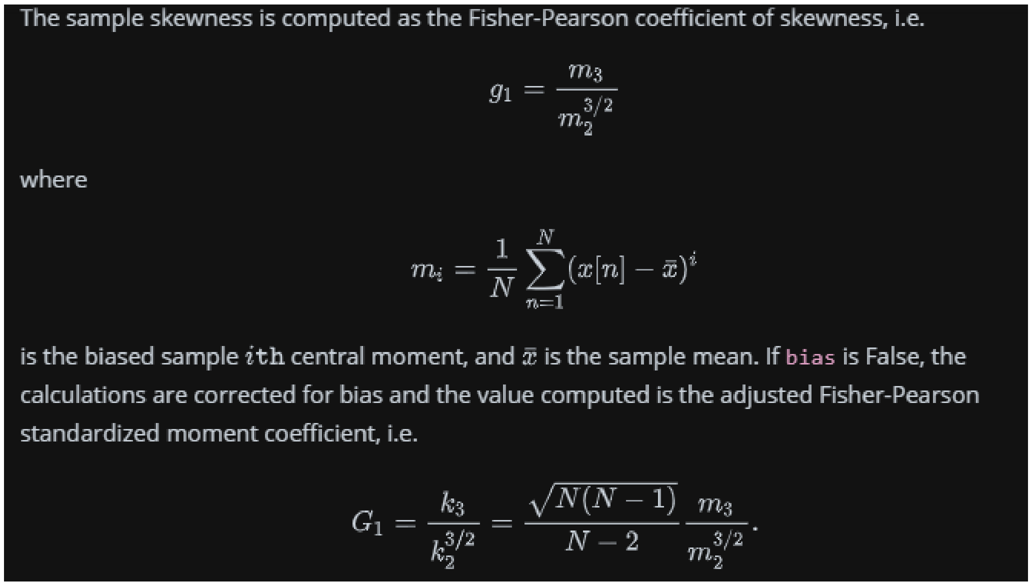
Moments are well known for their application in image analysis since they can derive invariants concerning specific transformation classes. A moment is a specific quantitative measure of the shape of a set of points. If the points represent probability density, then the zeroth moment is the total probability, the first moment is the mean, the second central moment is the variance, and the third moment is the skewness. The nth moment of a real-valued continuous function f(x) of a random variable x about a value C is

$$\mu_n = \int_{-\infty}^{\infty} (x - C)^n f(x) dx$$

The nth moment about zero of a probability density function f(x) is the expected value of X(n) and is called a raw moment. The moments about its mean (with C being the mean) are called central moments; these describe the shape of the function independently of translation. The central moments are usually used for the second and higher moments rather than the moments about zero. A central moment is the expected value of a specified integer power of the deviation of the random variable from the mean instead of from zero; therefore, they provide more precise information about the distribution's shape. The first central moment µ1 is 0. The second central moment µ2 is called the variance. The third central moment is used to define the standardized moment, which is used to describe skewness because it is the measure of the lopsidedness of the distribution. In a symmetric distribution, its third central moment equals zero. A skewed distribution to the left (the tail of the distribution is longer on the left) will have negative skewness. An uneven distribution to the right (the tail of the distribution is longer on the right) will have positive skewness.

If the gray distribution is uniform, the dynamic range is high, the contrast is high, and the image is clear. As the gray distribution is uniform, it is symmetric too. As described in section A, its third central moment equals zero in a symmetric distribution. Meanwhile, the experimental result shows that the skewness of a histogram approaches zero as the gray distribution becomes more uniform, especially for the images whose histogram is not bimodal. Therefore, the skewness of a histogram is adopted to evaluate the uniformity of the histogram.

In infinite-isp, a digital gain is applied to an image to correct the histogram skewness. Skewness defined in the above paper gave a huge value in a hundred thousand, so the skew function from the 'scipy.stats' library was used to implement skewness.



The sample skewness is computed as the Fisher-Pearson coefficient of skewness, i.e.

$$g_1 = \frac{m_3}{m_2^{3/2}}$$

where

$$m_i = \frac{1}{N}\sum_{n=1}^{N}(x[n] - \bar{x})^i$$

is the biased sample `ith` central moment, and $\bar{x}$ is the sample mean. If `bias` is False, the calculations are corrected for bias and the value computed is the adjusted Fisher-Pearson standardized moment coefficient, i.e.

$$G_1 = \frac{k_3}{k_2^{3/2}} = \frac{\sqrt{N(N-1)}}{N-2}\frac{m_3}{m_2^{3/2}}.$$

Figure 5: Skewness function from scipy.stats

Configuration Parameters

Parameters	Details
isEnable	When enabled, apply the 3A- Auto Exposure algorithm.
isDebug	Flag to output module debug logs.
center_illuminance	The value of center illuminance for skewness calculation ranges from 0 to 255. The default is 90.
histogram_skewness	The range of histogram skewness should be between 0 and 1 for correct exposure calculation.

Table 8: AE configuration parameters

Auto Focus

To be implemented.

Color Filter Array

Introduction

The CFA module is an integral part of the ISP pipeline that converts the black-level corrected, white-balanced linearized 2D image into the 3D RGB image format. A color filter array (CFA) is placed before the sensor to capture color information. This array, usually composed of three filters, limits the sensitivity of each photocell to one part of the visible spectrum. Consequently, each pixel of the CFA image only contains information about this limited range, i.e., one color response. However, three colors per pixel are needed to render images on a particular display device. Demosaicing is therefore applied to regenerate the missing colors.

After analyzing, we selected Malwar He Cutler's Demosaicing algorithm. Image demosaicing (or demosaicing) is the interpolation problem of estimating complete color information for an image captured through a color filter array (CFA), particularly on the Bayer pattern. The demosaicing is implemented by convolution with a set of linear filters. There are eight different filters for interpolating the different color components at different locations.

Algorithm Explanation

The CFA module is implemented in the pipeline after DPC, BLC, and WB. Instead of using a constant or near-constant hue approach, the algorithm uses the criterion of edges to have much stronger luminance than chrominance components. Thus, for interpolating a green value at a red pixel location, the algorithm compares the red pixel information with its estimate for a bilinear interpolation for the nearest red samples. If it differs from that estimate, it means there is a sharp luminance change at that pixel. Thus the algorithm bilinearly interpolates the green value by adding a portion of this estimated luminance change. So, this method is a gradient-corrected bilinear interpolation approach with a gain parameter to control how much correction is applied. The flowchart for the algorithm is shown as follows,

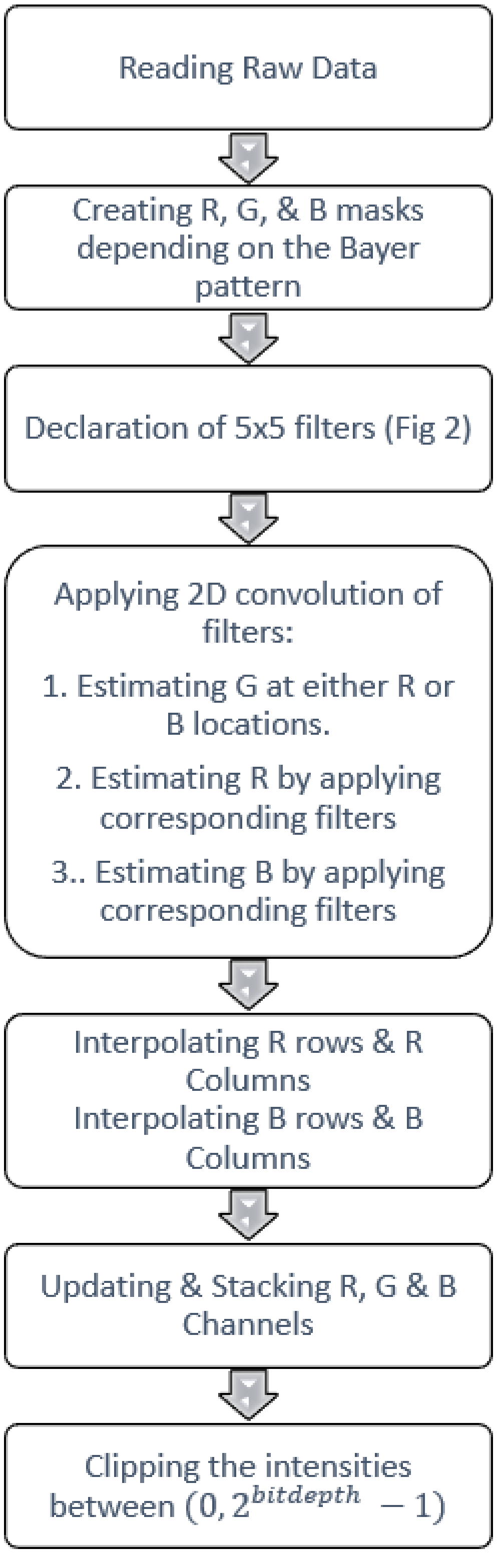

Reading Raw Data
Creating R, G, & B masks depending on the Bayer pattern
Declaration of 5x5 filters (Fig 2)
Applying 2D convolution of filters:
1. Estimating G at either R or B locations.
2. Estimating R by applying corresponding filters
3.. Estimating B by applying corresponding filters
Interpolating R rows & R Columns
Interpolating B rows & B Columns
Updating & Stacking R, G & B Channels
Clipping the intensities between $(0, 2^{bitdepth} - 1)$

Figure 6: Flowchart for the Malwar He Cutler's Demosaicing Algorithm

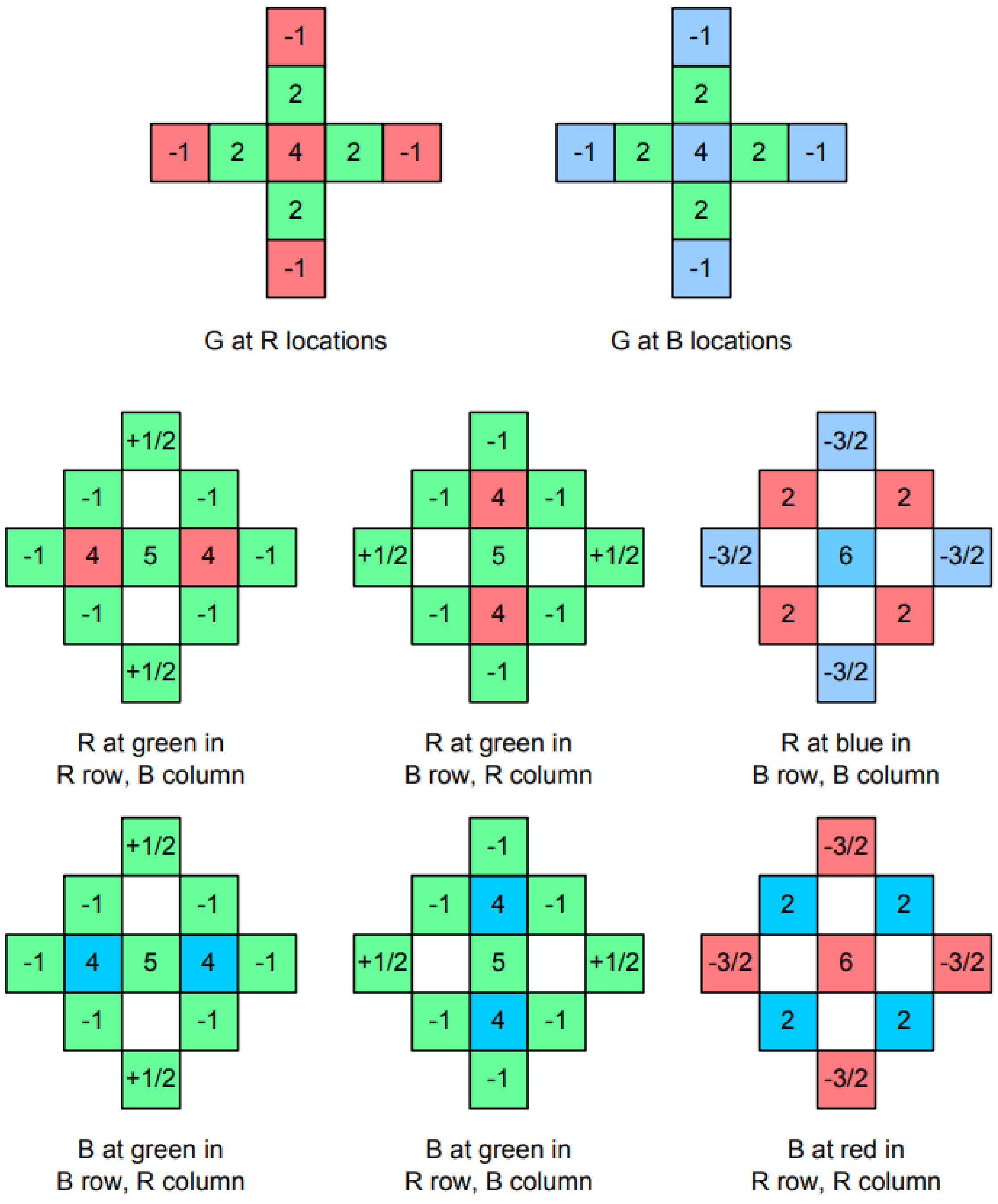

G at R locations
G at B locations
R at green in R row, B column
R at green in B row, R column
R at blue in B row, B column
B at green in B row, R column
B at green in R row, B column
B at red in R row, R column

Figure 7: Filter Coefficients for linear interpolation of R, G & B data.

Pictorial Explanation for algorithm

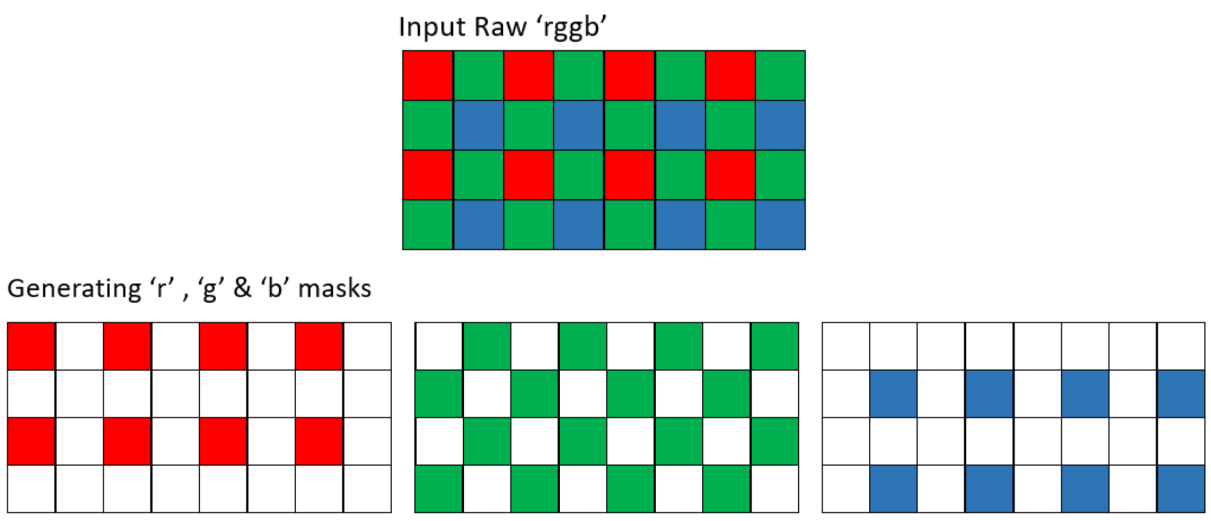
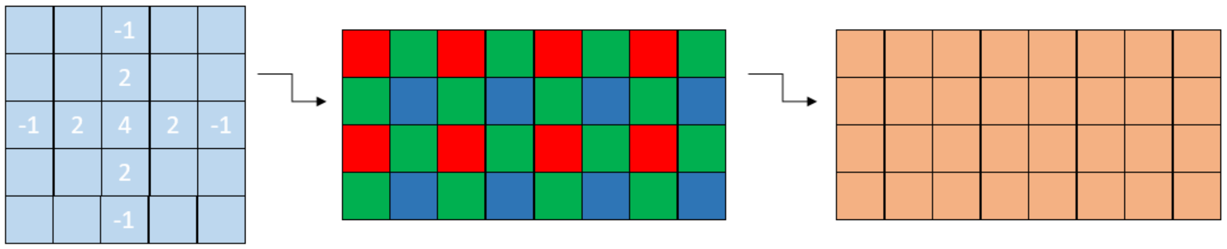


Figure 8: Masking channels

G Channel Extraction

5x5 filter Applied on Input Raw Raw after applying filter



Interpolating G channel after applying filter 1 on input raw. Estimating g pixels at r and b locations

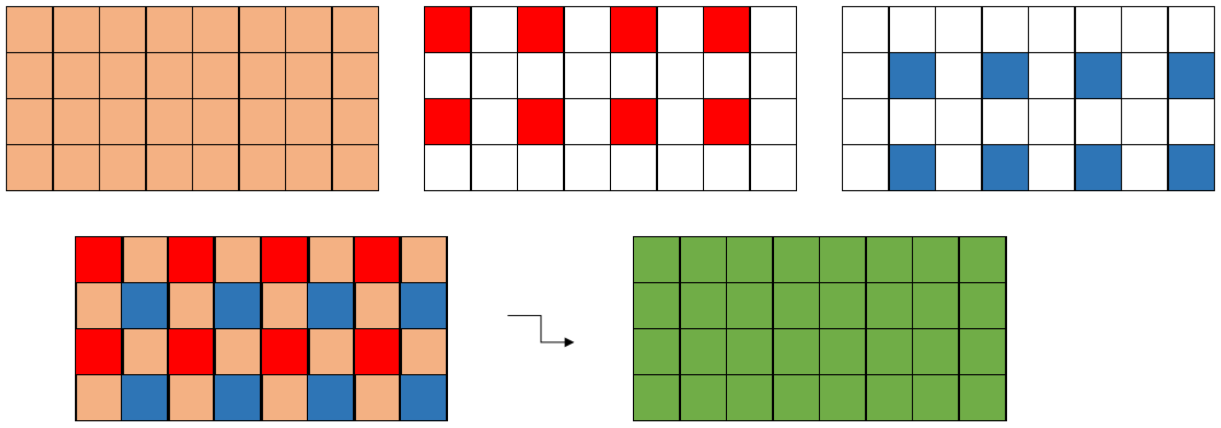


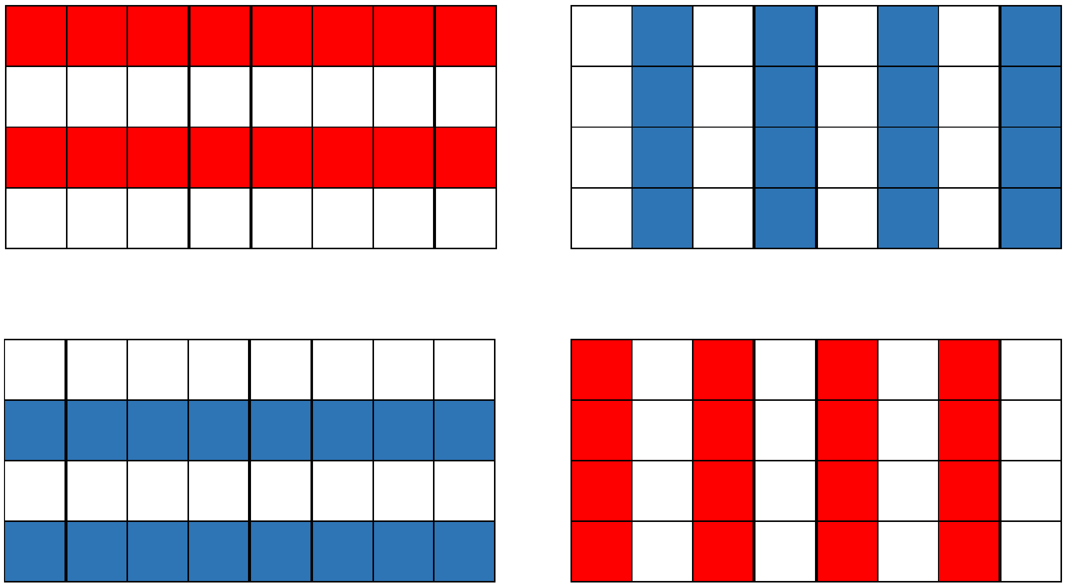
Figure 9: G interpolation

R Channel Extraction

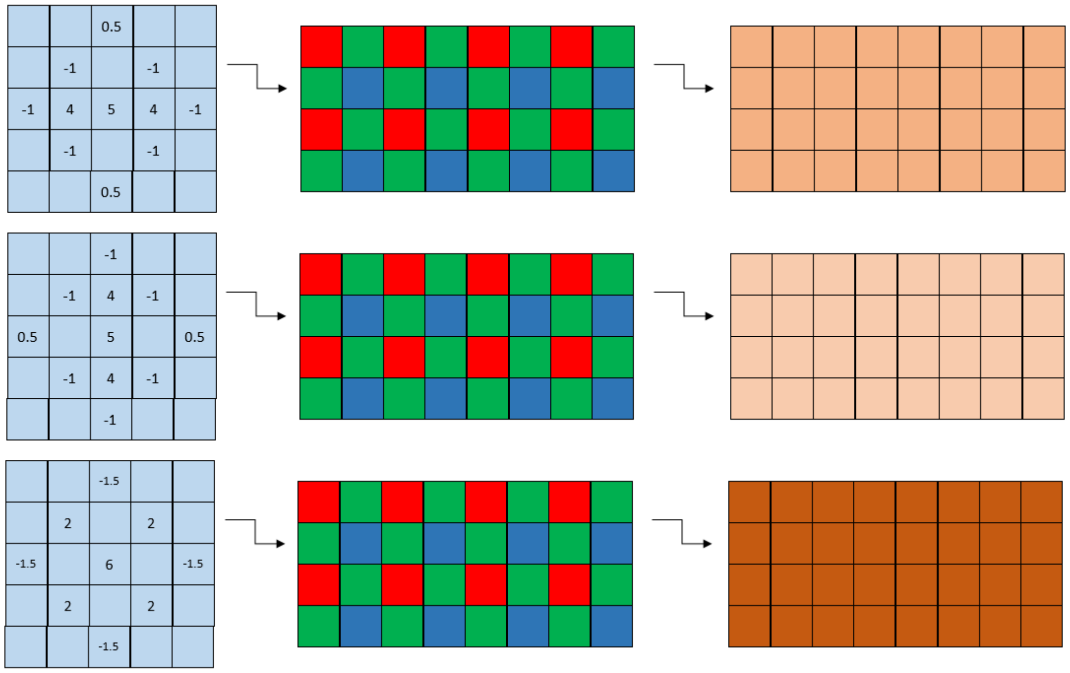
We have to estimate pixel values at three locations for R channel extraction.

1. R at green in R row and B column
2. R at green in B row and R column
3. R at blue in B row and B column

For this, we first extract R & B rows and R & B columns that we need to estimate.



Now we apply all the three respective filters.



Extracting R Channel

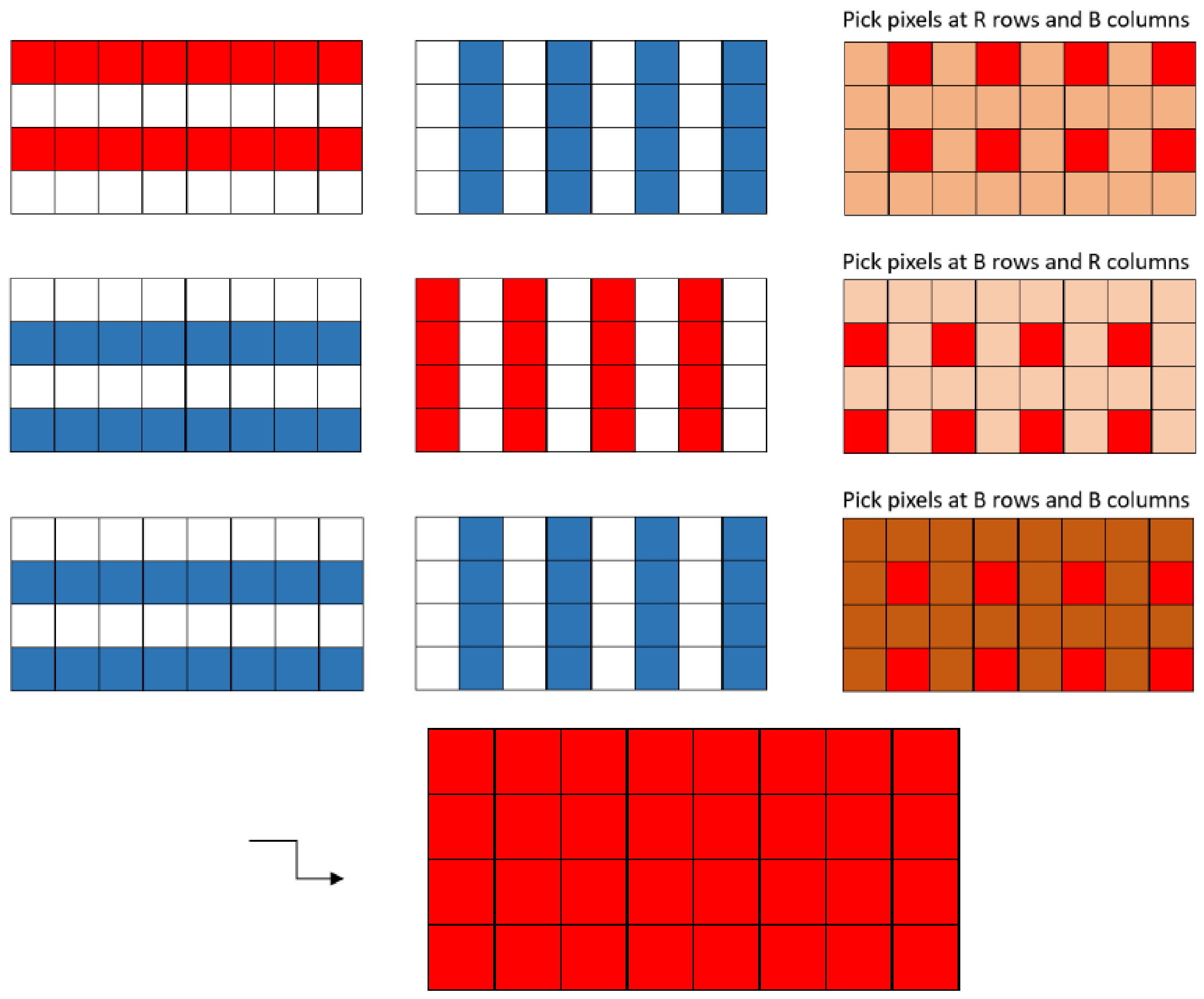

Pick pixels at R rows and B columns
Pick pixels at B rows and R columns
Pick pixels at B rows and B columns

Figure 10: R interpolation

B Channel Extraction

For B channel extraction the same pattern as for the R channel will be used, but now we have to estimate pixel values at three locations.

1. B at green in B row and R column
2. B at green in R row and B column
3. B at red in R row and R column

Extraction of B Channel

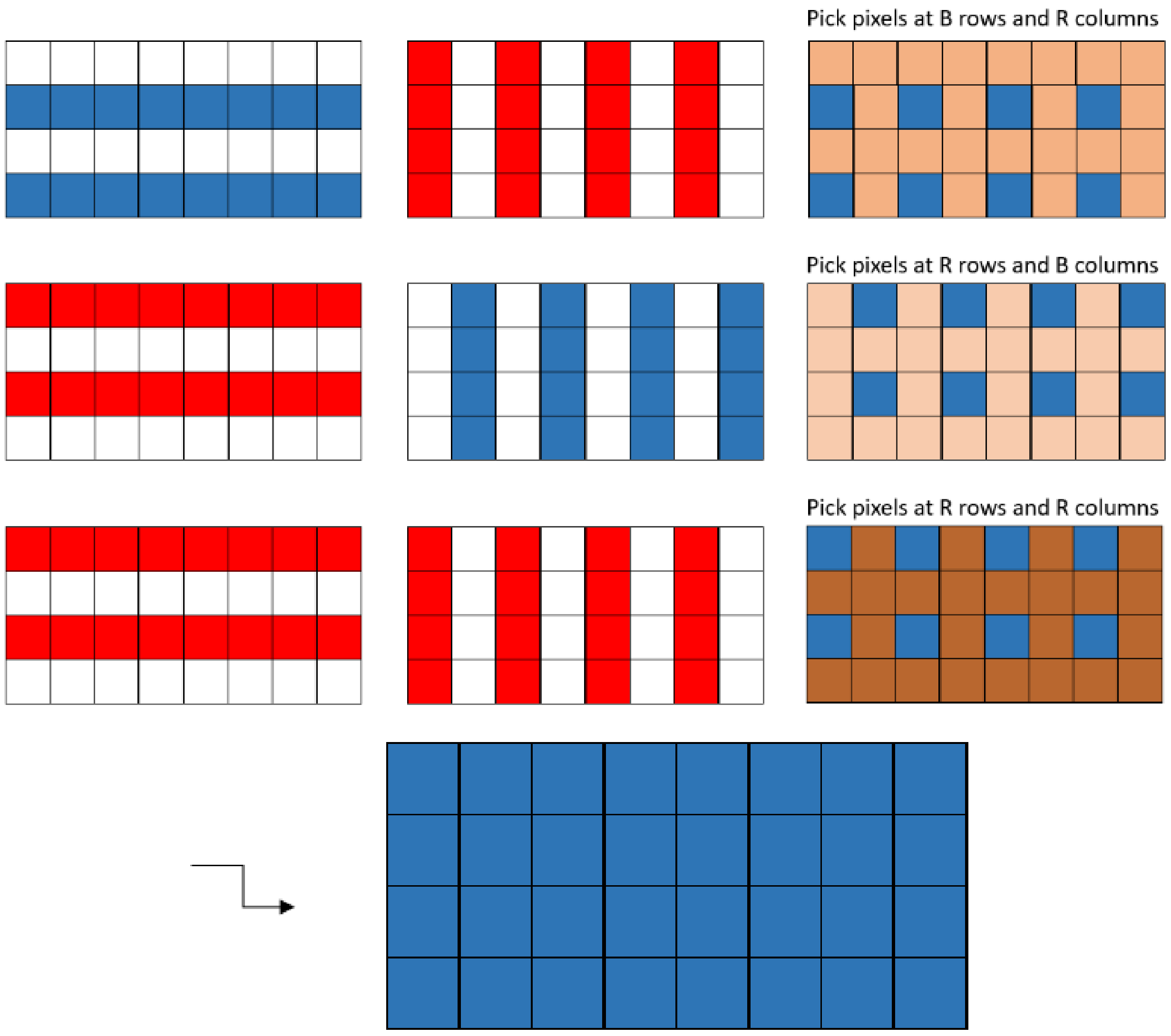

Pick pixels at B rows and R columns
Pick pixels at R rows and B columns
Pick pixels at R rows and R columns

Figure 11: B interpolation

Configuration Parameters

There is just one parameter in this module that determines whether it is enabled or disabled (isEnable). Since CFA is a crucial module, the pipeline is configured to have it activated by default.

Color Correction Matrix

Introduction

Optical spectral properties (lens, filters), various lighting sources, including tungsten, fluorescent, daylight and characteristics of the sensor's color filters, are responsible for producing inaccurate colors in the imaging. This module, thus, helps to mitigate the effect of false colors and applies color correction matrix on the image to get the correct display of the colors as obtained by the source imaging system.

Algorithm Explanation

A simple color correction matrix is applied in the following way to get the correct colors on the display.

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} RR & RG & RB \\ GR & GG & GB \\ BR & BG & BB \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} R_{offset} \\ G_{offset} \\ B_{offset} \end{bmatrix}$$

Configuration Parameters

Parameters	Details
$isEnable$	When enabled, the user given 3x3 CCM is applied to the 3D RGB image with rows sum to 1 convention. False: Disable True: Enable
$corrected\_red$	Row 1 of CCM
$corrected\_green$	Row 2 of CCM
$corrected\_blue$	Row 3 of CCM

Table 9: CCM configuration parameters

Gamma

Introduction

Gamma correction is a tool to alter the input by translating it to a nonlinear response. It matches the nonlinear response of display devices and broadens or compresses the dynamic range of images.

Algorithm Explanation

A look-up table is a common method for implementing gamma correction in hardware and is simple to modify and encode.

Configuration Parameters

Parameters	Details
$isEnable$	When enabled, applies tone mapping gamma using the LUT. False: Disable True: Enable
$gammaLut$	The look-up table for the gamma curve.

Table 10: Gamma configuration parameters

Color Space Conversion

Introduction

The color space conversion converts the RGB color of an image to another color space. Different color spaces are designed based on different proportions of Chroma and luma. The hardware display usually uses YUV color space. This is mainly because, in black and white television, only a single luminance channel was used. But as color television was introduced, U and V channels were used to transmit color and Y to transmit luminance to maintain backward compatibility.

Algorithm Explanation

In infinite-isp, the conversions are based on ITU-R standards. There are three ITU standards for YUV conversion.

1. BT.601 – for SDTV
2. BT.709 – for HDTV
3. Bt.2020 – for UHDTV

BT.601

We can divide conversion into two types analog and digital

Analog

In analog conversion, the input RGB is considered to be analog. The formula used for this conversion is derived using the ITU-R standard.

$$E_Y = 0.299E_R + 0.587E_G + 0.114E_B$$

$$E_{C_R} = \frac{E_R' - E_Y'}{1.402} = \frac{0.701E_R' - 0.587E_G' - 0.114E_B'}{1.402}$$

$$E_{C_B} = \frac{E_B' - E_Y'}{1.772} = \frac{-0.299E_R' - 0.587E_G' + 0.886E_B'}{1.772}$$

The resultant matrix is

$$Y\ U\ V\ = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The corresponding ranges are,

- Y : 0 to 1
- U&V : -0.5 to 0.5

Digital

In analog conversion, the input RGB is considered digital based on a number of bits. When the input RGB is considered digital based on the used, the digital conversion is called YCbCr.

$$Y = int\{(219E'_Y + 16) \times D\}/D$$

$$C_R = int\{\left(224E'_{C_R} + 128\right) \times D\}/D$$

$$C_B = int\{\left(224E'_{C_B} + 128\right) \times D\}/D$$

After putting the above E_Y, E_{C_R} and E_{C_B} and assuming analog RGB the resultant matrix is given by

$$Y\ U\ V\ = \begin{bmatrix} 66 & 129 & 25 \\ -38 & -74 & 112 \\ 112 & -94 & -18 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

- Y : 16 to 235
- U&V : 0-255 or [16-240]

The formula extends to n bit digital form.

In case we have digital n bit input the formula becomes.

$$E'_{R_D}(\text{in digital form}) = int\{(219E'_R + 16) \times D\}/D$$

$$E'_{G_D}(\text{in digital form}) = int\{(219E'_G + 16) \times D\}/D$$

$$E'_{B_D}(\text{in digital form}) = int\{(219E'_B + 16) \times D\}/D$$

$$Y = int\{\left(0.299E'_{R_D} + 0.587E'_{G_D} + 0.114E'_{B_D}\right) \times D\}/D$$
$$\approx int\left\{\frac{k'_{Y1}}{2^m}E'_{R_D} + \frac{k'_{Y2}}{2^m}E'_{G_D} + \frac{k'_{Y3}}{2^m}E'_{B_D}\right\} \times D\}/D$$

$$C_R = int\left\{\left\{\left(\frac{0.701E'_{R_D} - 0.587E'_{G_D} - 0.114E'_{B_D}}{1.402}\right)\frac{224}{219} + 128\right\}\ \times D\right\}/D$$
$$\approx int\left\{\left(\frac{k'_{Y1}}{2^m}E'_{R_D} + \frac{k'_{Y2}}{2^m}E'_{G_D} + \frac{k'_{Y3}}{2^m}E'_{B_D}\right) + 128\right\} \times D\}/D$$

$$C_B = int\left\{\left\{\left(\frac{-0.299E'_{R_D} - 0.587E'_{G_D} + 0.886E'_{B_D}}{1.772}\right)\frac{224}{219} + 128\right\}\ \times D\right\}/D$$
$$\approx int\left\{\left(\frac{k'_{Y1}}{2^m}E'_{R_D} + \frac{k'_{Y2}}{2^m}E'_{G_D} + \frac{k'_{Y3}}{2^m}E'_{B_D}\right) + 128\right\} \times D\}/D$$

The coefficients derived by using the above equations listed in a table in the referenced ITU-R document. These coefficients are different from what was calculated, requiring further research.

BT.709

BT.709 is similar to BT.601. Following are the conversion matrices

Analog

$$Y\ U\ V\ = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.1147 & -0.3854 & 0.5 \\ 0.5 & -0.4542 & -0.0458 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Digital

$$Y\ U\ V\ = \begin{bmatrix} 47 & 157 & 16 \\ -26 & -86 & 112 \\ 112 & -102 & -10 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

All these conversion matrices are derived from ITU R formulas, like above.

In all the above cases, the resultant signals range from 16 to 235 for Y’ (Cb and Cr range from 16 to 240); the values from 0 to 15 are called foot-room, while the values from 236 to 255 are called head-room.

Configuration Parameters

Parameters	Details
$isEnable$	This is an essential module, so it cannot be disabled.
$conv\_standard$	The standard to be used for conversion - 1 : Bt.709 HD - 2 : Bt.601/407
$conv\_type$	The conversion types - 1 : Analogue YUV - 2 : Digital YCbCr

Table 11: CSC configuration parameters

Local Dynamic Contrast Improvement

Introduction

LDCI stands for local dynamic contrast improvement. This module enhances the image's contrast, especially in cases where the light intensity is low. After exploring a variety of algorithms, we chose the CLAHE (contrast limited adaptive histogram equalization) method because of its results and low complexity level.

Algorithm Explanation

1. The algorithm is applied to the luminance channel of the RGB image obtained after the CSC module.
2. The input luminance map **I** of dimensions $\boldsymbol{H} \times \boldsymbol{W}$ is divided into tiles (non-overlapping) based on the input parameter of block/window size **w**.
3. A LUT is created using the histogram equalization method for each block. HE method gets the histogram of the block and clips the bin counts between 0-1 for smaller window size (avoiding too sharp details). After clipping the histogram, the pdf is computed by dividing it with the total count. CDF is then calculated based on the pdf; this is how a new LUT is created for each block.

   Explanation: We have to make the following changes to the algorithm for smaller window size.
   1. We compute histograms for each tile in the minimum and maximum gray levels range, i.e. (0 - 255).
   2. We used the clipping limit as a user-defined parameter. Still, we check on it to be within the available window size because, in either case, there will be artifacts like too much detail or over-contrast enhancement.
   3. We adjusted the total count by adding the clipped number of pixels in the clipped histogram to get the appropriate normalization factor for computing CDF.
   4. To give appropriate weightage to every gray level, I added a DC offset of 1 to the updated histogram so that in generating an updated LUT, each gray level shows some value that previously remained unconsidered in low-light images.
   5. This concept can be better visualized from the following picture.

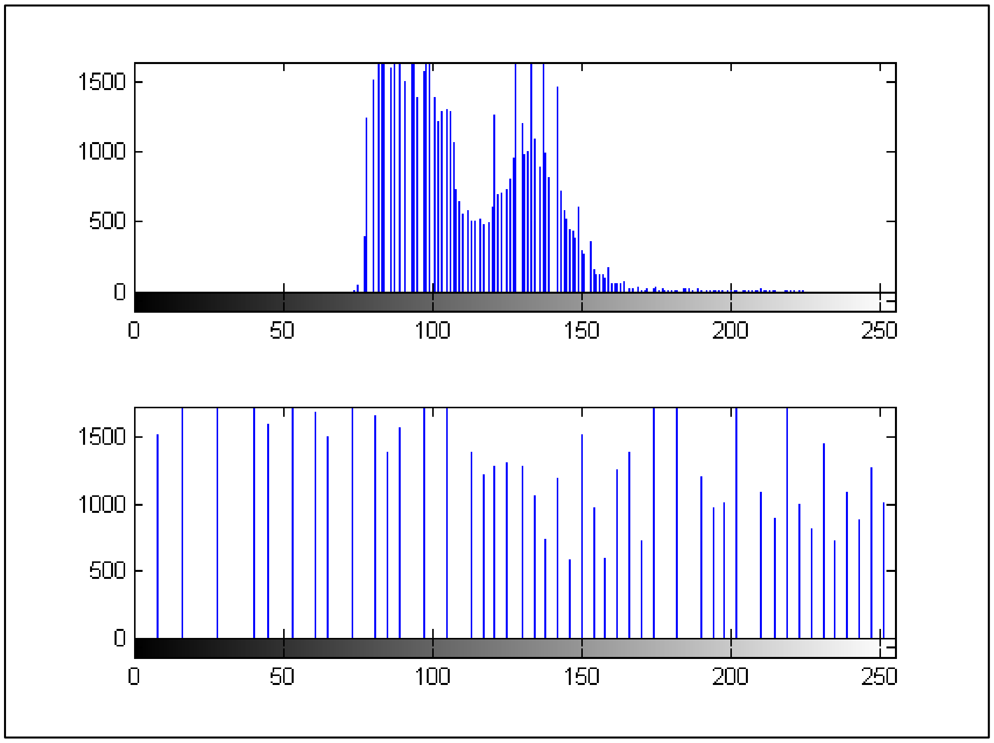


Figure 12: Histograms for low contrast and enhanced contrast images

4. After generating the LUT for each block, the interpolation process is done to mitigate the block artifact. For this, there are four checks in the algorithm to see if the block is present at the corner, left or right, top or bottom, or in the middle. Weighted LUT is applied on the neighboring blocks that are then interpolated to get the updated values in the current block.
5. After interpolation, the current block is updated; thus, the output luminance map is generated at the end.

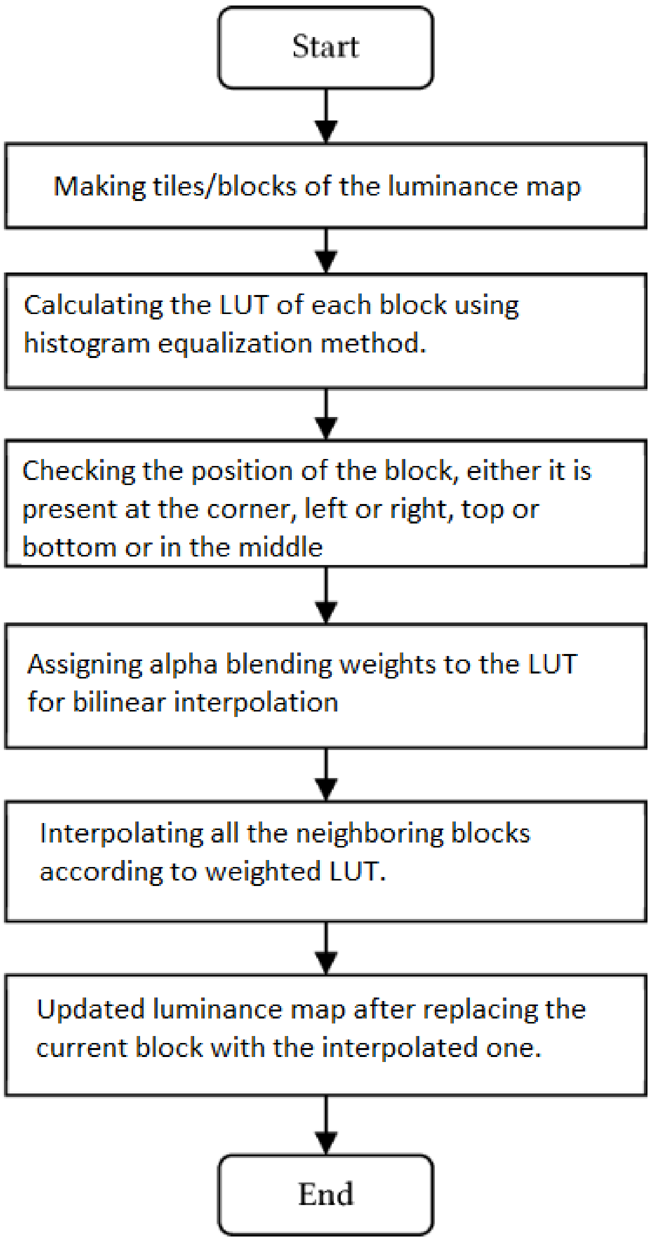

Start
Making tiles/blocks of the luminance map
Calculating the LUT of each block using histogram equalization method.
Checking the position of the block, either it is present at the corner, left or right, top or bottom or in the middle
Assigning alpha blending weights to the LUT for bilinear interpolation
Interpolating all the neighboring blocks according to weighted LUT.
Updated luminance map after replacing the current block with the interpolated one.
End

Figure 13: Flowchart for LDCI module (CLAHE)

Configuration Parameters

Parameters	Details
$isEnable$	When enabled, applies the LDCI to the Y channel. False: Disable True: Enable
$clip\_limit$	The clipping limit controls the amount of detail to be enhanced.
$wind$	Window size for applying the filter.

Table 12: LDCI configuration parameters

2D Noise Reduction

Introduction

As a denoising tool, the 2DNR module in the ISP pipeline is implied following the LDCI module. This module was added to the system to reduce the impact of image noise, which degrades the image quality. In doing a literature survey, the behavior of different spatial domain denoising methods was studied along with their effect caused by adaptively chosen weights associated with them. Considering the hardware & computational constraints, the conventional Non-local means filter method was selected for denoising images.

Algorithm Explanation

NLM is a spatial domain denoising technique. These types of methods can be further categorized as local and non-local filters. Natural images have a characteristic of self-similarity, and exploiting this information, non-local means filter NLM, was originated. NLM denoises an image by taking weighted average values of all the similar pixels in the entire image.

The weights assigned to these similar pixels are computed based on the Euclidean distance between the intensities of the pixels. They are sorted in a decreasing fashion to assign more weight to the most similar pixel in the image. This notion is based on the idea that for denoising a single pixel in an image, the weighted average is computed using all those pixels in the entire image, which are similar in intensity y of gray levels but close to each other based on their Euclidean distance. The NL-means filter assigns more weight to the pixels close to the targeted pixel and will assign less weight to those far away.

Visual representation

1. Taking input image of $H \times W \times 3$ size
2. Extracting Y channel
3. Declaring zero arrays for output and weights matrices
4. Padding the Y_in using reflect mode as shown below

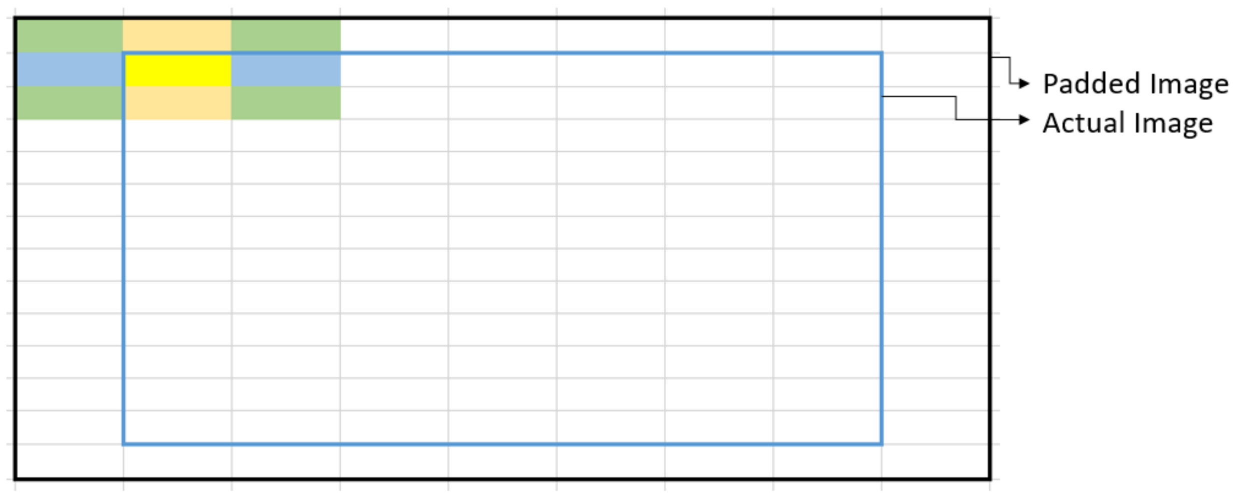


Figure 14: Actual and padded image for 2DNR

5. Generating LUT (look-up table) linearly using an exponential function. The minimum distance (most similar pixels) would be assigned the largest weight.
6. Generating shifted arrays according to the search window like below.

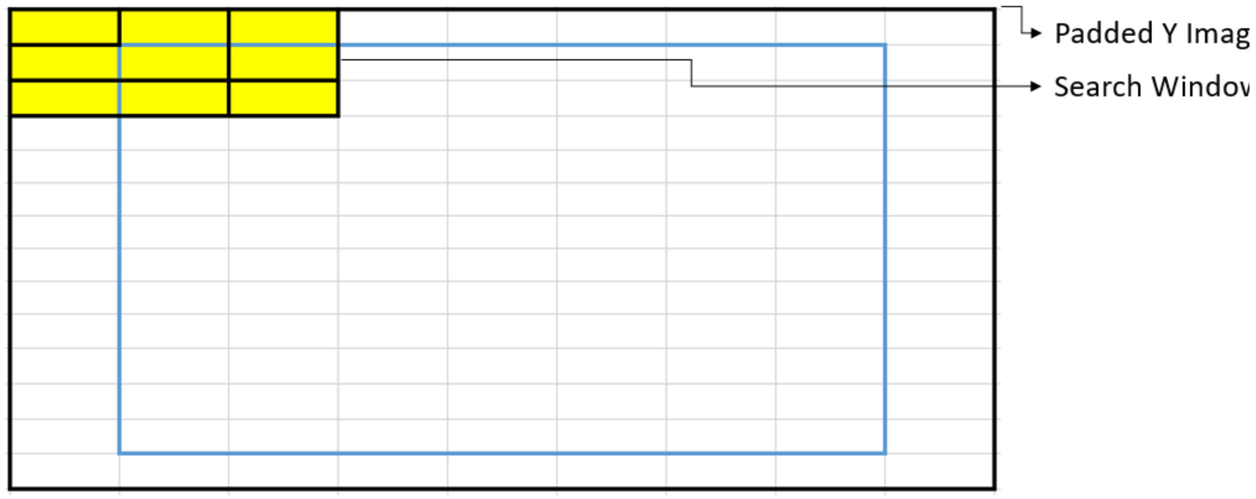


Figure 15: Shifted arrays according to the search window

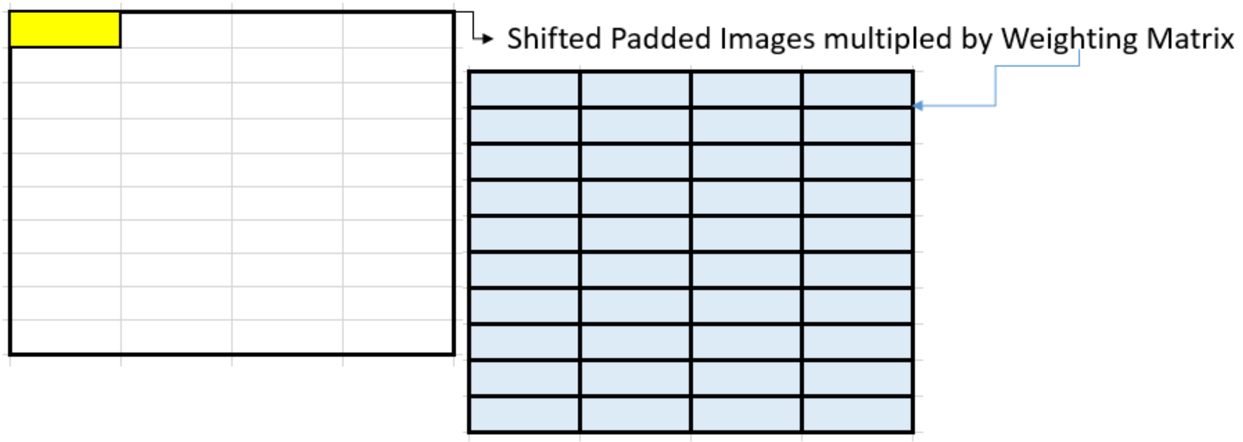


Figure 16: Shifted padded image multiplied by weighting matrix

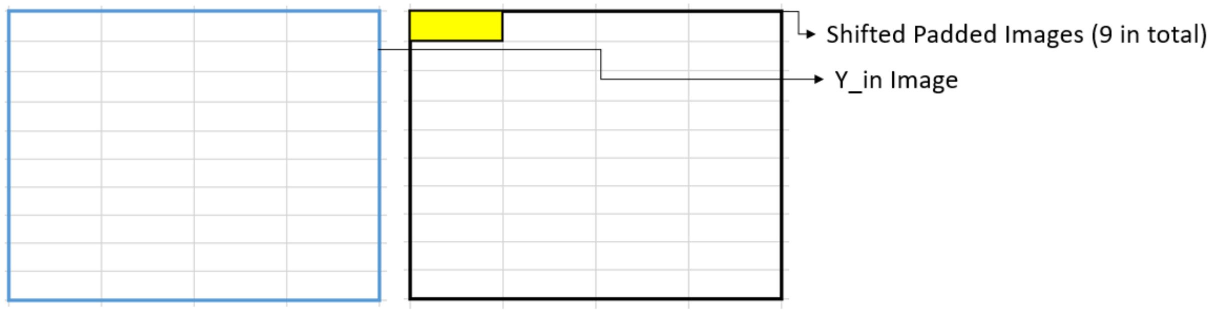


Figure 17: Shifted image

7. The distance will be computed between Y_in and each shifted array.

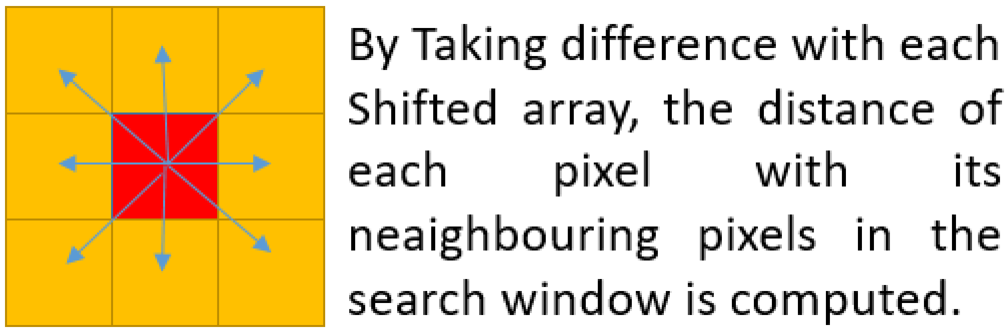

By Taking difference with each Shifted array, the distance of each pixel with its neaighbouring pixels in the search window is computed.

Figure 18: Computation pattern within a window

8. Mean filter is applied in each iteration after computing distance. Every time the mean filter is applied, it takes the averaging of neighborhoods around each pixel.
9. Based on the updated distances, weights are assigned to each pixel using the LUT method.
10. These weights are multiplied by the shifted array in each iteration and added to the denoised output declared zero before.
11. The weighting matrix is also added up in each iteration

12. After iterations, the denoised result is divided by the final weighting matrix to get the final denoised output.
13. The output is stacked to get the final 3D denoised YUV output.

Configuration Parameters

Parameters	Details
$isEnable$	When enabled, apply the non-local mean filtering. False: Disable True: Enable
$window\_size$	Search window size for applying the filter. It should be odd. 9 is set to avoid the computational cost.
$patch\_size$	Patch size for applying mean filtering. It should be odd. 5 is set by default.
h	Strength parameter. The higher the value, the higher will be the smoothing or blurring. Set to 5.

Table 13: 2DNR configuration parameters

YUV Format —444-422

Introduction

The YUV conversion format is a way of subsampling YUV images to save bandwidth. This is possible because eyes do not perceive color differences as sharply as they do for luminance differences; hence, this allows subsampling of the CbCr channel without adversely affecting the output.

No subsampling is done for the Y channel

Algorithm Explanation

In theory, there are two types of formats:

1. Packed Format
2. Planar Format

Packed Format

YUV formats are divided into *packed* formats and *planar* formats. In a packed format, the Y, U, and V components are stored in a single array. Pixels are organized into groups of macro pixels whose layout depends on the format. Each of the YUV formats described has an assigned FOURCC code. A FOURCC code is a 32-bit unsigned integer created by concatenating four ASCII characters.

Here is a list

- 4:4:4
    - AYUV
- 4:2:2
    - YUY2
    - UYVY

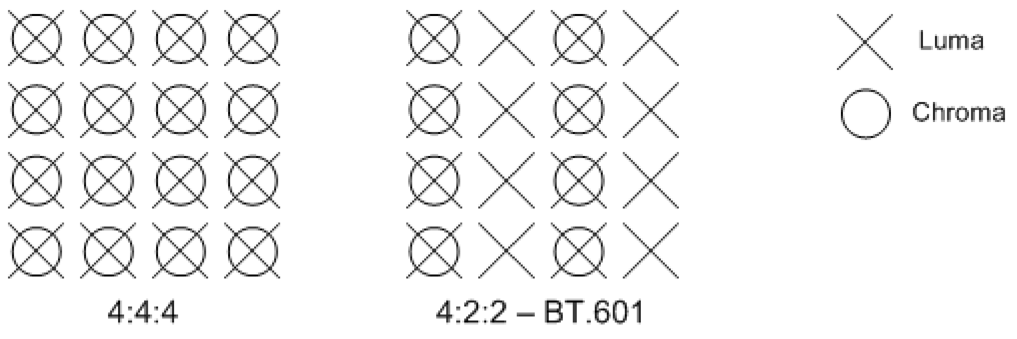


Figure 19: Pictorial representation of 4:4:4 and 4:2:2

4:4:4 Format

In this format, no subsampling is done, and YUV is saved in the following manner displayed

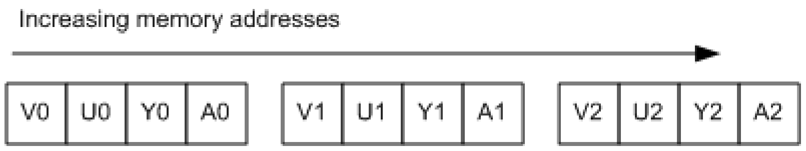
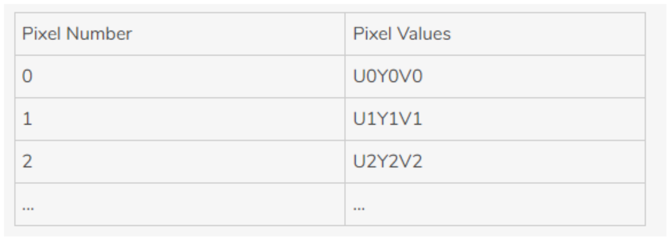


Figure 20: Memory view of bytes for packed format

The recommended FOURCC format for 4:4:4 is displayed above. In FOURCC, each memory element is 32 bits having 4 bytes, each representing either Y or U, or V based on the type of formatting chosen.

In infinite ISP, however, we have implemented the simple 4:4:4 format where each memory chunk has 3 bytes of Y, U, or V. Since we don't have any information regarding the alpha channel yet, we're not considering it in this conversion.

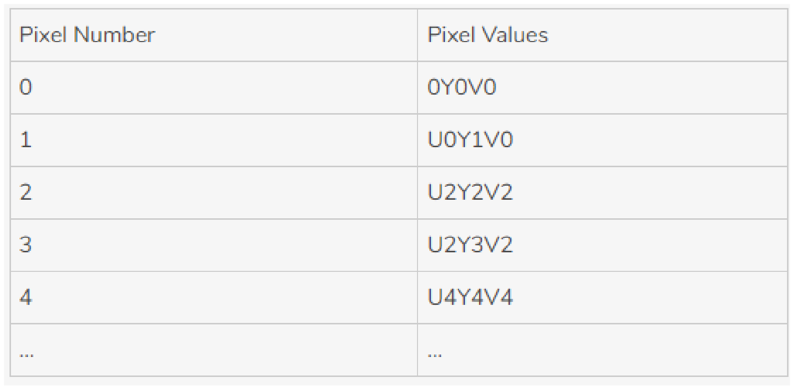


Pixel Number	Pixel Values
0	U0Y0V0
1	U1Y1V1
2	U2Y2V2
...	...

Figure 21: 4:4:4 format

4:2:2 Format

Every 2 pixels share U and V. This is achieved by taking the same U and V for each of the two pixels. The image below shows that the same U and V are used for Y1 and Y2. Similarly, the same U and V are used for Y3 and Y4.



Pixel Number	Pixel Values
0	0Y0V0
1	U0Y1V0
2	U2Y2V2
3	U2Y3V2
4	U4Y4V4
...	...

Figure 22: 4:2:2 format

In FOURCC format, the above is saved as follows.

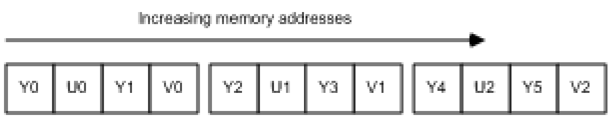
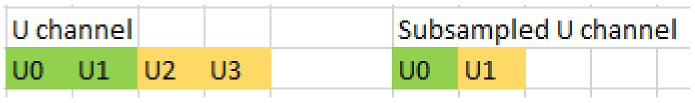


Figure 23: Memory view of 4:2:2

The first 4 bytes represent 2 pixels. Both the 2 pixels have the same U and V but different Y. Please note that U and V in the above picture are subsampled entries of corresponding U and V channels.



U channel					Subsampled U channel			
U0	U1	U2	U3		U0	U1		

The subsampling is done horizontally for each row of the U, and the V channel in infinite-isp UYVY format is implemented.

Planar Format

The Y, U, and V components are stored in a planar format as three separate planes.

- 4:2:0
    - IMC1
    - IMC3
- 4:2:0
    - IMC2
    - IMC4
    - YV12
    - NV12

Right now, no planar format is supported by infinite-isp.

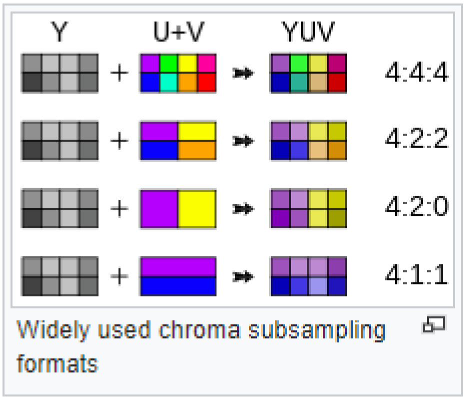


Figure 24: Another visualization of formats

Configuration Parameters

Parameters	Details
$isEnable$	Will enable or disable this module. False: Disable True: Enable
$conv\_type$	Can convert the YCbCr to YUV. - 444 - 422

Table 14: YUV Formats configuration parameters

Scaler

Introduction

The Scaler block will be implemented as one of the final blocks in the ISP pipeline to downscale a full-resolution image into a desired resolution/size per the user's requirements. The Scalers block is generally implemented in IP camera ISPs to create multiple channels/streams for storage and transmission in surveillance cameras. Aspect ratio is the ratio between image height and width $\frac{h}{w}$. Scaling only preserves the aspect ratio when height and width are scaled by the same factor numerator $\frac{h*c}{w*c} = \frac{h}{w}$. Cropping does not preserve the aspect ratio even if same amount is cropped from both height and width $\frac{h-c}{w-c} \neq \frac{h}{w}$.

Up/Down scaling height and width at the same time is equivalent to scaling the height first and width second or vice versa. For example, downscaling with averaging using a square 2×2 window is equal to downscaling by 2×1 window followed by downscaling with 1×2 window.

Algorithm Explanation

- Input image: 3D array (RGB or YUV image) of size 2592×1944 or 2592×1536 or 1920×1080.
- Output image: 3D array (RGB or YUV image).
  - Hardware-friendly approach: Valid output sizes are mentioned in the table.
  - Image can be up/down scaled to any size directly (using float values) using Nearest Neighbor or Bilinear method.

Hardware friendly approach

This approach works with integer values only (except upscaling with the bilinear method, where weights are float values).

- Down-sample the image with an even integer factor [2].
- Crop image such that after cropping, the size of the array becomes a multiple of the required output size i.e.

  $\text{required size} = \text{q} * \text{cropped size}$, where q $\{2/3, 3/4, 4/7, 5/7\}$ as mentioned in Table below.
- Once a rational scale factor (n/d) is determined, scale the image to get the scaled output.
- The scaling algorithm shows an image by a non-integer factor n/d is equivalent to upscaling the image with an integer factor n followed by downscaling with an integer factor d.
- Upscaling and downscaling are implemented using the nearest neighbor algorithm and bilinear interpolation. These parameters are user-defined.

- Downscaling for NN and bilinear interpolation is implemented using convolution to reduce run time.
- For each input size, crop value (minimum number of rows or columns to crop) is computed as follows:

$$(Input\ size\ -\ crop_{val}) * non_{integer} scale\ factor\ =\ Output\ size$$

$$crop_{val} = Input\ size\ - \frac{Output\ size}{noninteger\ scale\ factor}$$

- The algorithm scales each channel one by one using the same steps.
- The user can use the “isDebug” flag to print out details of the above three steps.
- Why use the 3-step approach? The non-integer scale factor to directly scale an image from a specified input size to output size could be memory intensive; hence the down-sampling is performed in 3 steps (e.g., for 1944 → 720, the scale factor is 2.7 or 27/10 or 1536→1080, the scale factor is 1.4222222222222223 which cannot be written in fraction form and requires cropping).

The table below shows the hand-picked scaling factors and crop values used for hardware friendly approach. These values are hardcoded in the code for each input size using classes.

Input Size	Output size	Downscale scale factor		Crop value		Non-integer scale factor	
		width	height	width	height	width	height
2592 × 1944	2560 × 1440	–	–	32	24	–	3/4
2592 × 1944	1920 × 1080	–	–	32	54	3/4	4/7
2592 × 1944	1280 × 960	2	2	16	12	–	–
2592 × 1944	1280 × 720	2	2	16	12	–	3/4
2592 × 1944	640 × 480	4	4	8	6	–	–
2592 × 1944	640 × 360	4	4	8	6	–	3/4
1920 × 1080	1280 × 720	–	–	–	–	2/3	2/3
1920 × 1080	640 × 480	3	2	–	60	–	–
1920 × 1080	640 × 360	3	3	–	–	–	–
2592 × 1536	1920 × 1080	–	–	32	24	3/4	5/7
2592 × 1536	1280 × 720	2	2	16	48	–	–

Input Size	Output size	Downscale scale factor		Crop value		Non-integer scale factor	
		width	height	width	height	width	height
2592×1536	640×480	4	3	8	32	–	–
2592×1536	640×360	4	4	8	24	–	–

Table 15: Hardware friendly scaling

Direct scaling: The user can scale an image to any output size with NN and Bilinear method by setting the "isHardware" flag to False.

Configuration Parameters

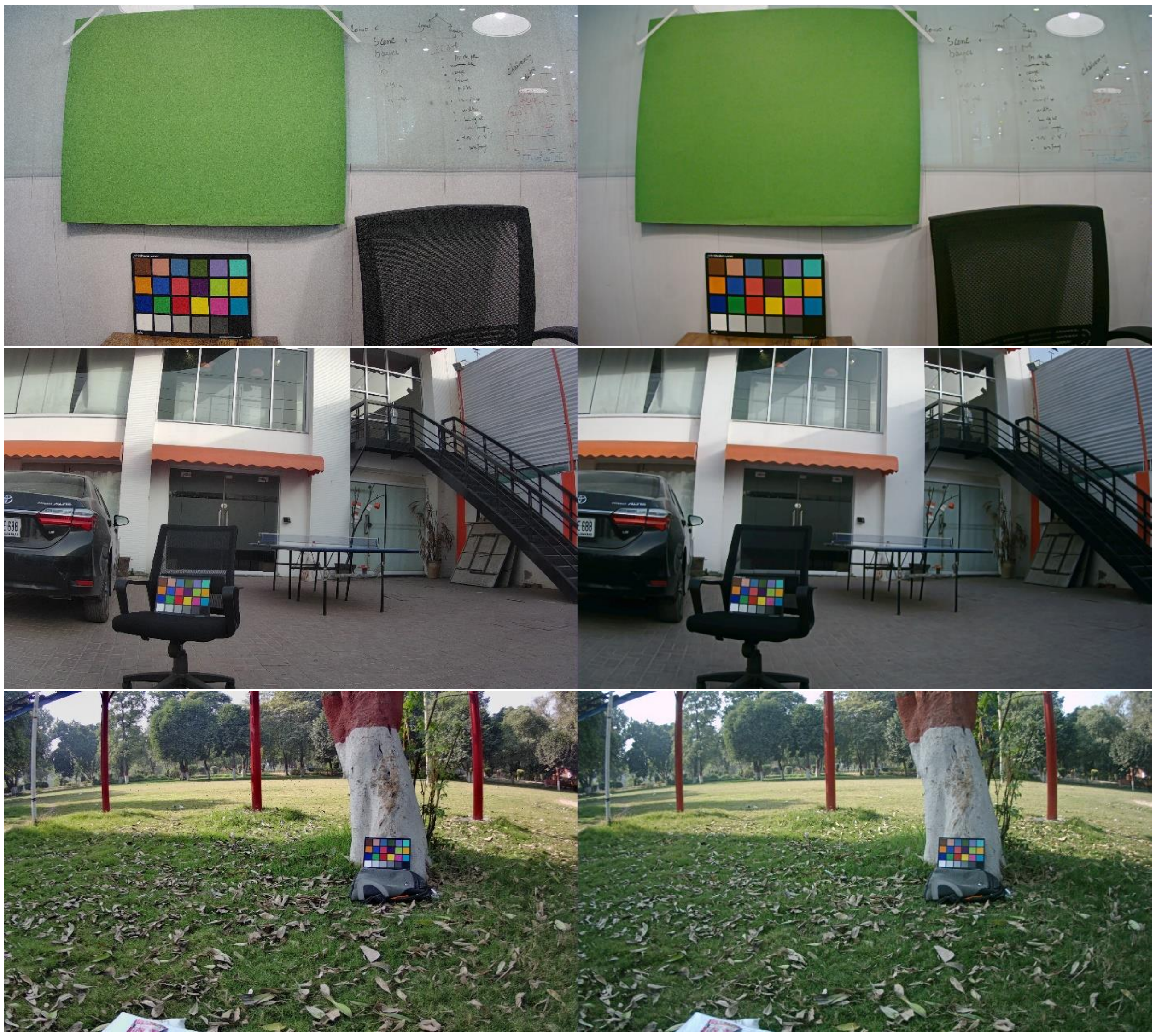
Parameters	Details
isEnable	When enabled, scales down the input image.
isDebug	Flag to output module debug logs.
new_width	Downscaled width of the output image.
new_height	Downscaled height of the output image.
isHardware	When true, applies the hardware-friendly techniques for downscaling. This can only be applied to any one of the input sizes 3 input sizes and can downscale to - 2592x1944 to 1920x1080 or 1280x960 or 1280x720 or 640x480 or 640x360 - 2592x1536 to 1280x720 or 640x480 or 640x360 - 1920x1080 to 1280x720 or 640x480 or 640x360
Algo	Software-friendly scaling. They are only used when *isHardware* is disabled. - *Nearest_Neighbor* - *Bilinear*
upscale_method	Used only when isHardware is enabled. The upscaling method can be one of the above algorithms.
downscale_method	They are used only when isHardware is enabled. The downscaling method can be one of the above algorithms.

Table 16: Scale configuration parameters

Pipeline Results

Here are the results of this pipeline compared with a market-competitive ISP. infinite-isp outputs are displayed on the right, with the underlying ground truths on the left.

Ground Truths **infinite-ISP**



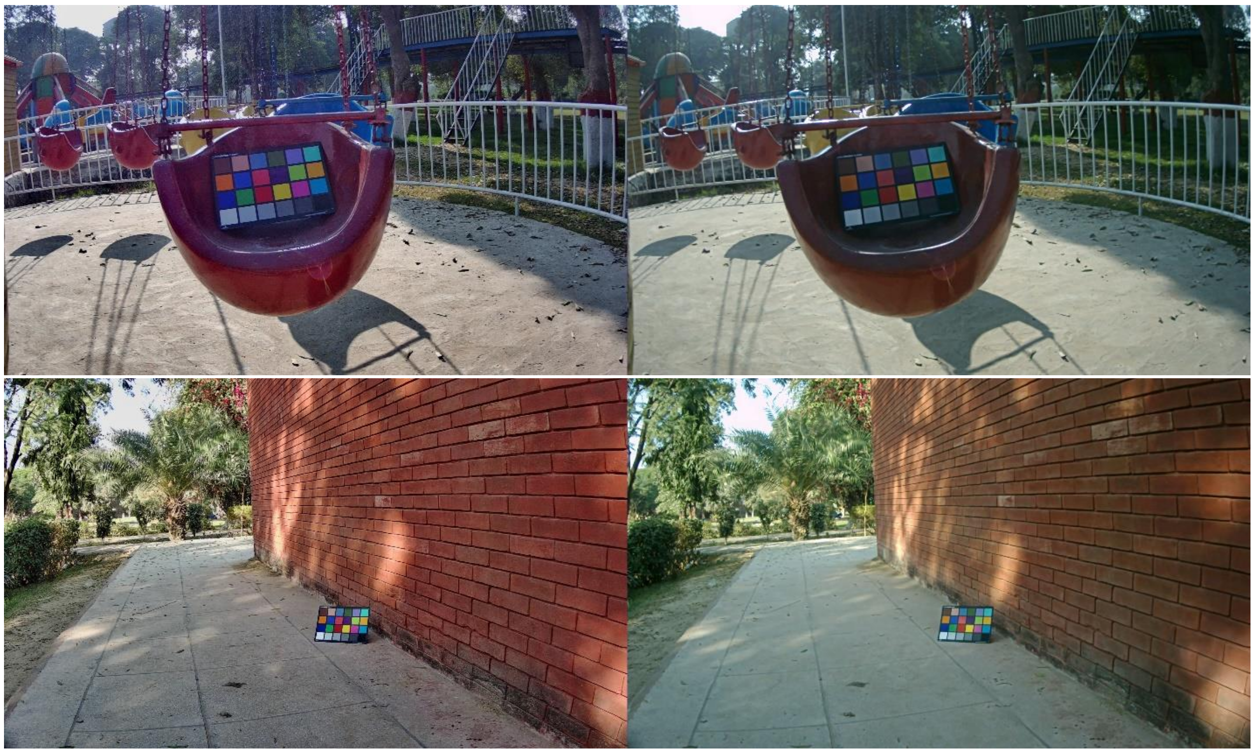


Figure 25: Pipeline results

IQ Metrics Analysis

Images	PSNR	SSIM
Indoor1	21.51	0.8624
Outdoor1	22.87	0.9431
Outdoor2	20.54	0.8283
Outdoor3	19.22	0.7867

Table 17: IQ Metrics Analysis

Features Comparison Matrix

A comparison of features with the famous openISP. infinite-isp also tries to simulate the **3A algorithms**.

Modules	infinite-isp	openISP
Crop	Bayer pattern-safe cropping	----
Dead Pixel Correction	Modified Yongji's et al., Dynamic Defective Pixel Correction for Image Sensor	Yes
Black Level Correction	Calibration/sensor dependent - Applies BLC from the config	Yes
Optical Electronic Transfer Function (OECF)	Calibration/sensor dependent - Implements a LUT from the config	----
Anti-Aliasing Filter	----	Yes
Digital Gain	Gains from config file	Brightness contrast control
Lens Shading Correction	To Be Implemented	----
Bayer Noise Reduction	Green Channel Guiding Denoising by Tan et al	Chroma noise filtering
White Balance	WB gains from config file	Yes
CFA Interpolation	Malwar He Cutler's demosaicing algorithm	Yes - Malwar He Cutler
3A - Algorithms	**AE & AWB**	----
Auto White Balance	- Gray World - Norm 2 - PCA algorithm	----
Auto Exposure	- Auto Exposure based on skewness	----
Color Correction Matrix	Calibration/sensor dependent - 3x3 CCM from config	Yes - 4x3 CCM
Gamma Tone Mapping	Gamma LUT in RGB from config file	Yes - YUV and RGB domain

Color Space Conversion	YUV analogue and YCbCr digital - BT 601 - Bt 709	Yes - YUV analog
Contrast Enhancement	Modified contrast limited adaptive histogram equalization.	----
Edge Enhancement / Sharpening	----	Yes
Noise Reduction	Non-local means filter	Yes - NLM filter - Bilateral noise filter
Hue Saturation Control	----	Yes
Scale	- Integer Scaling - Non-Integer Scaling	----
False Color Suppression	----	Yes
YUV Format	- YUV - 444 - YUV - 422	----

Table 18: Feature comparison

References

1. Measurement for optoelectronic conversion functions (OECFs) of the digital still-picture camera - NASA/ADS (harvard.edu)
2. https://www.imatest.com/?s=oecf
3. https://www.imatest.com/docs/esfriso_instructions/
4. https://slideplayer.com/slide/5759755/
5. ISO - ISO 14524:2009 - Photography — Electronic still-picture cameras — Methods for measuring optoelectronic conversion functions (OECFs)
6. (PDF) Linearisation of RGB Camera Responses for Quantitative Image Analysis of Visible and UV Photography: A Comparison of Two Techniques (researchgate.net)
7. Linearization in detail - three different ways to linearize images (image-engineering.de)
8. https://patentimages.storage.googleapis.com/f9/11/65/a2b66f52c6dbd4/US8538199.pdf
9. https://static.aminer.org/pdf/PDF/000/319/504/large_scale_infographic_image_downsizing.pdf
10. https://www.ipol.im/pub/art/2011/g_mhcd/article.pdf
11. https://www.ipol.im/pub/art/2011/bcm_nlm/article.pdf
12. https://www.itu.int/rec/R-REC-BT.709-6-201506-I/en
13. https://www.flir.com/support-center/iis/machine-vision/knowledge-base/understanding-yuv-data-formats/
14. https://en.wikipedia.org/wiki/Chroma_subsampling

15. https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/YuY2_files/intro.htm#YV12
16. https://learn.microsoft.com/en-us/windows/win32/medfound/recommended-8-bit-yuv-formats-for-video-rendering
17. https://sci-hub.hkvisa.net/10.1109/ICAIIS49377.2020.9194921
18. https://sci-hub.hkvisa.net/10.1109/apccas.2012.6419046
19. https://www.atlantis-press.com/article/25875811.pdf
20. https://www.sciencedirect.com/science/article/abs/pii/0016003280900587
21. https://library.imaging.org/admin/apis/public/api/ist/website/downloadArticle/cic/12/1/art00008
22. https://opg.optica.org/josaa/viewmedia.cfm?uri=josaa-31-5-1049&seq=0
23. https://www.hindawi.com/journals/tswj/2014/979081/
24. https://www.semanticscholar.org/paper/Contrast-Limited-Adaptive-Histogram-Equalization-Zuiderveld/726c95fa3bd47401befc7513b6e52d1c806f26af.
25. https://web.archive.org/web/20120113220509/http://radonc.ucsf.edu/research_group/jpouliot/tutorial/HU/Lesson7.htm
26. https://github.com/cruxopen/openISP
27. https://github.com/QiuJueqin/fast-openISP