IMAGE PIPELINE

Project Report

Eklavya Mentorship Programme

At

SOCIETY OF ROBOTICS AND AUTOMATION,
VEERMATA JIJABAI TECHNOLOGICAL
INSTITUTE, MUMBAI

AUGUST - SEPTEMBER 2022

ACKNOWLEDGEMENT

We are incredibly grateful to our mentors Kunal Agarwal and Rishabh Bali for their patience, motivation, enthusiasm, and immense knowledge they shared with us throughout the duration of the project.
We were able to explore a new domain of Image Processing and successfully work on it only because of the expert guidance provided to us
We would also like to thank all mentors of SRA VJTI for their constant support and motivation and for giving us the opportunity to be a part of Eklavya 2022.

Our Team :

Om Doiphode
omdoiphode161@gmail.com

Kedar Dhamankar
kedard2004@gmail.com

TABLE OF CONTENTS

- Blurring
    - Mean Filter
    - Gaussian Filter
    - Median Filter
- Rotation

Implementation

- Reading the image
- Pre-processing
- Post-processing

Conclusion and Future Work

References

PROJECT OVERVIEW

Description of Project :

The image pipeline takes a raw image from the sensor and converts it to a meaningful image. Several algorithms like debayering, Black Level correction, auto-white balance, denoising, etc will be first implemented to construct a meaningful image. Then additional algorithms can be implemented on the constructed image to post-process it. Like Flipping, blending, and overlaying images.
All algorithms will be implemented on a static raw image captured from a sensor.
The first part of this project is similar to what happens in an ISP (Image Signal Processor) in which all algorithms are designed based on hardware, but we will be designing those such that they are hardware-independent.

RAW Image

Processed Image

Tech Stack :

1. OpenCV
2. C++
3. Python

Brief Idea :

First, convert the RAW image into a .tiff file which stores the CFA image with missing pixel values. Interpolate the missing pixel values by applying Debayering algorithms like the ones proposed by Malvar-He-Cutler and store the image in a 2d vector. Then furthermore algorithms like Auto White Balance, Auto Exposure, Color Correction, etc are applied to the image which is stored in the form of a vector, and finally, the processed image can be displayed using OpenCV.

2. THEORY

What is RAW?

'RAW' is a class of computer files that typically contain an uncompressed image containing both the sensor pixel values and a large amount of meta-information about the image generated by the camera (the EXIF data). RAW files themselves come in many proprietary file formats (Nikon's .NEF, Canon's .CR2, etc.), but there is a common open format, .DNG, which stands for Digital Negative. The latter indicates how these files are supposed to be thought of by digital photographers: the master originals, and repositories of all the captured information of the scene.

Raw data from an image sensor contains light intensity values captured from a scene, but these data are not intrinsically recognizable to the human eye. The data is a single channel intensity image, possibly with a non-zero minimum value to represent 'black', with integer values that contain 10-14 bits of data (for typical digital cameras). Rather than speaking about an intrinsic 'white' value, no values in the image will be above some maximum which represents the saturation point of the physical camera sensor (e.g., a CMOS or CCD sensor).

Raw sensor data typically comes in the form of a Color Filter Array (CFA). This is an m-by-n array of pixels (where m and n are the dimensions of the sensor) where each pixel carries information about a single-color channel: red, green, or blue. Since light falling on any given photo site (pixel) in the CCD sensor is recorded as some number of electrons in a capacitor, it can only be saved as a scalar value; a single pixel cannot retain the 3-dimensional nature of observable light. CFAs offer a compromise where information about each of the three-color

channels is captured at different locations by means of spectrum-selective filters placed above each pixel.
The most common CFA pattern is the Bayer array. There are twice as many pixels that represent green light in a Bayer array image because the human eye is more sensitive to variation in shades of green and it is more closely correlated with the perception of light intensity of a scene.

PREPROCESSING ALGORITHMS -

2.1 Normalization

- Image normalization is a typical process in image processing that changes the range of pixel intensity values.
- Its normal purpose is to convert an input image into a range of pixel values that are more familiar or normal to the senses, hence the term normalization.
- In this work, we will perform a function that produces a normalization of an input image (grayscale or RGB).
- Then, we understand a representation of the range of values of the scale of the image represented between 0 and 255, in this way we get, for example, that very dark images become clearer.
- The linear normalization of a digital image is performed according to the formula
  Output_channel = 255 * (Input_channel - min) / (max-min)
- If we are using a grayscale image, we only need to normalize using one channel.

- However, if we are normalizing an RGB (3 channels) we need to normalize each channel using the same criteria.

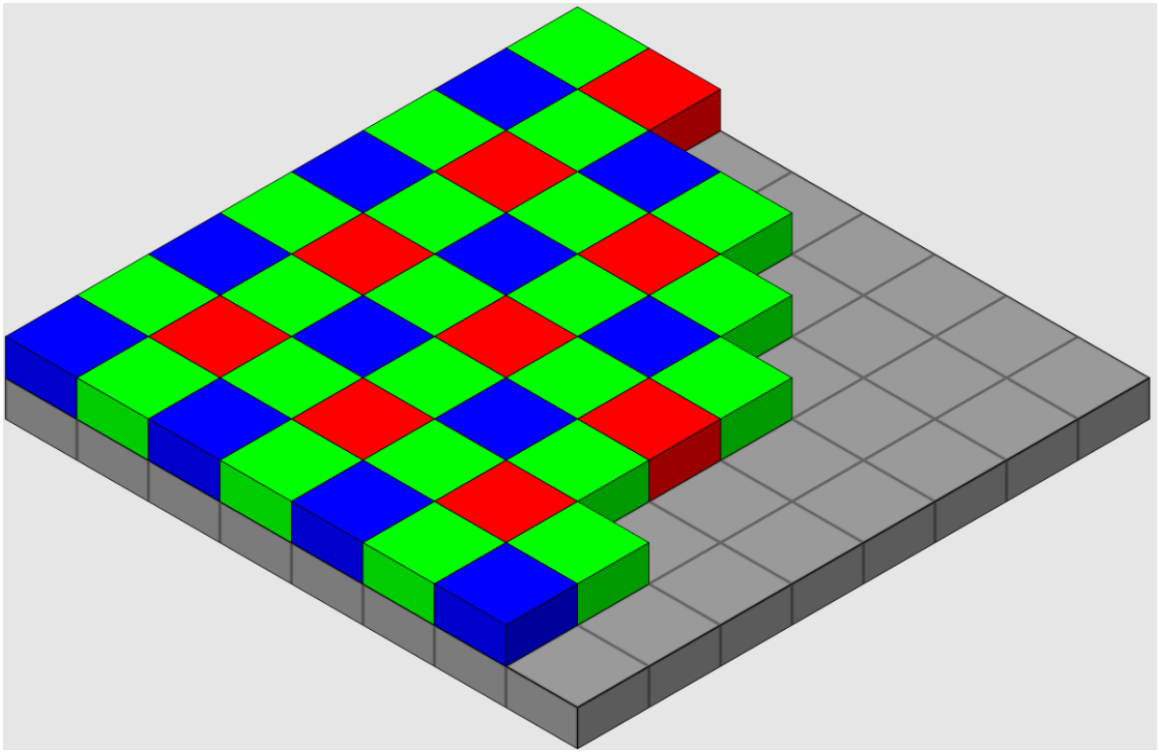
Example 1: The left image depicts the original image while the right picture shows the results after the normalization process. We can see that the original image is too dark while the normalized image is not.

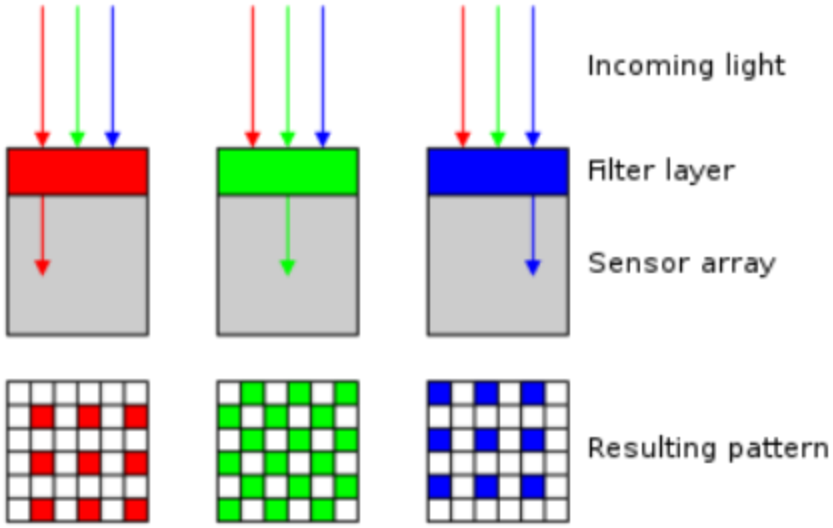
Example 2: The left image depicts the original image while the right picture shows the results after the normalization process. We can see that the original image is very bright and difficult to observe by the human eye, while the resulting image presents a better contrast.

2.2 DEBAYERING

Color Filter Arrays:

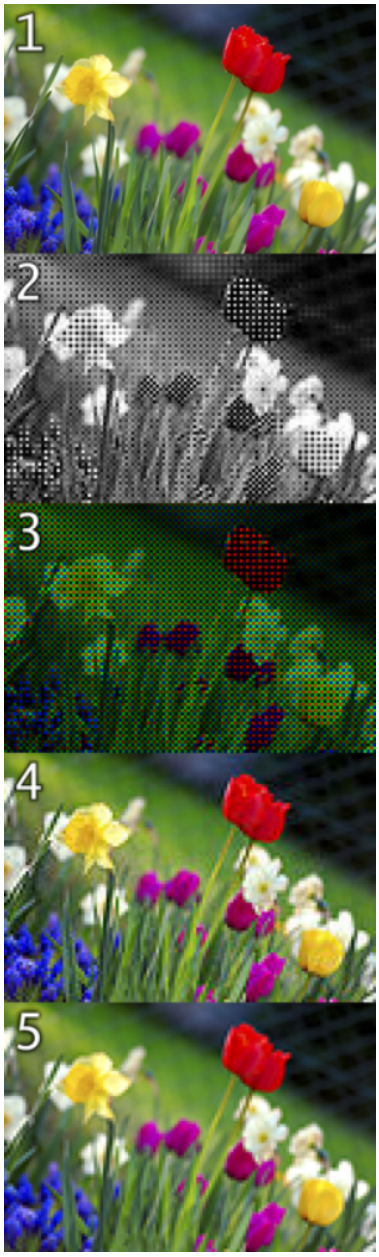
- It is an array of tiny color filters placed before the image sensor array of a camera.
- The resolution of this array is the same as that of the image sensor array.
- Each color filter may allow a different wavelength of light to pass
- This is predetermined during the camera design.
- The most common type of CFA is the Bayer pattern which is shown below:





Profile/Cross-section of sensor

- The Bayer pattern collects information at red, green, and blue wavelengths only as shown above.
- The Bayer pattern uses twice the number of green elements as compared to red or blue elements.
- This is because both the M and L cone cells of the retina are sensitive to green light.
- The raw (uncompressed) output of the Bayer pattern is called the Bayer pattern image or the mosaiced image.
- The mosaiced image needs to be converted to a normal RGB image by a process called color image demosaicing.

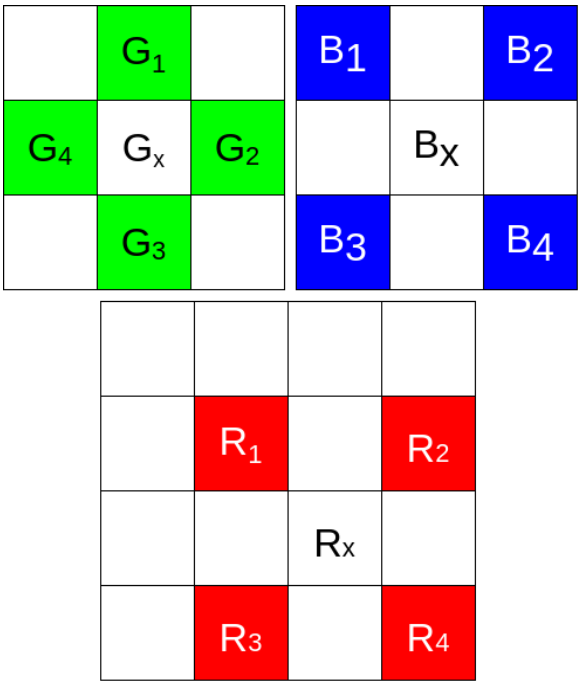


1) Original scene
2) Output of a 120×80-pixel sensor with a Bayer filter
3) Output color-coded with Bayer filter colors
4) Reconstructed image after interpolating missing color information
5) Full RGB version at 120×80-pixels for comparison (e.g. as a film scan, Foveon, or pixel shift image might appear)

<u>Demosaicing algorithm</u>:

- Demosaicing involves the interpolation of missing color values from nearby pixels.
- There exist a plethora of demosaicing algorithms.
- We will be seeing two algorithms here, Bilinear interpolation and its modified version, Malvar-He-Cutler Linear Image demosaicing.

Bilinear Interpolation:



- Bilinear interpolation is the easiest method we can use to demosaic a Bayer image.
- The idea behind this method is that since there is a high probability that the value of a missed pixel has a similarity to the value of its existing adjacent pixels, we can interpolate the missed values in each channel by taking the average of its adjacent pixels.
- In other words, we start from the red channel, and for any missed values, we take a look over its adjacent pixels. If they

contain a value, we take their average and assign the calculated average to the missed pixel.

As shown above, we can use the following equations to interpolate the values of Gx, Bx, and Rx:

$$R_x = \frac{1}{4} \times (R_1 + R_2 + R_3 + R_4) \quad B_x = \frac{1}{4} \times (B_1 + B_2 + B_3 + B_4)$$

$$G_x = \frac{1}{4} \times (G_1 + G_2 + G_3 + G_4)$$

Malvar-He-Cutler Algorithm:

- The idea behind high-quality interpolation is that for interpolating the missing pixels in each channel, it might not be accurate to use only the adjacent pixels located on the same channel.
- In other words, for interpolating a green pixel such as Gx in Fig we need to use the value of its adjacent green pixels as well as the value of the existing channel.
- For example, if at the location of Gx, we have a red value, we have to use that value as well as the adjacent available green values. They called their method gradient correction interpolation.
- Finally, they came up with 8 different 5*5 filters. We need to convolve the filters to pixels that we want to interpolate.
- The method is derived as a modification of bilinear interpolation. Let R, G, and B denote the red, green, and blue channels. At a red or blue pixel location, the bilinear

interpolation of the green component is the average of its four axial neighbors,
$\hat{G}_{bl}(i, j) = 1/4 (G(i - 1, j) + G(i + 1, j) + G(i, j - 1) + G(i, j + 1))$.

- Bilinear interpolation of the red and blue components is similar but uses instead the four diagonal neighbors.
- The visual quality of bilinear demosaicing is generally quite poor. Since the channels are interpolated independently, the misalignments near the edges produce strong color distortions and zipper artifacts.
- To improve upon the quality of the bilinear method, Malvar, He, and Cutler follow the work of Pei and Tam by adding Laplacian cross-channel corrections.
- Consider the case of finding G at an R or a B pixel.
- If the actual R-value differs considerably from the linearly interpolated R-value, it means that there is a sharp luminance change at that pixel.
- The green component at a red pixel location is estimated as
  $G(i, j) = \hat{G}_{bl}(i, j) + \alpha \Delta R(i, j)$,
  where ΔR is the discrete 5-point Laplacian of the red channel,
  $\Delta R(i, j) := R(i, j) - 1/4 (R(i - 2, j) + R(i + 2, j) + R(i, j - 2) + R(i, j + 2))$.
- To estimate a red component at a green pixel location,
  $\hat{R}(i, j) = \hat{R}_{bl}(i, j) + \beta \Delta G(i, j)$
  where ΔG is a discrete 9-point Laplacian of the green channel.
- To estimate a red component at a blue pixel location,
  $\hat{R}(i, j) = \hat{R}_{bl}(i, j) + \gamma \Delta B(i, j)$,

where **ΔB** is the discrete 5-point Laplacian of the blue channel.

- By symmetry, blue components are estimated in a manner similar to the estimation of red components. These formulas are shown in more detail in the next section.
- The parameters α, β, and γ control the weight of the Laplacian correction terms. To set these parameters optimally, the values producing the minimum mean squared error over the Kodak image suite were computed. These values were then rounded to dyadic rationals to obtain **$\alpha = 1/2$, $\beta = 5/8$, $\gamma = 3/4$.**
- The advantage of this rounding is that the filters may be efficiently implemented with integer arithmetic and bit shifting. The filters approximate the optimal Wiener filters within 5% in terms of mean squared error for a 5 × 5 support.
- 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.
- For example, at a green pixel location in a red row, the red component is interpolated by

$$
\begin{aligned}
R(i,j) = \tfrac{1}{8}\,(& & \tfrac{1}{2}\,G(i,j-2) & & \\
& -G(i-1,j-1) & & -G(i+1,j-1) & \\
-G(i-2,j) & + 4R(i-1,j) & + 5G(i,j) & + 4R(i+1,j) & + -G(i+2,j) \\
& -G(i-1,j+1) & & -G(i+1,j+1) & \\
& & +\tfrac{1}{2}\,G(i,j+2) & &).
\end{aligned}
$$

- The filters can be implemented using integer arithmetic. Suppose that the input CFA data is given as a 2D integer

array F (i, j), then the interpolation above can be implemented as

- **R(i, j) = (F (i, j – 2) + F (i, j + 2)+ –2(F (i – 1, j – 1) + F (i + 1, j – 1) + F (i – 2, j) + F (i + 2, j) + F (i – 1, j + 1) + F (i + 1, j + 1))+ +8(F (i – 1, j) + F (i + 1, j))+ +10F (i, j))/16,**
- and the division by 16 can be efficiently implemented by bit shifting. The other filters are implemented similarly.

G at red locations

		−1		
		2		
−1	2	4	2	−1
		2		
		−1		

G at blue locations

		−1		
		2		
−1	2	4	2	−1
		2		
		−1		

R at green locations in red rows

		$\frac{1}{2}$		
	−1		−1	
−1	4	5	4	−1
	−1		−1	
		$\frac{1}{2}$		

R at green locations in blue rows

		−1		
	−1	4	−1	
$\frac{1}{2}$		5		$\frac{1}{2}$
	−1	4	−1	
		−1		

R at blue locations

		$-\frac{3}{2}$		
	2		2	
$-\frac{3}{2}$		6		$-\frac{3}{2}$
	2		2	
		$-\frac{3}{2}$		

B at green locations in red rows

		$\frac{1}{2}$		
	−1		−1	
−1	4	5	4	−1
	−1		−1	
		$\frac{1}{2}$		

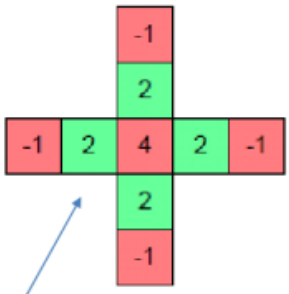
B at green locations in blue rows

		−1		
	−1	4	−1	
$\frac{1}{2}$		5		$\frac{1}{2}$
	−1	4	−1	
		−1		

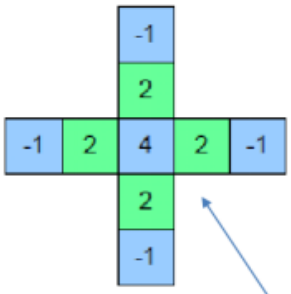
B at red locations

		$-\frac{3}{2}$		
	2		2	
$-\frac{3}{2}$		6		$-\frac{3}{2}$
	2		2	
		$-\frac{3}{2}$		

5 x 5 linear filters



G at R locations



G at B locations

$$\hat{g}(i,j) = \hat{g}_B(i,j) + \alpha\,\Delta_R(i,j)$$

$$\Delta_R(i,j) \triangleq r(i,j) - \frac{1}{4}\sum_{(m,n)=\{(0,-2),\,(0,2),\,(-2,0),\,(2,0)\}} r(i+m, j+n)$$

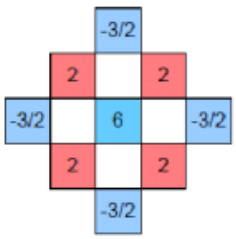
$$\bar{g}(i,j) = \bar{g}_B(i,j) + \gamma\Delta_B(i,j)$$

$$\Delta_B(i,j) = b(i,j) - \frac{1}{4}\sum_{(m,n)=\{(0,-2),(-2,0),(2,0),(0,2)\}} b(i+m, j+n)$$

- To obtain a R value at a B pixel, the corresponding formula is

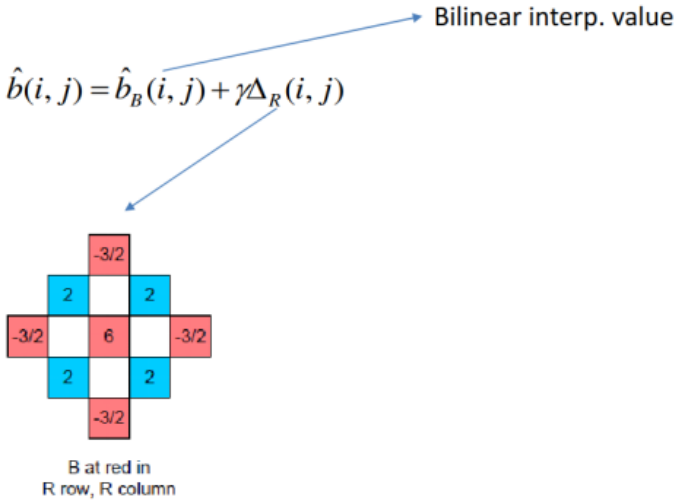
Bilinear interp. value

$$\hat{r}(i,j) = \hat{r}_B(i,j) + \gamma\,\Delta_B(i,j)$$

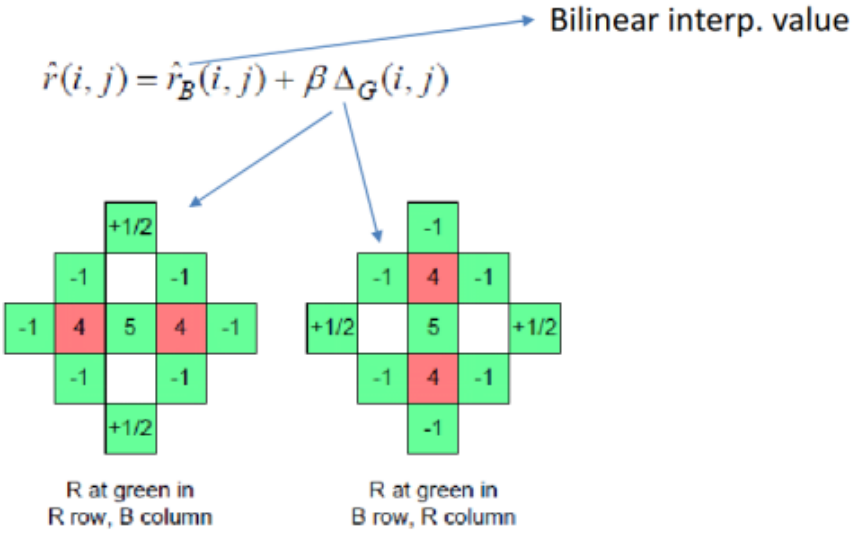


R at blue in
B row, B column

- To obtain a B value at a R pixel, the corresponding formula is



- To obtain a R value at a G pixel, the corresponding formula is



2.3 WHITE BALANCE:

- Any object can look like any color, depending on the light illuminating it. To reveal the color that we would see as humans, what we need is a reference point, something we know should be a certain color (or more accurately, a certain chromaticity). Then, we can rescale the R, G, and B values of the pixel until it is that color.
- As it is usually possible to identify objects that should be white, we will find a pixel we know should be white (or gray), which we know should have RGB values all equal, and then we find the scaling factors necessary to force each channel's value to be equal.
- As such, this rescaling process is called white balancing.
- Once we do this for a single pixel, we will assume that the same illuminant is lighting the entire scene and use these scaling values for all pixels in the image.

Gray world assumption -

- The Gray World Assumption 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 assumed to be the color of the light. Therefore, we can estimate the illumination color cast by looking at the average color and comparing it to gray.
- Gray world algorithm produces an estimate of illumination by computing the mean of each channel of the image.

- One of the methods of normalization is normalizing to the maximum channel by scaling by r

$$r_i = \frac{max(avg_R, avg_G, avg_B)}{avg_i}$$

2.4 Auto Adjustment

- Brightness and contrast are linear operators with parameter alpha and beta
- O(x,y) = alpha * I(x,y) + beta
- Looking at the histogram, alpha operates as a color range amplifier, beta operates as a range shift.
- Automatic brightness and contrast optimization calculate alpha and beta so that the output range is 0..255.
  input range = max(I) - min(I) wanted output range = 255;
  alpha = output range / input range = 255 / (max(I) - min(I))
- You can calculate beta so that min(O) = 0;
  min(O) = alpha * min(I) + beta
  beta = -min(I) * alpha

2.5 Auto Exposure

- If too much light strikes the image sensor, the image will be overexposed, washed out, and faded.
- If too little light reaches the camera sensor produces an underexposed image, dark and lacking in details, especially in shadow areas.
- Image channel having normalized values in the range 0-1 is run through a loop where each pixel value is compared to the mean intensity value of the image and correction is applied accordingly

- If pixel intensity is greater than the mean intensity, then the intensity of the pixel is increased by a factor of 0.6
- If pixel intensity is less than the mean intensity(underexposed), then the intensity of the pixel is increased by a factor of 0.8
- And if pixel intensity is greater than 0.8 (overexposed), then the intensity of the pixel is reduced by a factor of 0.2

2.5 Gamma Correction

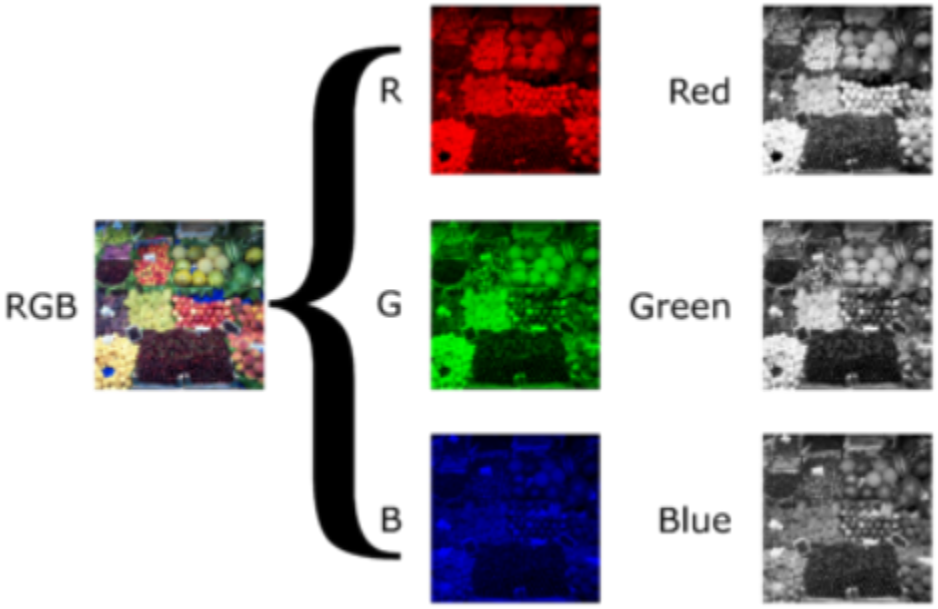
- Gamma correction is also known as the Power Law Transform.
- First, our image pixel intensities must be scaled from the range [0, 255] to [0, 1.0]. From there, we obtain our output gamma-corrected image by applying the following equation:
- O = I^(1 / G)
- Where I is our input image and G is our gamma value. The output image O is then scaled back to the range [0, 255].

COLOR CONVERSION -

2.6 RGB → Grayscale:

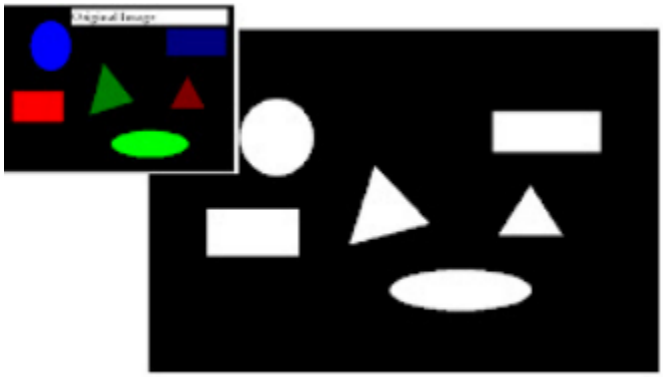
Luminosity Method for conversion

- The best method is the luminosity method which successfully solves the problems of previous methods.
- Based on the observations, we should take a weighted average of the components. The contribution of blue to the final value should decrease, and the contribution of green should increase.
- After some experiments and more in-depth analysis, researchers have concluded in the equation below:
  Grayscale = (0.3 * R + 0.59 * G + 0.11* B)/3
- Here most weight is given to green-colored pixels as humans are said to perceive green light well.



2.7 RGB → Binary:

- Binary images are images whose pixels have only two possible intensity values. They are normally displayed in black and white.
- Numerically, the two values are often 0 for black, and either 1 or 255 for white.
- Binary images are often produced by thresholding a grayscale or color image, to separate an object in the image from the background.
- The color of the object (usually white) is referred to as the foreground color. The rest (usually black) is referred to as the background color. However, depending on the image, which is to be thresholded, this polarity might be inverted, in which case the object is displayed with 0 and the background is with a non-zero value.
- Some morphological operators assume a certain polarity of the binary input image so that if we process an image with inverse polarity the operator will have the opposite effect. For example, if we apply a closing operator to black text on white background, the text will be opened.



2.8 RGB → HSV:

HSV – (hue, saturation, value), also known as HSB (hue, saturation, brightness), is often used by artists because it is often more natural to think about a color in terms of hue and saturation than in terms of additive or subtractive color components. HSV is a transformation of an RGB colorspace, and its components and colorimetry are relative to the RGB colorspace from which it was derived.

Algorithm

- Divide r, g, b by 255
- Compute cmax, cmin, and difference
- Hue calculation:

if cmax and cmin equal 0, then h = 0

if cmax equal r then compute h = (60 * ((g – b) / diff) + 360) % 360

if cmax equal g then compute h = (60 * ((b – r) / diff) + 120) % 360

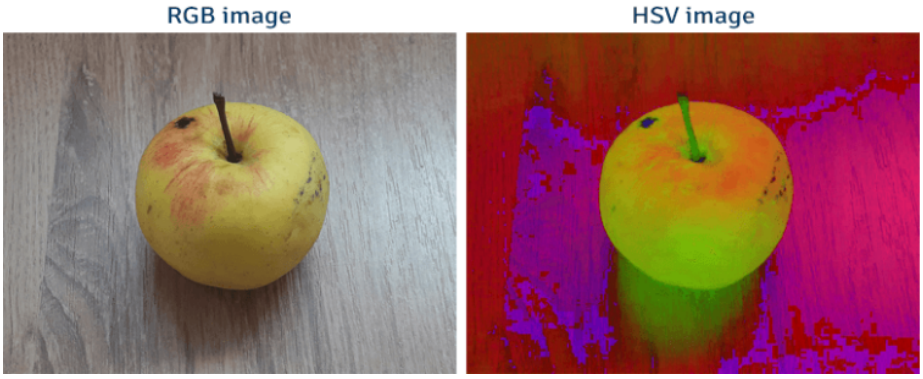
if cmax equal b then compute h = (60 * ((r – g) / diff) + 240) % 360

- Saturation computation:

if cmax = 0, then s = 0

if cmax does not equal 0 then compute s = (diff/cmax)*100

- Value computation:
- v = cmax*100



MORPHOLOGICAL TRANSFORMATIONS

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, and the second one is called the structuring element or kernel which decides the nature of the operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient, etc also come into play.

2.9 EROSION:

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of the foreground object (Always try to keep the foreground white). So what it does do? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel are 1, otherwise, it is eroded (made to zero). So, what happens is that all the pixels near the boundary will be discarded depending upon the size of the kernel. So, the thickness or size of the foreground objects decreases, or simply the white region decreases in the image. It is useful for removing small white noises, detaching two connected objects, etc.

2.10 DILATION:

It is just the opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So, it increases the white region in the image or the size of the foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because erosion removes white noises, it also shrinks our objects. So, we dilate it. Since the noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

2.11 OPENING:

Opening is just another name for erosion followed by dilation. It is useful in removing noise, as we explained above.

2.12 CLOSING:

Closing is the reverse of Opening, with Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object.

2.13 MORPHOLOGICAL GRADIENT:

It is the difference between the dilation and erosion of an image. The result will look like the outline of the object.

2.14 TOP HAT:

It is the difference between the input image and the Opening of the image. The below example is done for a 9x9 kernel.

2.15 BLACK HAT:

It is the difference between the closing of the input image and the input image.

EDGE DETECTION

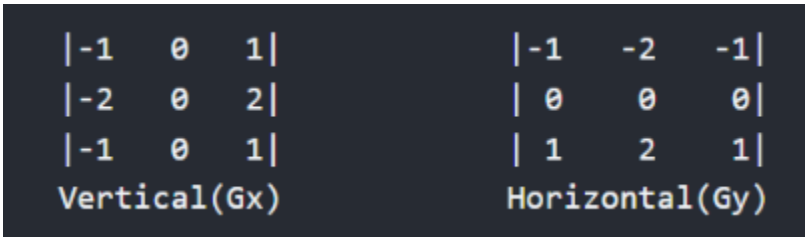
An edge in an image is a significant local change in the image intensity. As the name suggests, edge detection is the process of detecting the edges in an image.

Why do we need Edge Detection?

- Discontinuities in depth, surface orientation, scene illumination variations, and material properties changes lead to discontinuities in image brightness.
- We get the set of curves that indicate the boundaries of objects and surface markings and curves that correspond to discontinuities in surface orientation.
- Thus, applying an edge detection algorithm to an image may significantly reduce the amount of data to be processed and may therefore filter out information that may be regarded as less relevant while preserving the important structural properties of an image.

Sobel Edge Detection:

- The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically, it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.
- In theory, at least the operator consists of a pair of 3×3 convolution kernels. One kernel is simply the transposition of another.
- These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, with one kernel for each of the two perpendicular orientations.
- The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these Gx and Gy).



- When the vertical filter is applied, vertical edges become more prominent while horizontal edges get more prominent on applying a horizontal filter
- These can then be combined to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:
  $|G| = (Gx^2 + Gy^2)^{0.5}$

  Typically, an approximate magnitude is computed using:

|G| = |Gx| + |Gy|
which is much faster to compute.

Blurring

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It removes high-frequency content (e.g.: noise, edges) from the image. So, edges are blurred a little bit in this operation (there are also blurring techniques that don't blur the edges).

2.16 AVERAGING:

This is done by convolving an image with a normalized box filter. It simply takes the average of all the pixels under the kernel area and replaces the central element.
We should specify the width and height of the kernel.
A 3x3 normalized box filter would look like the below:

|1/9 1/9 1/9|
|1/9 1/9 1/9|
|1/9 1/9 1/9|

2.17 GAUSSIAN BLUR:

In the Gaussian Blur operation, the image is convolved with a Gaussian filter instead of the box filter. The Gaussian filter is a low-pass filter that removes high-frequency components.
In 2-D, an isotropic (i.e. circularly symmetric) Gaussian has the form:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution.

2.18 MEDIAN BLUR:

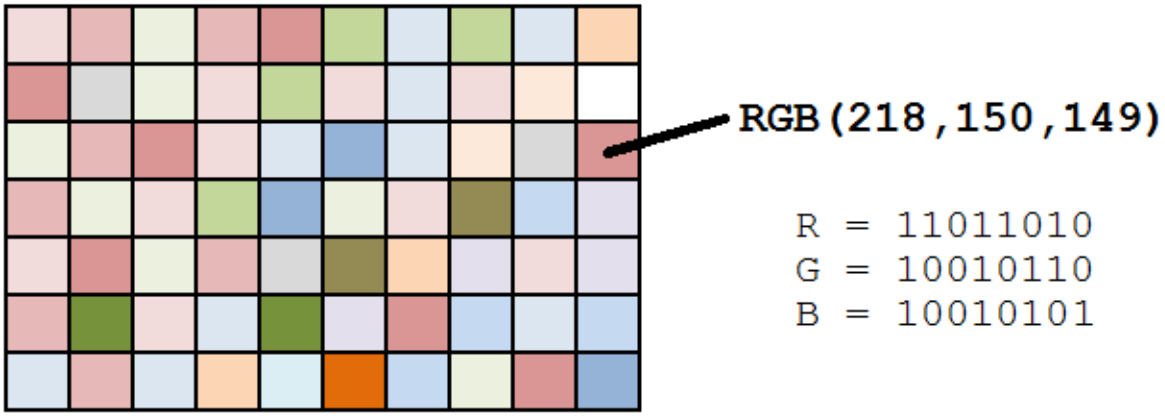
Here, the function takes the median of all the pixels under the kernel area and the central element is replaced with this median value.

This is highly effective against salt-and-pepper noise in an image. Interestingly, in the above filters, the central element is a newly calculated value which may be a pixel value in the image or a new value. But in median blurring, the central element is always replaced by some pixel value in the image. It reduces the noise effectively.

Its kernel size should be a positive odd integer.

2.19 ROTATION:

A typical computer image these days uses 24 bits to represent the color of each pixel. Eight bits are used to store the intensity of the red part of a pixel (00000000 through 11111111), giving 256 distinct values. Eight bits are used to store the green component, and eight bits are used to store the blue component.



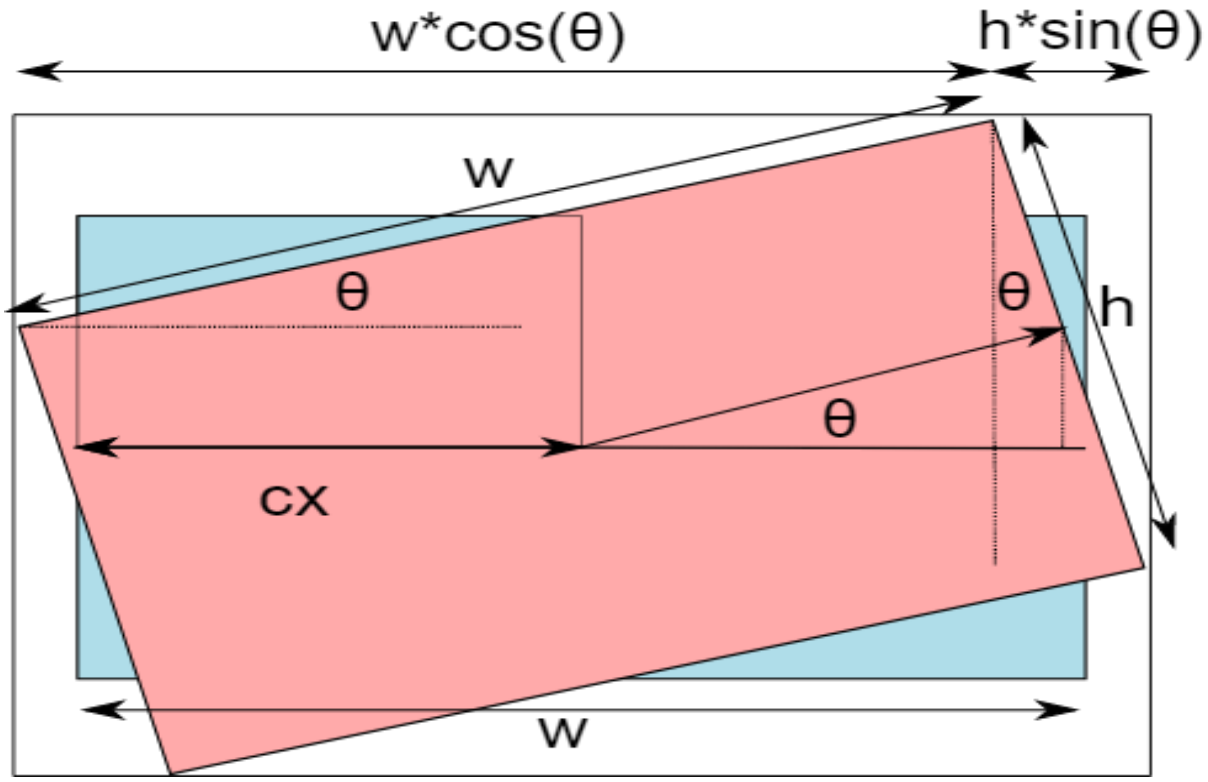
Each pixel has a coordinate pair (x,y) describing its position on two orthogonal axes from defined origin O.It is around this origin we are going to rotate this image. What we need to do is take the RGB values at every (x,y) location, rotate it as needed, and then write these values in the new location, considering (x,y) with respect to the origin we assumed. The new location is obtained by using the transformation matrix.

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

When image will be rotated ,the dimension of the frame containing it will change so to find the new dimensions we use this formula,

$$new\_width = |old\_width \times cos\theta| + |old\_height \times sin\theta|$$

$$new\_height = |old\_height \times cos\theta| + |old\_row \times sin\theta|$$



THREE SHEARS

This method works by expanding the 2D rotation matrix into three different matrices.

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin\theta & 1 \end{bmatrix} \begin{bmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

There are some interesting properties of this matrix:-

1. The three matrices are all shear matrices
2. The first and the last matrices are the same
3. The determinant of each matrix is the same (each stage is conformal and keeps the area the same).

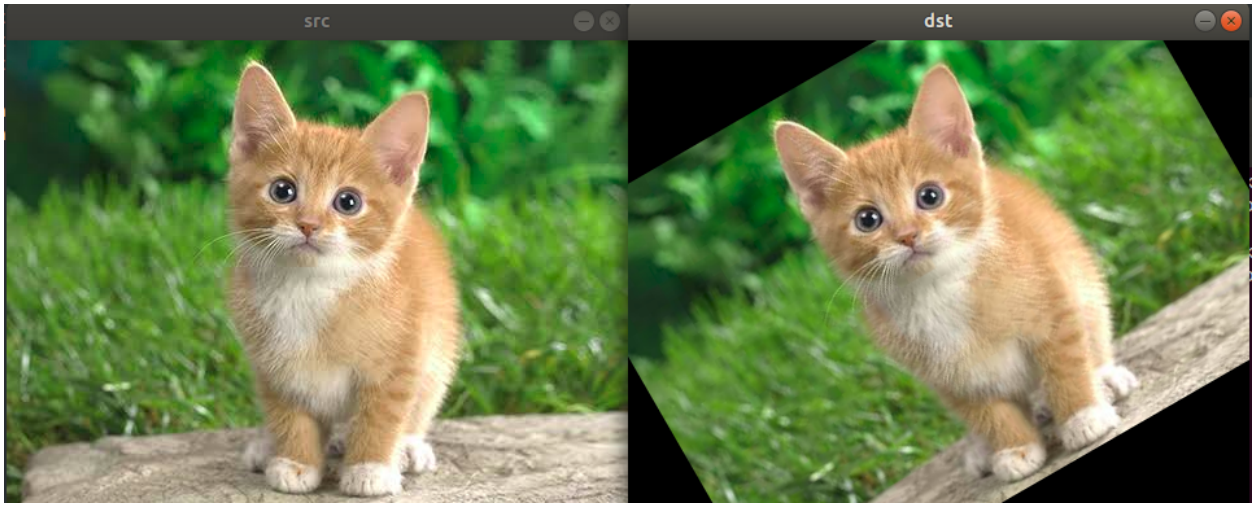
4. As the shear happens in just one plane at a time, and each stage is conformal in area, no aliasing gaps appear in any stage.

Note: Shear matrix shown above rotates in a clockwise direction so we need to take an angle in negative values to assess for that.

In the first shear operation, raster columns are simply shifted up and down relative to each other. The shearing is symmetric around the center of the image. It's analogous to shearing a deck of playing cards.

The second shear operation does a similar thing to the previous image, but this time does the shearing left to right.

The final shear is the same as the first operation; this time applied to the intermediate image.



3. IMPLEMENTATION

3.1 Reading the image:

To work with raw images, we must first use other pieces of software to convert the camera-manufacturer-specific formats and obtain the sweet image data found inside.

There is a fantastic piece of cross-platform, open-source software, written by Dave Coffin, called dcraw (pronounced dee-see-raw). This is the open-source solution for reading dozens of different types of RAW file and outputting an easily read PPM or TIFF file (that Python's Pillow package can directly read data from). Many open-source image editing suites incorporate this program as their own RAW-reading routine. It can read file from hundreds of camera models and perform many standard processing steps to take a RAW file and generate an attractive output.

Run the following command (again, you can run "dcraw" to get an idea what the following flags represent):

```
dcraw -4 -d -v -T <raw_file_name>
```

You will see the following formatted output:

Scaling with darkness <black>, saturation <white>,
and multipliers <r_scale> <g_scale> <b_scale> <g_scale>

The program is run using the following command:

```
../bin/working <gamma_value> <file_name>
```

read_image.py is used to read the .tiff image file so generated. Image from PIL module is used to open the image and then the image is converted into a numpy array. The numpy array is then stored into a txt file PixelData.txt

File handling is used in C++ to read the content of PixelData.txt file.

image is Vector of vector of double (matrix) having height x width dimensions used to store the pixel values.

3.2 Pre-processing:

First step is to perform debayering by using the debayering() function. First normalization of image is done using scale() function which normalizes the image to [0,1], then demosaicing operation is performed by separating the three color channels and then those channels are color corrected before combining them into a single channel.

whiteBalance() function returns the white balanced image.

gammaCorrection() returns the gamma corrected image by applying a nonlinear function to the brightness values to make them more perceptually pleasing.

AutoExposure() functions do automatic calculation of brightness and contrast.

Since the functions operate on single channels of the image, we then combine the 3 color channels. create3DImage() is used to

create a Mat object so that it can be displayed on the screen. Here OpenCV is used to display the image.

The final_image is then scaled back to range [0,255] so that it can be saved as a jpeg/png image file.

3.3 Post-Processing:

Here a menu driven program is provided to choose between the different post-processing functions as per the user needs.
Following is some of the functions implemented:
cvtGray() function converts the RGB image to grayscale by using the gray world algorithm.
cvtBinary() function converts the grayscale image to binary image by using a fixed threshold value, any pixel value lower than it will be assigned 0 otherwise 1.
rgb2hsv() function converts the RGB image to HSV image.
Erosion() function returns the eroded image(pixels near the boundaries are discarded) by using a 3x3 kernel as default otherwise kernel size can also be passed.
Dilation() function returns the dilated image (size of foreground object is increased).
Opening() function does erosion of image first and then its dilation.
Closing() function does dilation of image first and then its erosion.
Gradient() function returns the difference image between the dilated image and eroded image.
TopHat() function returns the difference between the input Image and opening of the image.

BlackHat() function returns the difference between the Closing of the image and dilated image.
edgeDetect() function returns the edges in an image by calculating gradients in x and y directions (Sobel Edge Detection).
meanFilter(), gaussianBlur() functions return the blurred image. Default kernel size is 3.
Rotate() returns the rotated image by using the three shears method.

4. CONCLUSION AND FUTURE WORK

4.1 Conclusion

Hence we successfully explored this domain of Image Processing and made a working Image Pipeline that takes a RAW image as input and produces a processed image with algorithms like Debayering, Auto White Balance, Auto Exposure, Gamma Correction, etc applied to it.

RAW Image	Preprocessed Image
	
	

4.2 Future Aspects

We enjoyed working on this project, and got to know more about image representation. We will try to -

1) Implement the library functions of OpenCV.
2) Extend the functionality to dynamic images (videos) .

5. REFERENCES

Useful links & Research Papers / Helpful Courses

[1] Link for downloading dcraw:

https://cs.brown.edu/courses/csci1290/labs/lab_raw/index.html

[2] White Balance ALgorithm:

https://www.codeproject.com/Articles/653355/Color-Constancy-Gray-Wowwrld-Algorithm

[3] Gamma Correction:

https://www.cambridgeincolour.com/tutorials/gamma-correction.htm

[4] RGB → HSV:

https://www.had2know.org/technology/hsv-rgb-conversion-formula-calculator.html

[5] Morphological operations:

https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

[6] Sobel Edge Detection:

https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection

[7] Blurring:

https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

[8] Auto Exposure:
https://link.springer.com/article/10.1007/s11042-019-08318-1