








Ahmedabad  
University

# SIGNALS AND SYSTEMS – ECE 210

## PROJECT REPORT

Name	Enrollment Number	Photo
Anshul Mehta	AU1940275	
Kavan Desai	AU1940126	
Harsh Patel	AU1940114	 Patel Harsh C. Dt.10-09-2018
Sarthak Bharad	AU1940176	 Bharad Sarthak D. Dt.10-09-2018
Nihal Aggarwal	AU1940217	

# **Image Processor with Inpainting and Colour Filtering Functionalities**

## **Synopsis:**

Image processing is an integral part of Digital Signal Processing. Image is nothing but a signal in two dimensions. It is generally a 2-D array of discrete signal samples. In today's day and age of technology image processing has become extremely important. Image processing basically means to perform some operations on the image at microscale level of pixels so that an enhanced image with greater clarity can be obtained. It is clinical task in a lot of domains to get crucial information from images and that's where restoration and enhancement of comes in. From human interpretation to Computer Vision it plays a very important role. In our image processor we will be primarily focusing on two functions of Image Inpainting and Colour Filtering.

### **1. Image Inpainting:**

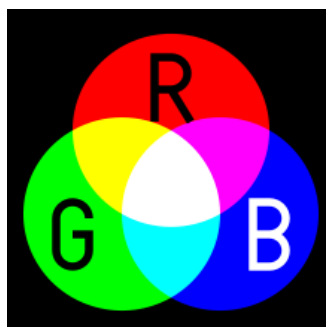
Image inpainting is the process of fixing and restoring damaged images. It helps us in mending scratches, noise and damages from the image and restoring it. Images can be damaged in a lot of ways like dust on lens, scratch on an image, some noise that has crept in the image. We aim to remove the noise and damage and restore the image by using the techniques of digital signal processing. So it is basically the process of filling out the undesirable or corrupted portions of the image with something relevant and for that we use the information of the area surrounding the region of concern and then do some procedures in order to fill in.

We are going to use two approaches of the plethora of options available:

- 1) Digital Inpainting by a modified convolution method
- 2) Fast Marching Method in Fast Fourier Transform Domain

### **2. Colour Filtering:**

Colour Filtering is primarily done by RGB manipulation. The fundamental of the image is derived from three additive primary Colours Red Green and blue. An image comprises of component values of these primary Colours to produce a broad array of Colours that constitute a new image. When any one of the components is dominant or say a stronger intensity then that Colour tends to be a primary Colour. When two colours have the same strong intensity or have strong components then they form secondary colours like cyan, magenta and yellow. Zero component for each RGB would result in absolute darkest Colour which in our code would result in Colourless image. When all the primary colours contain strong components then we would obtain the original input image.



## Techniques used and Flowcharts

### Image Inpainting

#### **1) Fast Marching Method in Fourier Transform Domain:**

The output of the transformation represents the image in the *Fourier* or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

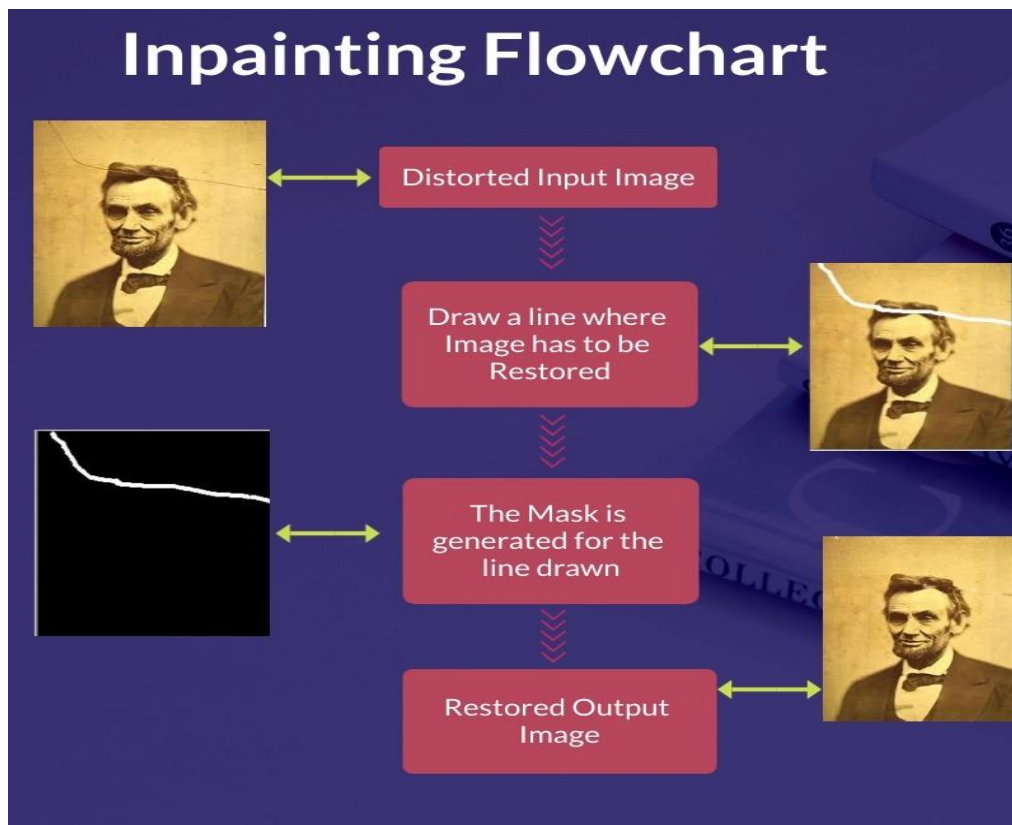
It is well known that for compressing an image, one can save the Fourier coefficients associated with the image function and then reconstruct it by inverse Fourier transform. It is also a well known property that a lot of the image information in the transfer domain is sparse (In numerical analysis and scientific computing, a sparse matrix or sparse array is a matrix in which most of the elements are zero). Sparsity is exploited in either of DCT or FFT domains.

We capitalize on the fact that for a damaged image the Fourier Coefficients in the sparse matrix are unreliable and they need to be replaced and we can use the information near to image in the transfer domain and replace the Coefficients. So this is primarily a sparsity driven approach with the pixels. It replaces each pixel on which Inpainting has to be performed with a weighted sum of the pixels in the background, with more weight given to nearer pixels and boundary of the pixels to be replaced and lesser and lesser as we move away from the boundary. The Fast Marching Method ensures that the pixels near are inpainted first and thus it works just like a manual heuristic approach.

Which means we duplicate the image information from the region that is alright and put it into the corrupted region at patch level for restoring. A sparsity operator maps the transform Coefficients with amplitudes. The resultant is called the sparse image and the location of zeros is the sparsity pattern.

**The Mask:** This is the replica of image where spatial dimensions are shared by the image and the mask and non-zero pixels correspond to areas that need to be fixed and zero pixels are considered to be normal as the sparsity approach states that the Coefficients in transfer domain are sparse. The sparsity operator adds a redundancy of the original image and the pattern is used side by side to recover the information. So now we have two images in which one is the same image as input and another is the mask that has same dimensions of image with the sparsity pattern underlying the black pixelated image. The mask is used to recover the information as process is done on it. The mask is basically a binary matrix which has the sparsity pattern.

## Methodology and Flowchart:



The process from the flowchart is evident. First the damaged image is taken as an input and reconstructed along with a sparsity pattern in form of a mask. The region is sketched and then they are replaced by the surrounding ones as mentioned in the above process. More sparsity results in more similarity to damaged image and less similarity to original one. This means that DFT which is calculated using the Fast Fourier Transform algorithm has very crucial role to play in giving a sparse matrix and then the procedure of weighted average kicks in.

**Image Inpainting using Modified Convolution:** This algorithm although not used in our approach but has been a discovery to get an optimized and enhanced version of the Fast-Marching algorithm. The Fast-marching algorithm that is an PDE (Partial differential equation) based approach can be realized slightly differently by using a modified convolution method. This also helps in mitigating a slight downside of the Fast- Marching algorithm which makes the image a little blur and thicker by 10 pixels.

In this modified approach let the area on which inpainting is to be done be denoted by  $A$  and the boundary region be denoted by  $DA$ . The first step involves sketching the part on which image inpainting is to be done manually. 3<sup>rd</sup> step involves initializing the  $A$  region by wiping out its' Colour info. 4<sup>th</sup> step involves repeatedly convolving the region that is to be inpainted with the signals of its diffusion plane template.  $DA$  is a one-pixel thick boundary and the number of convolution iterations is independently controlled by a certain threshold on the change of pixel values from the previous value by the user. There are visible results only after 100 iterations which help progress from Del Omega to Omega.

Convolving the image with the averaging filter to compute the weighted averages of pixels' neighbourhoods is the same as isotropic diffusion. The algorithm uses a weighted average method that has a zero weight at the centre. The positives are that the algorithm is really fast and works well

for images which do not have many high contrast edges or high frequency component and thus provides an edge over the aforementioned algorithm.

Mathematically, repeated blurring and diffusion are identical, when an image is blurred, the Colours of each pixel are averaged with a small portion of the Colour from neighbouring pixels and contributes a small part of its Colour to each of its neighbours. Modifying the diffusion kernel or the filter to zero weight at the bottom right corner instead of the centre and making the convolution from the bottom right corner are sample but accurate and play an important role in the inpainting process and this modification has forbidden the need to iterate the convolution operation because the inpainted pixel produced from the above left neighbourhoods pixels (known pixels) , now we don't need to repeated blurring (convolution) because the goal of repetition achieved from the first averaging iteration.

$$\begin{bmatrix} a & b & a \\ b & a & b \\ a & b & 0 \end{bmatrix} \begin{bmatrix} c & c & c \\ c & c & c \\ c & c & 0 \end{bmatrix}$$

## **2) Navier Stokes Method:**

The second method is the Navier Stokes Method. It is a well know approach in fluid dynamics and fluid mechanics. It is the method that has been stated in the documentation of the OpenCV library that has been used to implement the function. It is an essential concept of computational fluid dynamics.

It helps in framing an important method for automatic image inpainting. The approach takes ideas from classical fluid theory. An approximate solution for the inpainting problem is carried out by solving the Navier Stokes' velocity transportation equation along with simultaneous solving of the Poisson problem between the velocity and stream function in the area of concern where restoration needs to be done. This has not been discussed in great detail as it is not directly linked to subtle concepts of Signals and Systems or Digital Signal Processing.

## **Colour Filtering**

In our User Interface we input an image in jpg or png format. In our code we have inculcated the Colour filter by varying the RGB component of the input image. We have defined eight functions which comprises of primary and secondary Colours which could be applied on the input image from our User Interface to apply the Colour filter. We namely have used primary Colours like Red, Blue, Green. Secondary Colours like Yellow, Purple, Cyan, and other Colours such as Grey and Orange. The component wise breakdown of each Colour is shown in the below table:

Colours	Red Component	Blue Component	Green Component	Output Component
Red	R	0	0	R+0+0
Blue	0	B	0	0+B+0
Green	0	0	G	0+0+G
Yellow	R	0	G	R+0+G
Purple	R	B	0	R+B+0
Cyan	0	G	B	0+G+B
Grey	R+G+B/3	R+G+B/3	R+G+B/3	R+G+B
Orange	2R	0	0	2R+0+0
Original Image	R	G	B	R+G+B

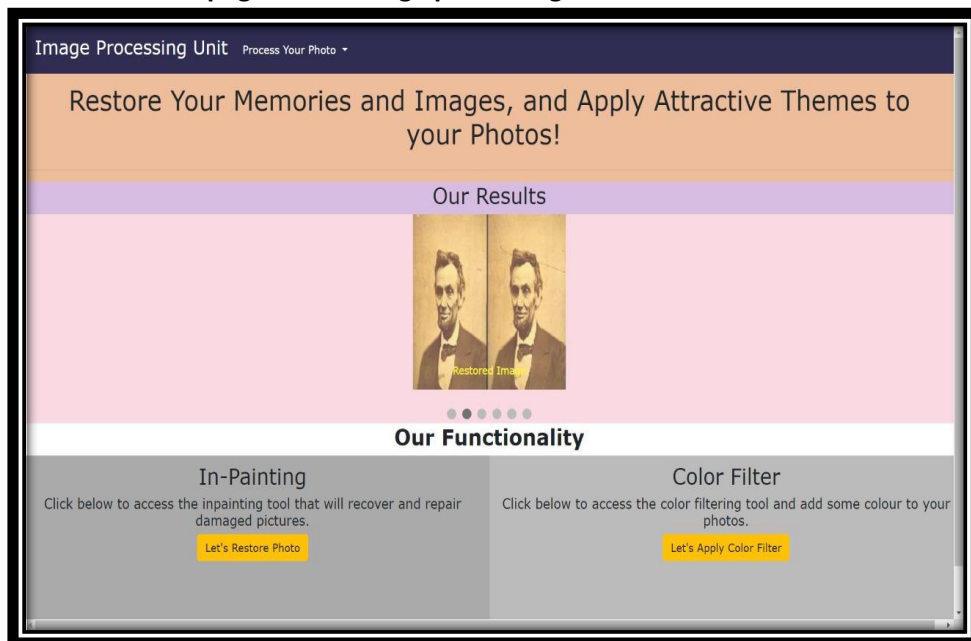
For example, user selects to filter the image in red Colour then the following function is called:

```
def red(r,g,b):  
  
    newr=r  
    newg=0  
    newb=0  
    return(newr,newg,newb)
```

## Interface

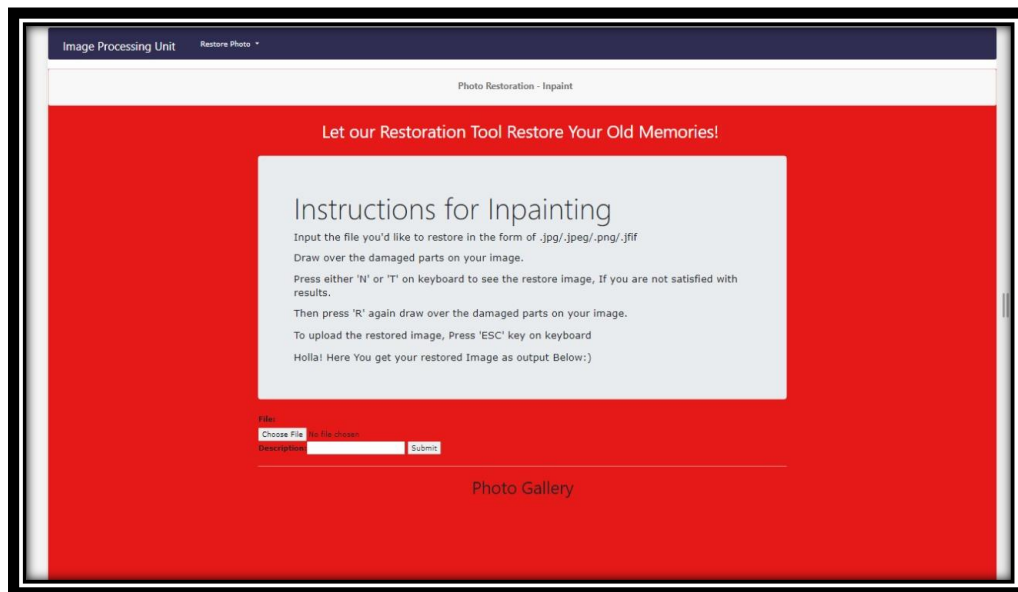
The interface has been created by implementation of our python script along with Django framework to code the backend of the user interface. We have used Django to create a website on a local server. When we run the code, we are redirected to the server webpage from where we can see the following homepage:

🚦 This is the Homepage of our image processing tool.

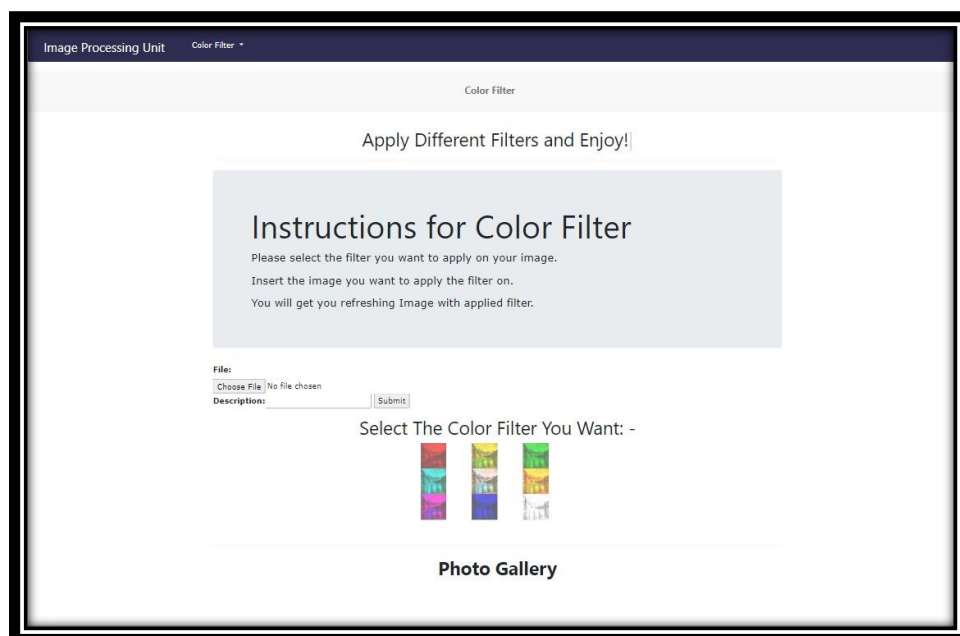


- Some Sample results are displayed as a b-roll on the Homepage in the centre.
- On the top Left -Corner you can see a dropdown from where you can select the functionalities.
- Either of the Functionalities can also be selected from below where buttons that redirect to the respective functionality's page are given.

- After selecting the Inpainting option the user is directed to the Page where inpainting tool can be accessed from. There the user needs to follow the instructions as mentioned on the page and then upload the image. Highlight the part that needs to be corrected and enter submit. This will directly fetch results for Inpainting.



- If the user clicks on Colour Filter then the user is redirected to the Colour Filter tool. There the user can select the Colour/theme that needs to be applied to the photo and after clicking on the submit button the webpage will display the desired Colour filter applied to the photograph.



## Program

These are code snippets primarily for the 2 functions of the image processor. The entire code is not uploaded here due to size constraints.

### Colour Filter:

```
def grey(r,g,b):
    newr=r+g+b//3
    newg=r+g+b//3
    newb=r+g+b//3
    return(newr,newg,newb)

def orange(r,g,b):
    newr=r+r
    newg=g
    newb=0
    return(newr,newg,newb)

img= Image.open("imageG.jfif").convert("RGB")

width,height = img.size
pixels= img.load()

for py in range(height):
    for px in range(width):
        r,g,b=img.getpixel((px,py))
        pixels[px,py]=orange(r,g,b)

outfile=img.show()
outfile= img.save("output.jpg")
```



## Inpainting code snippets:

```
import numpy as np
import cv2 as cv
import sys
class Sketcher:
    def __init__(self, windowname, dests, colors_func):
        self.prev_pt=None
        self.windowname=windowname
        self.dests=dests
        self.colors_func=colors_func
        self.dirty= False
        self.show()
        cv.setMouseCallback(self.windowname, self.on_Mouse)

    def show(self):
        cv.imshow(self.windowname, self.dests[0])
        cv.imshow(self.windowname+"Masks", self.dests[1])

    def on_Mouse(self, event, x, y, flags, param):
        pt=( x, y)

        if event == cv.EVENT_LBUTTONDOWN:
            self.prev_pt=pt

        elif event == cv.EVENT_LBUTTONUP:
            self.prev_pt=None

        if self.prev_pt and flags & cv.EVENT_FLAG_LBUTTON:
            for dst,color in zip(self.dests,self.colors_func()):
                cv.line(dst,self.prev_pt,pt,color,5)
                self.dirty=True
            self.prev_pt=pt
            self.show()
```

```
def main():
    print ("Inpainting Python ")
    print ("Keys: ")
    print ("t- inpainting using Fast Marching method")
    print ("n- Inpainting using NS technique")
    print ("r-reset the mask")
    print ("ESC-exit ")

    img=cv.imread("image2.JPG",cv.IMREAD_COLOR)
    if img is None:
        print ("Failed to import the Image".format(img))
        return

    img_mask=img.copy()

    inpaintMask=np.zeros(img.shape[:2],np.uint8)

    sketch =Sketcher('image',[img_mask,inpaintMask],lambda : ((255,255,255,),255))

    while True:
        ch=cv.waitKey(0)

        if ch==27:
            break
        if ch==ord('t'):
            res=cv.inpaint(src=img_mask,inpaintMask=inpaintMask,inpaintRadius=4,flags=cv.INPAINT_TELEA)
            cv.imshow("Inpaint using Fast March Methodology", res)

        if ch==ord('n'):
            res=cv.inpaint(src=img_mask,inpaintMask=inpaintMask,inpaintRadius=4,flags=cv.INPAINT_NS)
            cv.imshow("Inpaint using NS segmentation", res)

        if ch==ord('r'):
            img_mask[:]=img
            inpaintMask[:]=0
            sketch.show()

    cv.imwrite("Output.jpg",res)
    print("Process Complete")

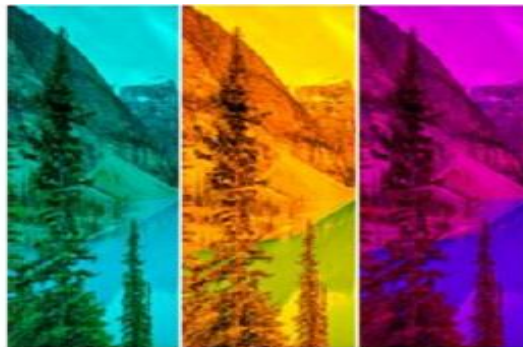
if __name__ == '__main__':
    main()
    cv.destroyAllWindows()
```

## Results

**Results of our inpainting Code:**



**Results of Colour Filter Tool:**



# Conclusions

There can be various uses of Image Inpainting and Colour Filters

## Uses of Image Inpainting

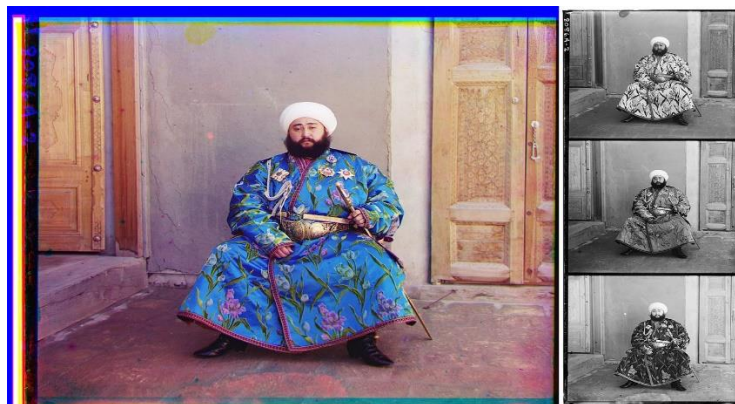
- 1) Image restoration from inpainting is used in correction for microscopic image results.
- 2) Inpainting process is also viable for restoration of physical artwork such as images of ancient paintings, sculptures.
- 3) Modified versions of inpainting algorithms that are much advanced and use Convolution Neural Networks and Machine learning are helping archaeological and geographical departments across the world for restoration purposes.

## Uses of Colour Filter processing

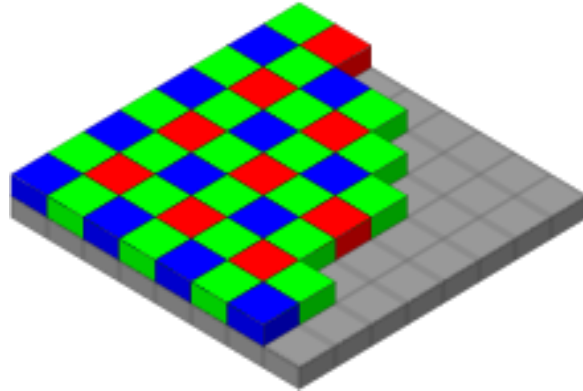
- 1) Colour Filter was used in early Photography where the output image was produced by combining RGB filtered images.
- 2) Early televisions also used Colour filters to transmit the video graphics.
- 3) After the arrival of Virtual Graphic Adapter in personal computers the RGB model became the first choice to display the virtual graphics.
- 4) Colour Filtered images are also used in treatment of ailments in eye visuals and related disorders.
- 5) Photographic Digital cameras also uses the concept of RGB model and images derived from the model outputs an enhanced image.
- 6) Colour Filtered images are also used in scanners, printers, fax machines monitor to interpret an image and process it for best user experience.

## Miscellaneous

- 1) The below photograph was developed by using the three photos of Colours Red, Blue and Green filtered images.



2) Photographic Digital Camera uses image sensors which operate with minor variation of RGB model. The Bayer Filter in this camera have an arrangement of sensors such that number of detectors for green is twice then red and blue (1:2:1 ratio for red, green and blue respectively). This sensor possesses a grid which runs a sequence of RGRGRGRG and GBGBGBGB and the sequence continues. The missing pixels in the grid are obtained by the process of Interpolation. The example of Bayer arrangement grid for image sensor is as follows:



## References and Bibliography

- O. Elharrouss, N. Almaadeeda, S. Al-Maadeeda, and Y. Akbaria, "Image inpainting: A review," 2019. [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/1909/1909.06399.pdf#:~:text=Image%20inpainting%20was%20applied%20on,target%20region%20with%20estimated%20values> [Accessed: 2020].
- A. S. Awati and M. R. Patil, "DIGITAL IMAGE INPAINTING USING MODIFIED CONVOLUTION METHOD.," Dec-2016. [Online]. Available: [https://www.ijesird.com/1\\_December\\_8\\_16.PDF](https://www.ijesird.com/1_December_8_16.PDF) . [Accessed: Nov-2020].
- H. Hosseini, N. B. , Marvasti, and F. Marvast, "Image Inpainting Using Sparsity of the Transform Domain ." [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/1011/1011.5458.pdf> . [Accessed: 2020].
- "Image inpainting with OpenCV and Python," *PyImageSearch*, 28-May-2020. [Online]. Available: <https://www.pyimagesearch.com/2020/05/18/image-inpainting-with-opencv-and-python/> . [Accessed: 08-Dec-2020].
- "Colour Image Processing and Applications." [Online]. Available: [https://www.researchgate.net/publication/243766531\\_Colour\\_Image\\_Processing\\_and\\_Applications](https://www.researchgate.net/publication/243766531_Colour_Image_Processing_and_Applications) . [Accessed: 2020].
- " Analysis of Colour Image Filtering Methods." [Online]. Available: [https://www.researchgate.net/publication/319815286\\_Analysis\\_of\\_Colour\\_Image\\_Filtering\\_Methods](https://www.researchgate.net/publication/319815286_Analysis_of_Colour_Image_Filtering_Methods) . [Accessed: 2020].