

# Interesting Special Effects in Color Images

---

*Image Processing Project Documentation*

---



**TECHNICAL UNIVERSITY**  
**of Cluj Napoca**

*Student : Deac Dan Cristian*

## *Content*

### **1. Introduction**

- context
- issues that should be solved
- proposed objectives

### **2. Theoretical background**

- bibliographical study from the literature (cite every source)
- description of the methods that can be applied
- description of the possible solutions

### **3. Design and implementation**

- description of the chosen solution which accomplishes the proposed objectives
- block diagram
- description of the implementation flow
- description of the functionalities from all implemented modules
- description of the implemented algorithms
- description of the application usage

### **4. Experimental results**

- practical results obtained in different test cases (scenarios)
- remarks on each obtained result
- proof that the obtained results are valid

### **5. Conclusions**

- achievements and the degree of accomplishing the objectives
- personal contributions
- observations about the obtained results
- future development directions

### **Bibliography**

- list of the references that were used in the documentation

## Introduction

Context:

“ **Interesting special effects on color images.**

- you'll study some interesting special effects that can be applied on color images.
- you'll implement a set of them (the number depends on the difficulty).

“

This project implements different filters and “effects” to an image making use of the OpenCV computer vision library in C++. This is the documentation of the semester project of the Image processing Course. Alongside the laboratory work during the 2<sup>nd</sup> semester of 3<sup>rd</sup> year, I developed the ability to solve such problems and make use of this computer vision library.

Issues:

- Select an image (color image).
- Make sure the image was correctly transformed to a `cv::Matrix`
- Find filters and apply them to an image.
- Iterate through every pixel if it is the case and apply the required filters
- Create a destination matrix if needed and use `cv::imshow` to print it alongside the source image
- In case of filters such as Vignette or others sliders are required to play with the intensity or other values

Proposed Objectives:

These objectives represent the filters that I am going to implement.

- For the beginning I implemented the simple color image to grayscale function from the laboratory 2 work.
- Black and White : Transforms the color image to a grayscale first, then to a black and white image based on a threshold selected by the user.
- Sepia: This applies a standardized matrix to each pixel based on its color channels and adds a vintage look to the image
- Vignette: I personally use this one while designing my wallpapers. This adds dark faded corners to the image. Also, I implemented a slider to control how dark everything gets.
- Cat ears: Yes, cat ears. This is an idea of a filter proposed to me by the lab instructor :) This does not use machine learning to detect the face, but instead I implemented some sliders that apply matrix transformations on the cat ears matrix to adjust it manually.

## 2. Theoretical background

As mentioned in the introduction, The base of my knowledge represents the Image processing lectures and the laboratory work, especially the 2<sup>nd</sup> laboratory which introduces the basics of image processing and laboratory work 9, where we work with filters and kernels to make changes to the source image. Additionally, for the implementation of the project and some filters like Sepia, I found a huge help from the LearnOpenCV website. It represents one of the best sources for this computer vision library.

Because I have several filters, I will talk about only the Sepia filter in this section, but everything can be translated to the other filters as well.

Here is the code from learnopencv.com

```
1 void sepia(Mat img){
2   Mat res = img.clone();
3   cvtColor(res,res,COLOR_BGR2RGB);
4   transform(res,res,Matx33f(0.393,0.769,0.189,
5     0.349,0.686,0.168,
6     0.272,0.534,0.131));
7   cvtColor(res,res,COLOR_RGB2BGR);
8   imshow("original",img);
9   imshow("Sepia",res);
10  waitKey(0);
11  destroyAllWindows();
12}
```

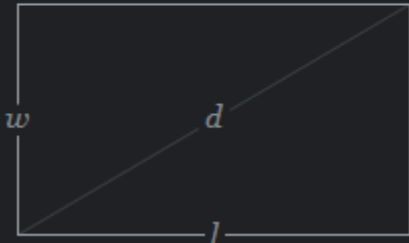
Here it is explained that a standard matrix[3][3] with values seen above is always applied to the pixel of the image. This I implemented without using the transform function already implemented in OpenCV, by taking the channels of each pixel and performing multiplication and addition as required. More details in the Implementation. Alongside these sources, I used the OpenCV documentation (docs.opencv.org) to learn different features like the slider implementation, which uses a trackbar.

Furthermore I used some mathematics when compute the distance from each pixel to either corners of the image when doing the Vignette filter. There are some equations like the diagonal of the image :

$$d = \sqrt{w^2 + l^2}$$

$l$  Length

$w$  Width



### 3. Design and Implementation

- Select Image

```
void utcn::ip::Project::selectImageFile() {  
    currentImagePath = fileutil::getSingleFileAbsPath();  
}
```

This function is called at the beginning of the execution, and the user can select an image to apply a filter to. This function can also be called during the execution from user input. It is not really recommended to use global variables, but for the purpose of this project I have `currentImagePath` as a global variable.

- Grayscale

I will not get into much detail here, as it is part of the laboratory work. This function takes the sum of each color channel of a pixel in the image, and divides it by 3. The output is saved in the destination image and this is done for each pixel.

```
for (int i = 0; i < height; i++) {  
    for (int j = 0; j < width; j++) {  
        cv::Vec3b val = src.at<cv::Vec3b>(i, j);  
        dst.at<uchar>(i, j) = (val[0] + val[1] + val[2]) / 3;  
    }  
}
```

- Black and White

This applies the grayscale to the image, and then based on the threshold entered by the user transforms the image to a black and white image. It is checked if the

```
for (int i = 0; i < height; i++) {  
    for (int j = 0; j < width; j++) {  
        uchar val = src.at<uchar>(i, j);  
        if (val < thresh)  
            dst.at<uchar>(i, j) = 0;  
        else  
            dst.at<uchar>(i, j) = 255;  
    }  
}
```

- Vignette

Here comes the more interesting part. This filter uses some mathematical equations to compute the destination image. This filter needs to first find the center of the image and the diagonal:

```
// Calculate center of the image

cv::Point center = cv::Point(cols / 2, rows / 2);
// The diagonal - shown at theoretical considerations
double maxDist = std::sqrt((cols / 2) * (cols / 2) + (rows / 2) * (rows / 2));
```

The center and the diagonal are used to compute the degree of darkening for each pixel.

This is done by iterating through each pixel, and we calculate a distance and a vignetteStrength which is distance / maxDist. This calculates the degree of darkening based on how far the pixel is from the center. ( Euclidian distance which uses the Pythagorean theorem  $\text{distance} = \sqrt{(i - \text{rows}/2)^2 + (j - \text{cols}/2)^2}$ ). Instead of rows and cols/ 2 I use the center point ( center.x and center.y)

```
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        double distance = std::sqrt((center.y - i) * (center.y - i) +
                                     (center.x - j) * (center.x - j));
        double vignetteStrength = distance / maxDist;

        for (int ch = 0; ch < 3; ++ch) {
            mask.at<cv::Vec3b>(i, j)[ch] = cv::saturate_cast<uchar>(
                (1 - vignetteStrength) * mask.at<cv::Vec3b>(i, j)[ch]);
        }
    }
}
```

In the last part I iterate through each pixel's color channels and use saturate\_cast to make sure there is no overflow when assigning the new channel values.

$(1 - \text{vignetteStrength}) * \text{mask.at<cv::Vec3b>(i, j)[ch]}$  computes the new value of the channel by reducing its intensity according to the vignette strength. The  $(1 - \text{vignetteStrength})$  factor is close to 1 for pixels near the center of the image (which should be less darkened), and close to 0 for pixels near the corners (which should be darkened more).

To implement the slider, I had to add a simple modification to the vignetteStrength, a multiplication factor.

```
double vignetteStrength = distance / maxDist * g_vignetteStrength;
```

This is implemented in the `on_trackbar` function which is called and generated in the main test Vignette function using this feature of the OpenCV library:

```
cv::createTrackbar("Strength", "Vignette Filter", &value, 100, on_trackbar);
```

- Sepia

This effect uses a standard transformation matrix that is applied to each pixel :

```
cv::Mat transformationMatrix = (cv::Mat_<float>(3,3) <<
                                .393, .769, .189,
                                .349, .686, .168,
                                .272, .534, .131);
```

Starting from the pseudocode provided in the [learnopencv.com](http://learnopencv.com) site, I implemented the matrix transform in the following way:

```
// get each pixel from src
```

```
cv::Vec3b val = src.at<cv::Vec3b>(i, j);
```

```
uchar inputBlue = val[0];
uchar inputGreen = val[1];
uchar inputRed = val[2];
```

```
// Multiply each row to the RGB values and add the result to the corresponding color channel.
```

```
// min() is used to make sure that the value is never over 255 ( avoid overflow )
```

```
uchar outputRed = std::min(255., (inputRed * .393) + (inputGreen * .769) +
                             (inputBlue * .189));
uchar outputGreen = std::min(255., (inputRed * .349) + (inputGreen * .686) +
                               (inputBlue * .168));
uchar outputBlue = std::min(255., (inputRed * .272) + (inputGreen * .534) +
                              (inputBlue * .131));
dst.at<cv::Vec3b>(i, j) = cv::Vec3b(outputBlue, outputGreen, outputRed);
```

- Cat Ears

The way I implemented this is by using a cat\_ears.png image which has a transparent background.

I iterate through the pixels of the cat, and when the pixel is not transparent, I add it to the source by replacing the src pixel with the cat\_ears.png pixel, and adding an offset(start\_row/start\_col) to row and col.

```
for (int row = 0; row < cat_ears.rows; row++) {
    for (int col = 0; col < cat_ears.cols; col++) {
        cv::Vec4b& pixel = cat_ears.at<cv::Vec4b>(row, col);
        // If the cat ears image pixel is not fully transparent
        if (pixel[3] > 0) {
            // Copy pixel from cat ears image to the main image
            src.at<cv::Vec3b>(start_row + row, start_col + col) =
                cv::Vec3b(pixel[0], pixel[1], pixel[2]);
        }
    }
}
```

The offset is controlled by the sliders. Each slider has its own on\_trackbar function which modifies the image in real time.

The value of ears\_height\_pos and ears\_width\_pos is modified by the slider.

```
int start_row = src.rows / 4 - ears_height_pos;
int start_col = src.cols / 4 + ears_width_pos;
```

The other sliders implement rotation matrix and the resize of the ears image.



## Application Usage

This is best described by this code :

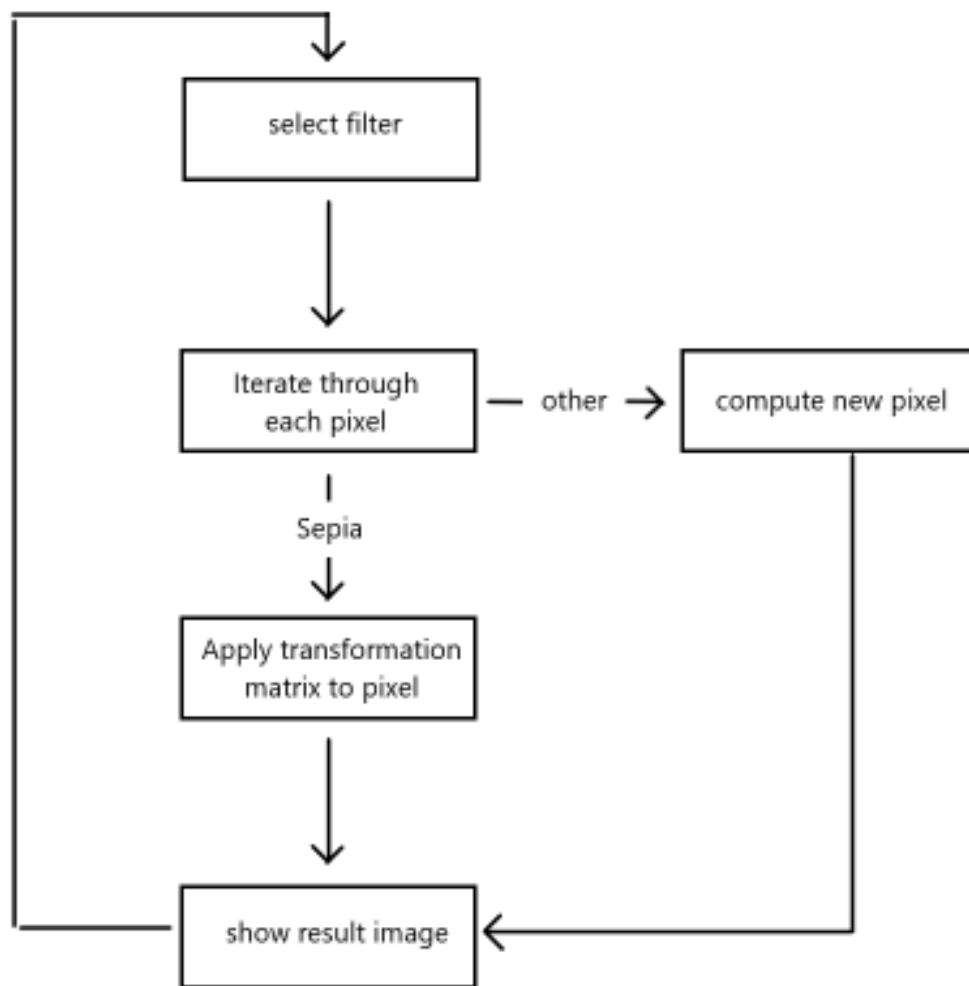
```
selectImageFile();
do {
    utcn::ip::Project::printMenu(LAB_MENU);
    std::cin >> op;
    switch (op) {
        case 0:
            break;
        case 11:
            selectImageFile();
            break;
        case 1:
            testColorToGrayscale();
            break;
        case 2:
            testColorToBW();
            break;
        case 3:
            testSepiaFilter();
            break;
        case 4:
            testVignette();
            break;
        case 5:
            addCatEars();
            break;
        default:
            std::cout << "Invalid selection" << std::endl;
    }
} while (op != 0);
```

The user first is prompted to select an image then there is a CLI interface where each filter is described, and the user can select a filter based on the number he enters.

```
Menu:
1 - Convert Color to grayscale
2 - BLACK AND WHITE: converts an image to black and white
3 - SEPIA filter: adds a warm brown color to the image
4 - VIGNETTE filter: adds dark faded edges to an image
5 - Cat ears.. yes cat years
11 - Select image file
0 - Exit

Option:
█
```

Block diagram



## 4. Experimental Results

Showcase video :

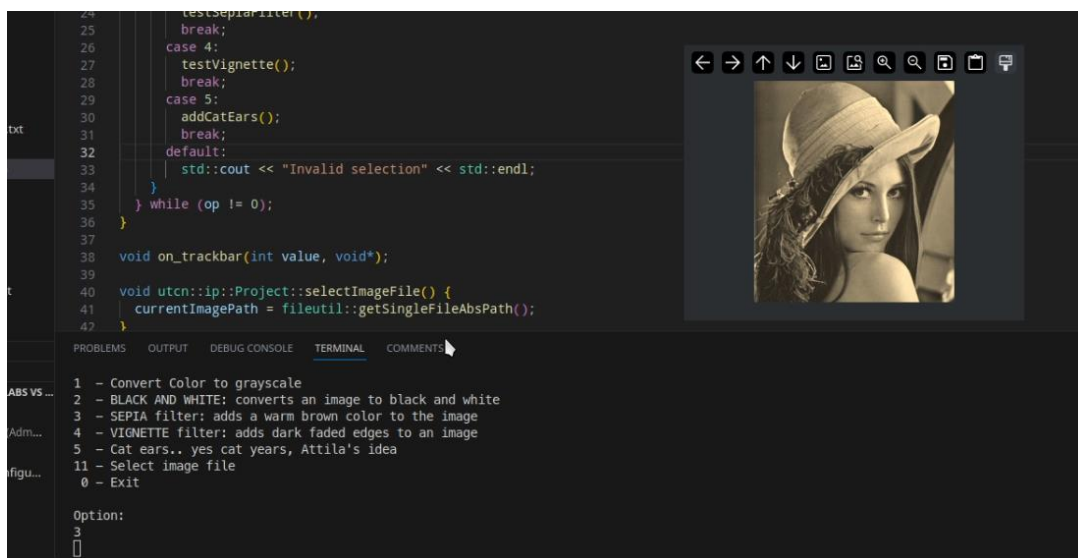
[https://drive.google.com/file/d/1z5cjbB3jf95punpS4tblyXZq\\_PrcaVbd/view?usp=sharing](https://drive.google.com/file/d/1z5cjbB3jf95punpS4tblyXZq_PrcaVbd/view?usp=sharing)

Base image :



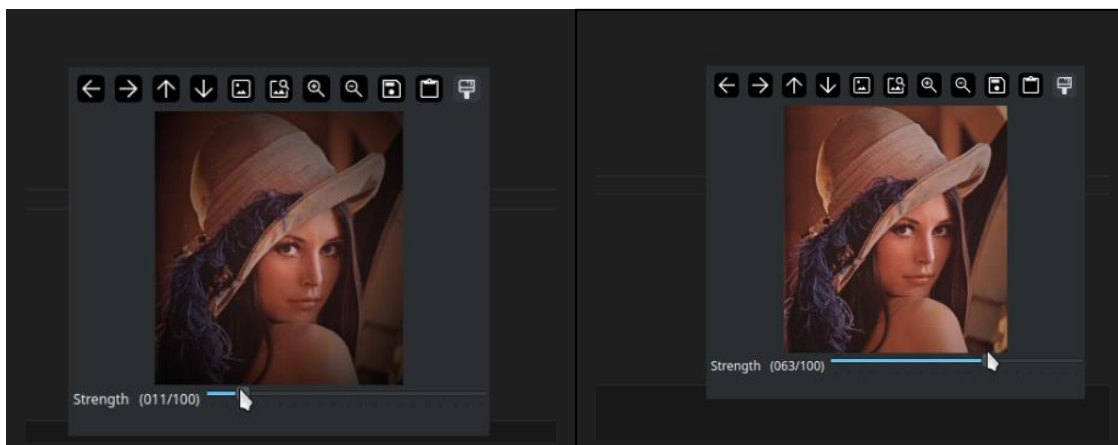
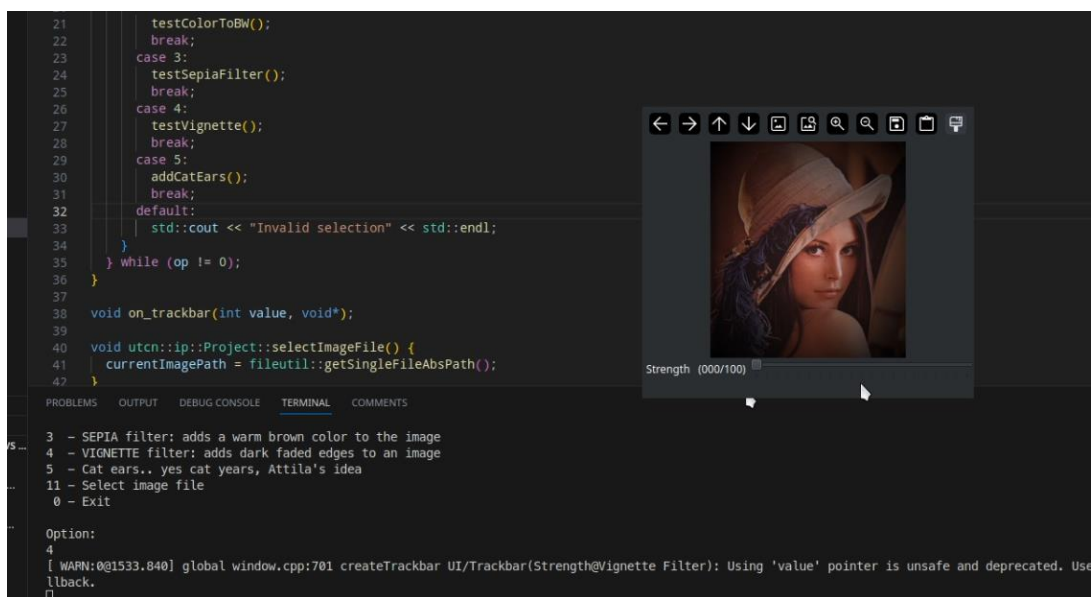
Sepia

There is nothing special to add, it is clearly seen that the picture gets a vintage tint to it which is a visible fact.



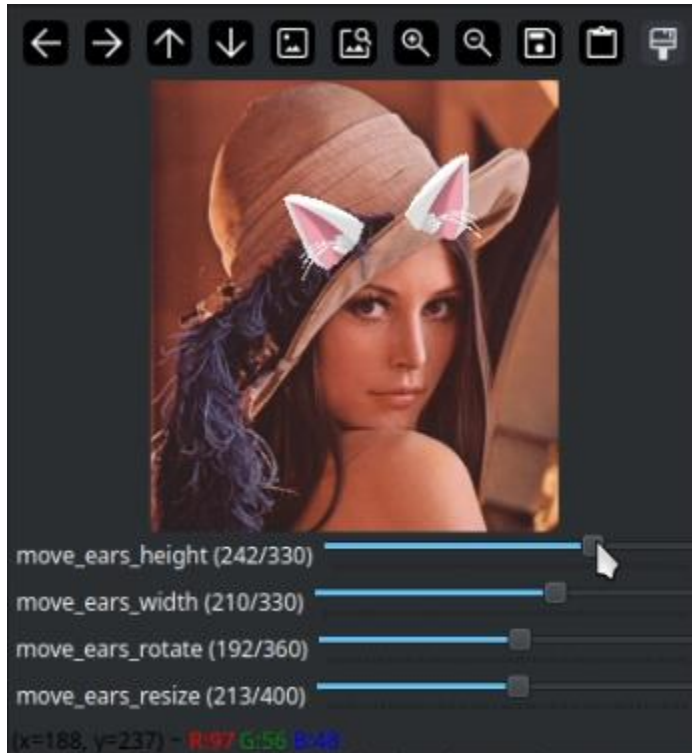
## Vignette

As seen in the pictures below, the intensity of the vignette is reduced by the slider. It is clearly seen that the pixels further from the center get darker and darker – corners are the furthest and they have a balanced darkening factor due to the fact that it's the same distance from each corner to the center.

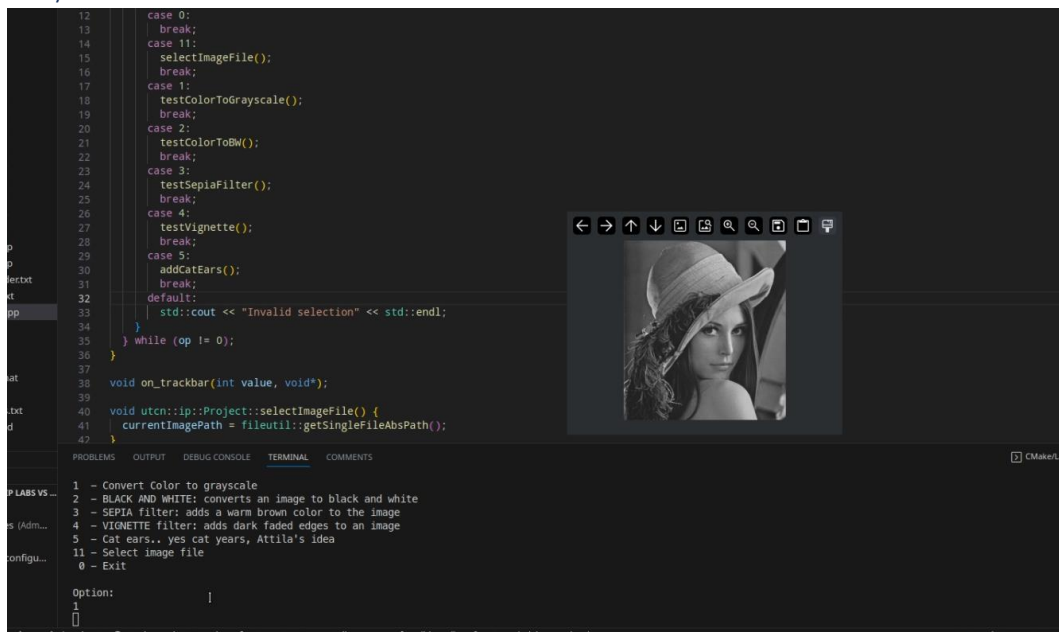


## Cat Ears

This can be used on any image and no matter the size of the image the cat ears can be resized and placed in the desired spot. Automating this means implementing an ML algorithm which is not in the scope of this lab

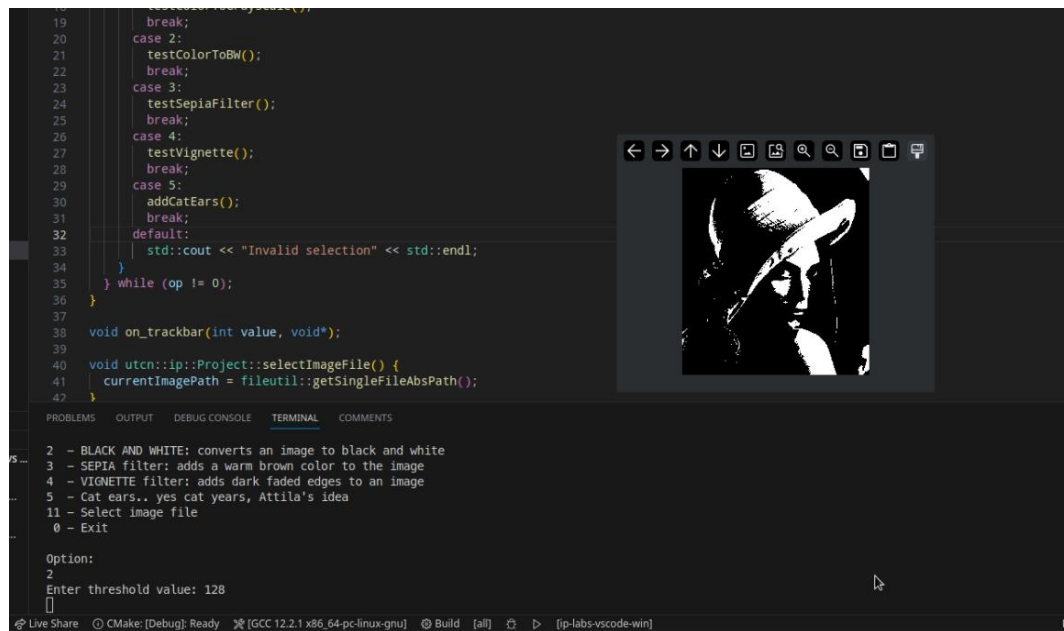


## Grayscale



## BW

User enters a threshold and based on it the BW image is generated.



## 5. Conclusions

I believe that all the filters have been successfully performed. Maybe the black and white image could have been refined a bit more. The achievements are represented by the successful implementation of the Sepia, Vignette and the Cat Ears filter.

I worked a lot on calibrating the sliders for the two filters and had to do a little bit of math to determine the best way for computing the distance from the center in the vignette filter. The Sepia filter was pretty much straight forward.

For future development, I believe there can be added a neat little graphical user interface using PyQt5 and have the filter change in real time. There can also be a save option and then add a stacking feature where you can stack filters on top of each other.

There can be sliders for the grayscale and black and white images as well and add a certain tint to the image.

Overall this project helped me develop skills with working in the OpenCV computer vision library and learned more about C++ classes during the implementation as well as lab work.

## 6. Bibliography

Lectures/labs from [http://users.utcluj.ro/~igiosan/teaching\\_ip.html](http://users.utcluj.ro/~igiosan/teaching_ip.html)

[https://docs.opencv.org/3.4/da/d6a/tutorial\\_trackbar.html](https://docs.opencv.org/3.4/da/d6a/tutorial_trackbar.html)

<https://learnopencv.com/photoshop-filters-in-opencv/#Sepia>

<https://www.haroldserrano.com/blog/matrices-in-computer-graphics>

Image attribution: Horns Png vectors by Lovepik.com - <https://lovepik.com/images/png-horns.html>