Parallel programming project

Juan José Hoyos Urcué - Katherine Camacho Calderón

Context

This project will show a paralell implementation of a library of filters in Python that can be applied to any vIdeo. Each filter allows modifying or improving certain characteristics of an image in order to show a different appearance, some of the objectives of the filters are : Smooth the image, remove noise, enhance edges, and detect edges.

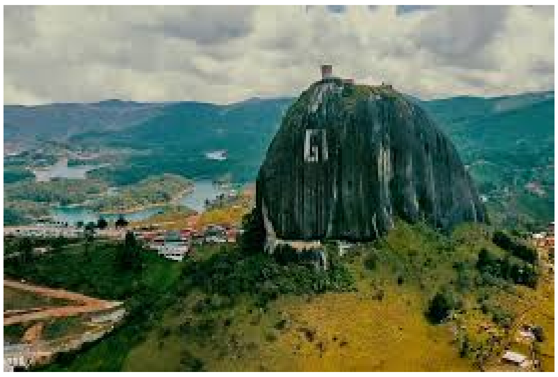
The filters that the library contains are:

- Black & White Filter
- Cartoonizer Filter
- Sketcher Filter
- Negative Filter
- Vintage Filter and
- Cool Filter

Next, an example of the results obtained when applying each of the filters on the original image will be shown

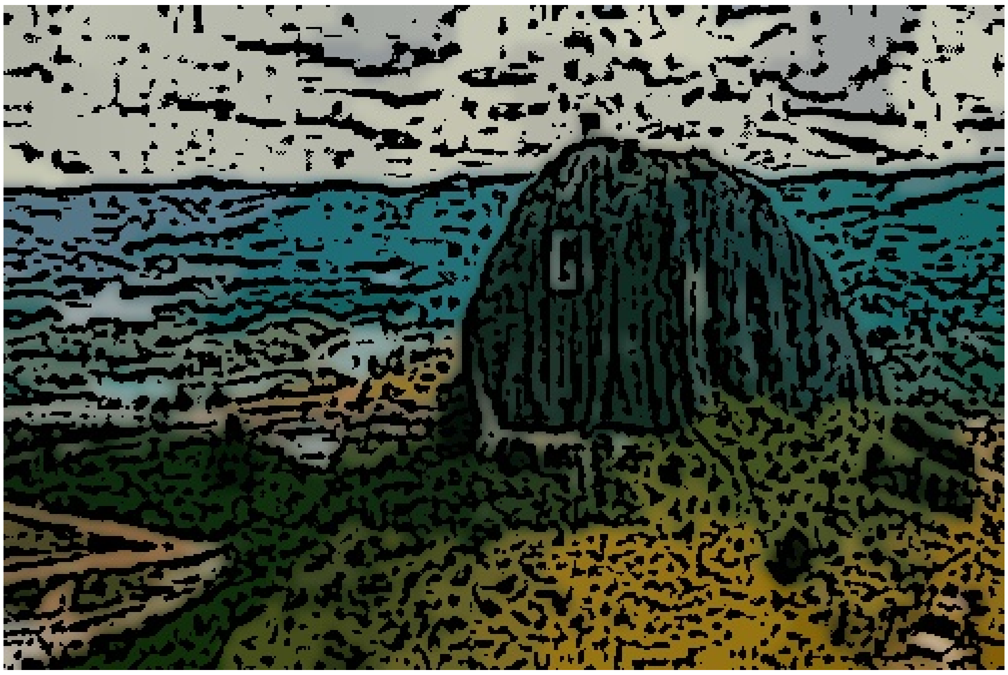
Original Picture

El peñón de Guatapé

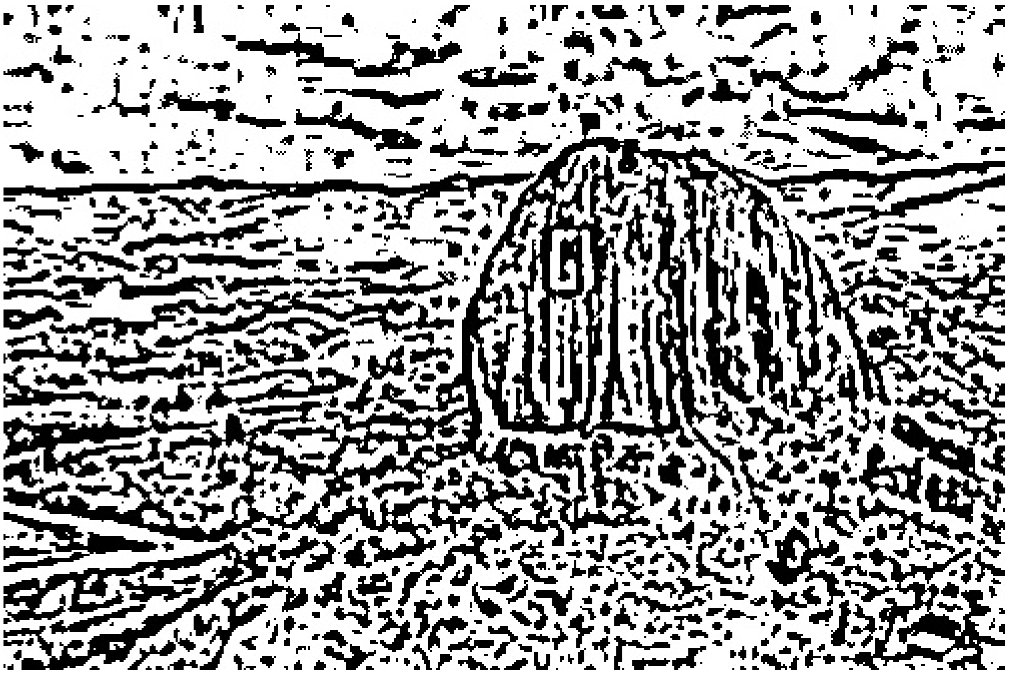


Black & White Filter

Cartoonizer Filter



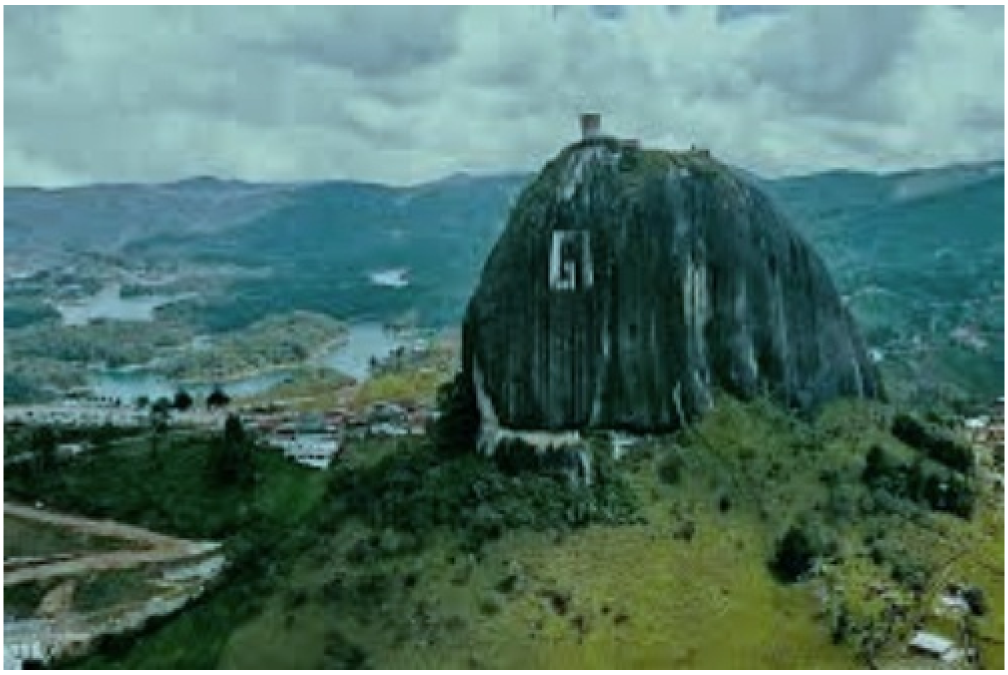
Sketcher Filter



Negative Filter

Vintage Filter

Cool Filter



Implementation in Python 3 programming language

Required libraries

```python
import sys
import cv2
import time
import numpy as np
from mpi4py import MPI
from multiprocessing import Process
from scipy.interpolate import UnivariateSpline
```

The original source code can be found here: https://github.com/ishita27/Image-Filters

Black & White Filter

```
class BlacknWhite(object):
    """BlacknWhite Filter
       A class that applies BlacknWhite filter to an image.
       The class uses downsampling, bilateral filter and upsampling to create
       a BlacknWhite filter.
    """
    def __init__(self):
        pass

    def resize(self,image,window_height = 500):
        aspect_ratio = float(image.shape[1])/float(image.shape[0])
        window_width = window_height/aspect_ratio
        image = cv2.resize(image, (int(window_height),int(window_width)))
        return image

    def render(self, img_rgb):
        img_rgb = cv2.imread(img_rgb)
        img_rgb = self.resize(img_rgb,500)
        numDownSamples = 2       # number of downscaling steps
        numBilateralFilters = 50  # number of bilateral filtering steps

        # -- STEP 1 --
        # downsample image using Gaussian pyramid
        img_color = img_rgb
        for _ in range(numDownSamples):
            img_color = cv2.pyrDown(img_color)

        # repeatedly apply small bilateral filter instead of applying
        # one large filter
        for _ in range(numBilateralFilters):
            img_color = cv2.bilateralFilter(img_color, 9, 9, 7)

        # upsample image to original size
        for _ in range(numDownSamples):
            img_color = cv2.pyrUp(img_color)

        # -- STEPS 2 and 3 --
        # convert to grayscale and apply median blur
        img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
        return img_gray

    def start(self, img_path,i):
        tmp_canvas = BlacknWhite() #make a temporary object
        file_name = img_path #File_name will come here
        res = tmp_canvas.render(file_name)
        cv2.imwrite("BlacNwhite_version/BlacNwhite_version{}.jpg".format(i),
res)
```

Cartoonizer - Filter

```
class Cartoonizer(object):
    """Cartoonizer effect
        A class that applies a cartoon effect to an image.
        The class uses a bilateral filter and adaptive thresholding to create
        a cartoon effect.
```

```python
    """
    def __init__(self):
        pass

    def resize(self,image,window_height = 500):
        aspect_ratio = float(image.shape[1])/float(image.shape[0])
        window_width = window_height/aspect_ratio
        image = cv2.resize(image, (int(window_height),int(window_width)))
        return image

    def render(self, img_rgb):
        img_rgb = cv2.imread(img_rgb)
        img_rgb = self.resize(img_rgb, 500)
        numDownSamples = 2       # number of downscaling steps
        numBilateralFilters = 50  # number of bilateral filtering steps

        # -- STEP 1 --
        # downsample image using Gaussian pyramid
        img_color = img_rgb
        for _ in range(numDownSamples):
            img_color = cv2.pyrDown(img_color)

        # repeatedly apply small bilateral filter instead of applying
        # one large filter
        for _ in range(numBilateralFilters):
            img_color = cv2.bilateralFilter(img_color, 9, 9, 7)

        # upsample image to original size
        for _ in range(numDownSamples):
            img_color = cv2.pyrUp(img_color)

        # -- STEPS 2 and 3 --
        # convert to grayscale and apply median blur
        img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
        img_blur = cv2.medianBlur(img_gray, 3)

        # -- STEP 4 --
        # detect and enhance edges
        img_edge = cv2.adaptiveThreshold(img_blur, 255,
                                         cv2.ADAPTIVE_THRESH_MEAN_C,
                                         cv2.THRESH_BINARY,9, 2)


        # -- STEP 5 --
        # convert back to color so that it can be bit-ANDed with color image
        (x,y,z) = img_color.shape
        img_edge = cv2.resize(img_edge,(y,x))
        img_edge = cv2.cvtColor(img_edge, cv2.COLOR_GRAY2RGB)

        return cv2.bitwise_and(img_color, img_edge)

    def start(self, img_path,i):
        tmp_canvas = Cartoonizer() #make a temporary object
        file_name = img_path #File_name will come here
        res = tmp_canvas.render(file_name)
        cv2.imwrite("Cartoon_version/Cartoon_version{}.jpg".format(i), res)
```

Sketcher - Filter

```python
class Sketcher(object):
    """Sketch Filter
        A class that applies Sketch filter to an image.
        The class uses a bilateral filter and adaptive thresholding to create
        a  sketch.
    """
    def __init__(self):
        pass

    def resize(self,image,window_height = 500):
        aspect_ratio = float(image.shape[1])/float(image.shape[0])
        window_width = window_height/aspect_ratio
        image = cv2.resize(image, (int(window_height),int(window_width)))
        return image

    def render(self, img_rgb):
        img_rgb = cv2.imread(img_rgb)
        img_rgb = self.resize(img_rgb, 500)
        numDownSamples = 2       # number of downscaling steps
        numBilateralFilters = 50  # number of bilateral filtering steps

        # -- STEP 1 --
        # downsample image using Gaussian pyramid
        img_color = img_rgb
        for _ in range(numDownSamples):
            img_color = cv2.pyrDown(img_color)

        # repeatedly apply small bilateral filter instead of applying
        # one large filter
        for _ in range(numBilateralFilters):
            img_color = cv2.bilateralFilter(img_color, 9, 9, 7)

        # upsample image to original size
        for _ in range(numDownSamples):
            img_color = cv2.pyrUp(img_color)

        # -- STEPS 2 and 3 --
        # convert to grayscale and apply median blur
        img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
        img_blur = cv2.medianBlur(img_gray, 3)

        # -- STEP 4 --
        # detect and enhance edges
        img_edge = cv2.adaptiveThreshold(img_blur, 255,
                                         cv2.ADAPTIVE_THRESH_MEAN_C,
                                         cv2.THRESH_BINARY,9, 2)

        # -- STEP 5 --
        # convert back to color
        (x,y,z) = img_color.shape
        img_edge = cv2.resize(img_edge,(y,x))
        img_edge = cv2.cvtColor(img_edge, cv2.COLOR_GRAY2RGB)
        #cv2.imwrite("edge.png",img_edge)
        return img_edge

    def start(self, img_path,i):
        tmp_canvas = Sketcher() #make a temporary object
        file_name = img_path #File_name will come here
        res = tmp_canvas.render(file_name)
        cv2.imwrite("Sketch_version/Sketch_version{}.jpg".format(i), res)
```

Negative - Filter

```
# find max GSV
def findMax(k):
    mx = 0
    for i in k:
        if i>mx:
            mx = i
    return mx

class Negative(object):
    def __init__(self):
        pass

    def resize(self,image,window_height = 500):
        aspect_ratio = float(image.shape[1])/float(image.shape[0])
        window_width = window_height/aspect_ratio
        image = cv2.resize(image, (int(window_height),int(window_width)))
        return image

    def render(self, img_rgb):
        img_gray = cv2.imread(img_rgb, 0)
        img_gray = self.resize(img_gray, 500)
        #get all image values
        k = []
        for i in range(img_gray.shape[0]):
            for j in range(img_gray.shape[1]):
                k.append(img_gray[i,j])

        L = findMax(k) #max GSV
        dst = img_gray[:] #copy image

        #update dst
        for i in range(img_gray.shape[0]):
            for j in range(img_gray.shape[1]):
                dst[i,j] = L - dst[i,j]
        return dst

    def start(self, img_path,i):
        tmp_canvas = Negative() #make a temporary object
        file_name = img_path #File_name will come here
        res = tmp_canvas.render(file_name)
        cv2.imwrite("Negative_version/Negative_version{}.jpg".format(i), res)
```

Vintage - Filter

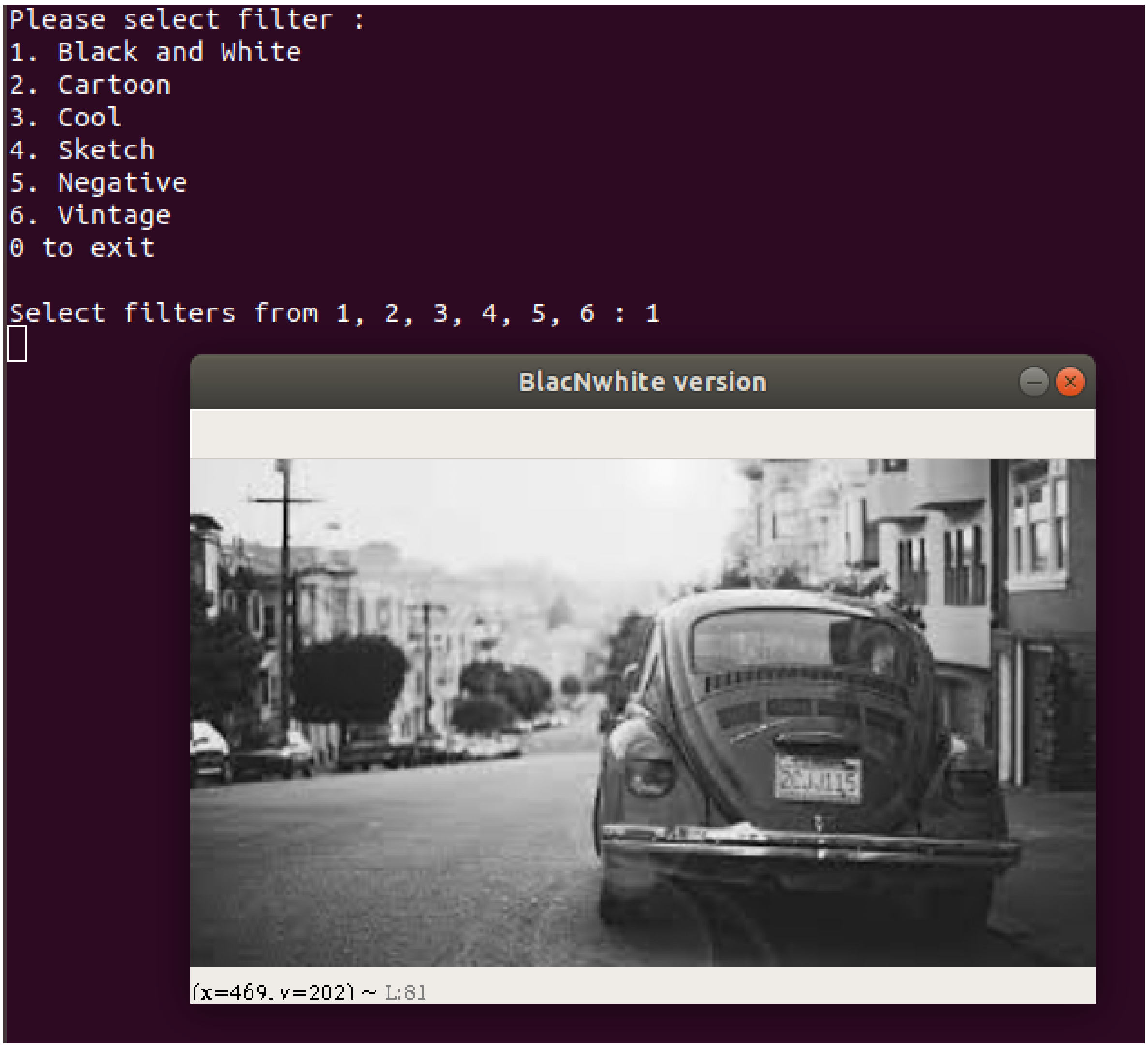
```
class Vintage(object):
    """Vintage filter ---
        This class will apply vintage filter to an image
        by applying a sepia tone to the image.
    """

    def __init__(self):
        pass

    def resize(self,image,window_height = 500):
        aspect_ratio = float(image.shape[1])/float(image.shape[0])
        window_width = window_height/aspect_ratio
        image = cv2.resize(image, (int(window_height),int(window_width)))
        return image

    def render(self, img_rgb):
        img_rgb = cv2.imread(img_rgb)
        img_rgb = self.resize(img_rgb, 500)
        img_color = img_rgb
        newImage = img_color.copy()
        i, j, k = img_color.shape
        for x in range(i):
            for y in range(j):
                R = img_color[x,y,2] * 0.393 + img_color[x,y,1] * 0.769 +
img_color[x,y,0] * 0.189
                G = img_color[x,y,2] * 0.349 + img_color[x,y,1] * 0.686 +
img_color[x,y,0] * 0.168
                B = img_color[x,y,2] * 0.272 + img_color[x,y,1] * 0.534 +
img_color[x,y,0] * 0.131
                if R > 255:
                    newImage[x,y,2] = 255
                else:
                    newImage[x,y,2] = R
                if G > 255:
                    newImage[x,y,1] = 255
                else:
                    newImage[x,y,1] = G
                if B > 255:
                    newImage[x,y,0] = 255
                else:
                    newImage[x,y,0] = B

        return newImage

    def start(self, img_path,i):
        tmp_canvas = Vintage()
        file_name = img_path #File_name will come here
        res = tmp_canvas.render(file_name)
        cv2.imwrite("Vintage_version/Vintage_version{}.jpg".format(i), res)
```

Cool - Filter

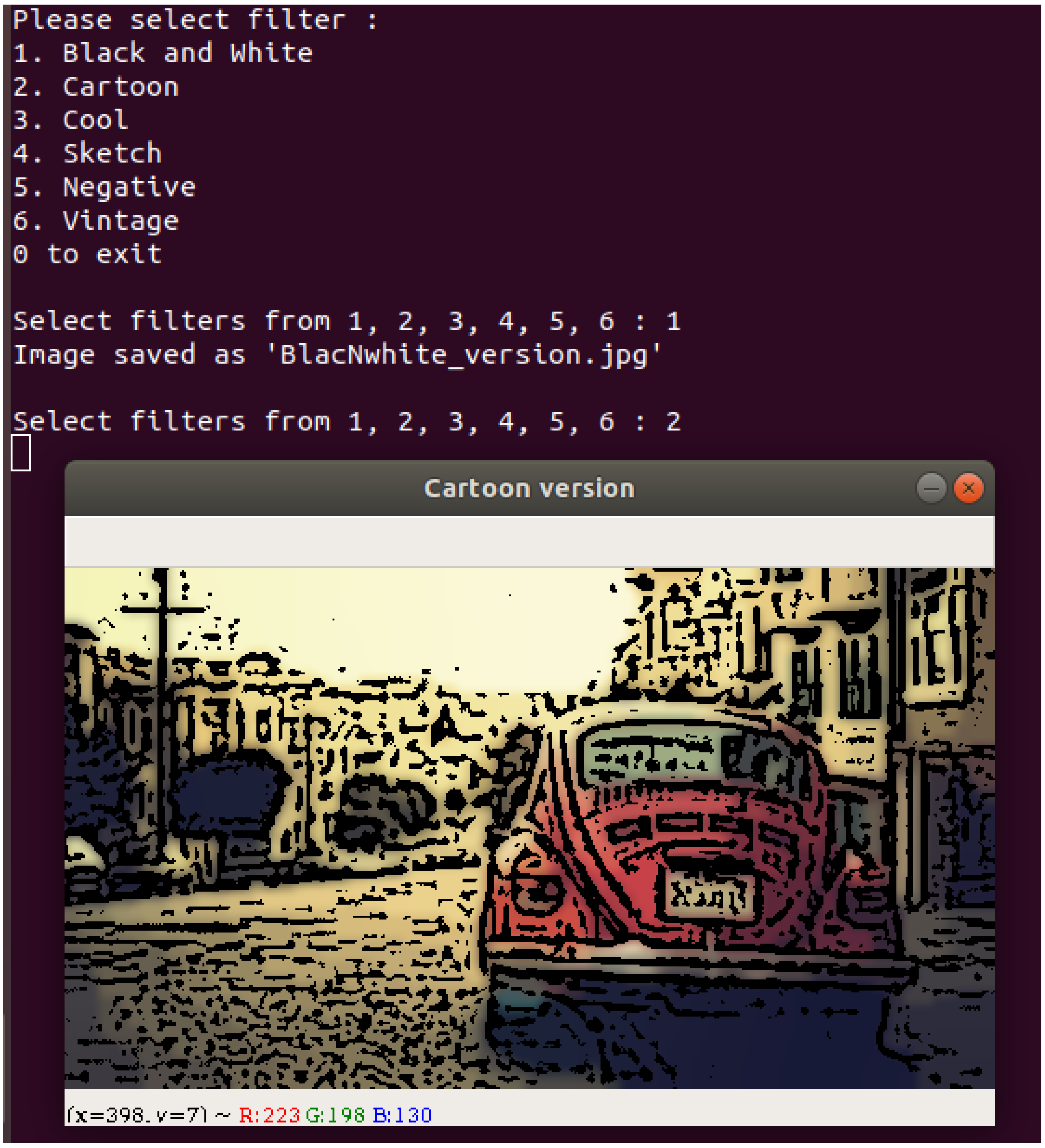
```
class Cool(object):
    """cool_filter ---
        This class will apply cool filter to an image
        by giving a sky blue effect to the input image.
    """
    def __init__(self):
        # create look-up tables for increasing and decreasing red and blue
resp.
        self.increaseChannel = self.LUT_8UC1([0, 64, 128, 192, 256],
                                              [0, 70, 140, 210, 256])
        self.decreaseChannel = self.LUT_8UC1([0, 64, 128, 192, 256],
                                              [0, 30,  80, 120, 192])

    def resize(self,image,window_height = 500):
        aspect_ratio = float(image.shape[1])/float(image.shape[0])
        window_width = window_height/aspect_ratio
        image = cv2.resize(image, (int(window_height),int(window_width)))
        return image

    def render(self, img_rgb):
        img_rgb = cv2.imread(img_rgb)
        img_rgb = self.resize(img_rgb, 500)
        #cv2.imshow("Original", img_rgb)
        r,g,b = cv2.split(img_rgb)
        r = cv2.LUT(r, self.increaseChannel).astype(np.uint8)
        b = cv2.LUT(b, self.decreaseChannel).astype(np.uint8)
        img_rgb = cv2.merge((r,g,b))
        # saturation decreased
        h,s,v = cv2.split(cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV))
        s = cv2.LUT(s, self.decreaseChannel).astype(np.uint8)


        return cv2.cvtColor(cv2.merge((h,s,v)), cv2.COLOR_HSV2RGB)

    def LUT_8UC1(self, x, y):
        #Create look-up table using scipy spline interpolation function
        spl = UnivariateSpline(x, y)
        return spl(range(256))

    def start(self, img_path,i):
        tmp_canvas = Cool() #make a temporary object
        file_name = img_path #File_name will come here
        res = tmp_canvas.render(file_name)
        cv2.imwrite("Cool_version/Cool_version{}.jpg".format(i), res)
```

¿What was the original implementation?

Initially, it could be used through the following menu that allowed access to a single filter with a single image, like this:



```
Please select filter :
1. Black and White
2. Cartoon
3. Cool
4. Sketch
5. Negative
6. Vintage
0 to exit

Select filters from 1, 2, 3, 4, 5, 6 : 1
```



```
Please select filter :
1. Black and White
2. Cartoon
3. Cool
4. Sketch
5. Negative
6. Vintage
0 to exit

Select filters from 1, 2, 3, 4, 5, 6 : 1
Image saved as 'BlacNwhite_version.jpg'

Select filters from 1, 2, 3, 4, 5, 6 : 2
```

However, our purpose is to be able to use this library to apply filters on videos

So we opted to process videos using frames, these frames were generated externally as follows

```
ffmpeg -i peces.mp4 -r 1 -f image2 image-%d.jpeg
```

Finally we run a program that applies all the filters to each frame of the video:

Sequential Algorithm Adapted for filtering a Video

```
bnw = blacknwhite_filter.BlacknWhite()
crt = cartoon_filter.Cartoonizer()
skch = sketch_filter.Sketcher()
ngtv = negative_filter.Negative()
vntg = vintage_filter.Vintage()
cool = cool_filter.Cool()

def main():
    for i in range(1,1380):
        path = "filtros/image-{}.jpeg".format(i)
        bnw.start(path,i)
        crt.start(path,i)
        skch.start(path,i)
        ngtv.start(path,i)
        vntg.start(path,i)
        cool.start(path,i)
t1 = time.time()
main()
t2 = time.time()
print("It takes {} sec".format(t2-t1))
```

So having saved the images that will be used, first we proceed to create an instance of each class, then what is done in the main is to load each image, and then each filter is called, it is given a distinction number so that it can save the image as a single image. Within each class there is an address that will be the folder in which the image will be saved, so that at the end of the cycle there will be 6 folders, each with 1380 images, each folder will have a different filter. The time taken by this algorithm was **84.39 minutes**, which is equivalent to **1 hour and 24.39 minutes**

First try of paralellizing the algorithm

Then in order to improve performance, we tried to parallelize using threads as follows:

```
bnw = blacknwhite_filter.BlacknWhite()
crt = cartoon_filter.Cartoonizer()
skch = sketch_filter.Sketcher()
ngtv = negative_filter.Negative()
vntg = vintage_filter.Vintage()
cool = cool_filter.Cool()


def f(path,i):
    bnw.start(path,i)
    crt.start(path,i)
    skch.start(path,i)
    ngtv.start(path,i)
    vntg.start(path,i)
    cool.start(path,i)


def main():
    for i in range(1,1380):
        path = "filtros/image-{}.jpeg".format(i)
        p = Process(target=f, args=(path,i))
        p.start()
        p.join()
t1 = time.time()
main()
t2 = time.time()
print("It takes {} sec".format(t2-t1))
```

In this algorithm, an instance of each class is created first, then inside a for loop what is done is load each frame of the video, and then a thread is created to call the function that contains the call to each filter function. The idea is to create a different thread for each iteration to speed up processing, so as many threads as possible are created.

The time taken by this algorithm was **68.49** minutes, that is, **1 hour and 8.49 minutes**

It's **speed up was 1.23**.

Paralellization using MPI

Finally, another parallel algorithm was implemented that gives us a greater improvement using MPI, as follows:

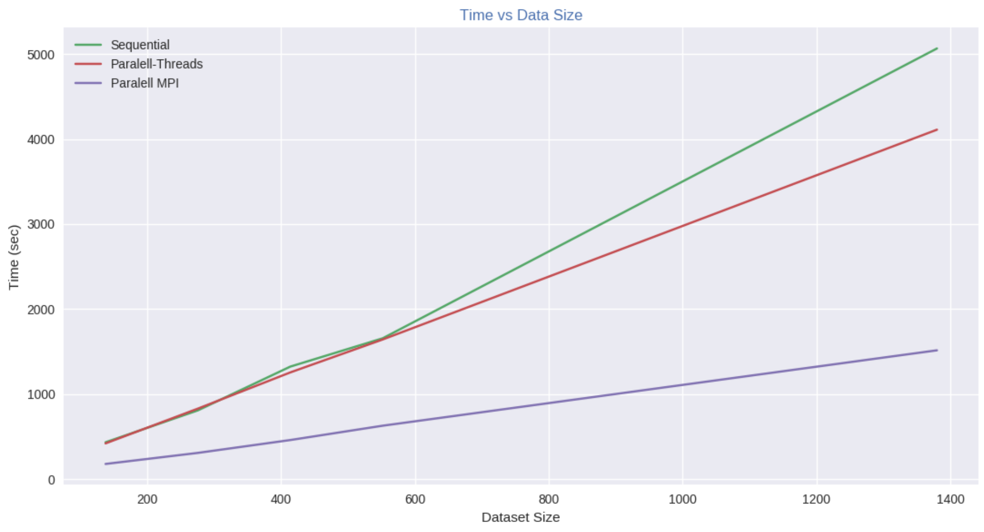
```
bnw = blacknwhite_filter.BlacknWhite()
crt = cartoon_filter.Cartoonizer()
skch = sketch_filter.Sketcher()
ngtv = negative_filter.Negative()
vntg = vintage_filter.Vintage()
cool = cool_filter.Cool()

t1 = MPI.Wtime()
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()


for i in range(rank+1,1380,size):
    path = "filtros/image-{}.jpeg".format(i)
    bnw.start(path,i)
    crt.start(path,i)
    skch.start(path,i)
    ngtv.start(path,i)
    vntg.start(path,i)
    cool.start(path,i)

t2 = MPI.Wtime()

if rank==0:
    print("It takes {} sec".format(t2-t1))
```

In this algorithm, one instance of each class is created first, then the communicator in which it is located is obtained, then the number of cores inside the communicator is obtained, and finally the rank of each one of the cores is obtained. Within the for cycle, what is done is that for each of the processors a frame of the video is loaded and all the filters are applied until all the frames of the video are processed. In our case it was tested with a **4 processors machine with 12GB of RAM**

This algorithm took **25.27 minutes** and the **speed up was 3.34**

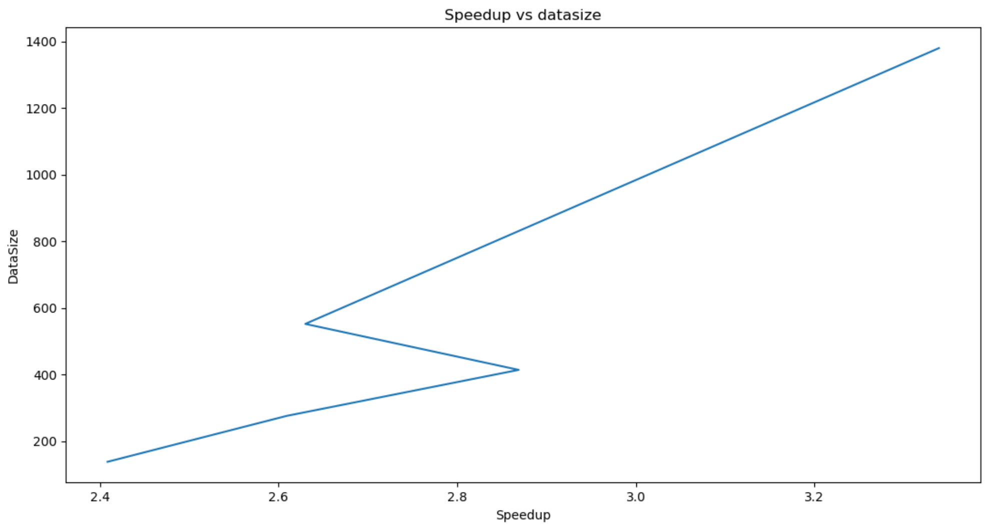
Plots

Time vs Data Size

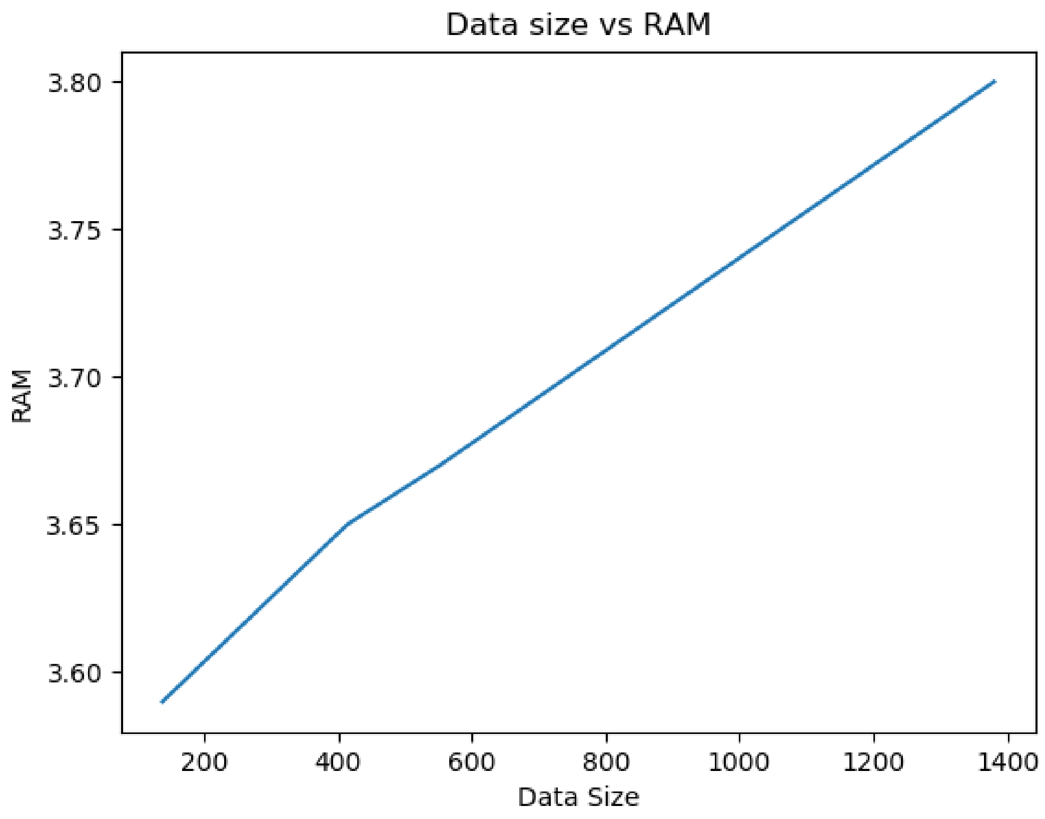


The previous plot allows us to observe the behavior of the three implemented algorithms, thus visualizing the time it takes for each one of them to process a certain number of frames of the video that was selected (represented as the dataset size).The parallel algorithm that uses threads show a similar behavior to the sequential one until the size of the data set is approximately 600 frames,but after this point there is a small difference between them, the parallel being better, but this difference isnt significant.On the other hand, the parallel algorithm that use MPI allow a better performance for any dataset of frames, and compared to the other two algorithms, the difference is really significant.

Speed Up vs Data Size (MPI - ALGORITHM)



Data Size vs RAM



This graph of RAM memory consumption allows us to identify that the behavior of the Speed Up shown in the previous graph is not limited by the capacity of RAM, since the maximum consumption is 3.6GB from 12GB

Results

At the end of this project, we conclude that within parallel programming it is important to think of a good design to solve a problem, it is not enough to find a way that allows solving the problem, it is necessary to find the best way since this way you can find the Desired solution to the problem that is established, for this reason during the development of the project two algorithms were performed in order to find the one with the best performance, in order to have a solution when using parameters that have very large values.

The use and function of filters in photography is more important for most photographers, since their use can represent the difference between a faded photograph and one with variations in tone and contrasts of brilliance.

The function of the filters is to block or absorb some of the light that would otherwise reach the lens to impress the film.
One of the most widespread uses of contrast filters is to highlight clouds in the sky, to give dramatic effects, and to penetrate fog.

Thanks to the parallelization of these algorithms, many works carried out by professionals dedicated to the field of design, photography, video and media in general can be carried out in much less time than those that would be used if software with sequential algorithms were used.