|  |  |  |
| --- | --- | --- |
|  | **T.C.**  **MANİSA CELAL BAYAR ÜNİVERSİTESİ**  **MÜHENDİSLİK FAKÜLTESİ**  **BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ** |  |

**REAL-TIME IMAGE FILTERING AND FACE DETECTION**

**Tasarım Projesi / Lisans Bitirme Tezi**

**HAZIRLAYANLAR**

**ARDA DUMANOĞLU 190315072**

**TOLGA TUGAN 170315050**

**DANIŞMAN**

**PROF. DR. BORA CANBULA**

**MANİSA 2021**

|  |  |  |
| --- | --- | --- |
|  | **T.C.**  **MANİSA CELAL BAYAR ÜNİVERSİTESİ**  **MÜHENDİSLİK FAKÜLTESİ**  **BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ** |  |

**REAL-TIME IMAGE FILTERING AND FACE DETECTION**

**Tasarım Projesi / Lisans Bitirme Tezi**

**HAZIRLAYANLAR**

**ARDA DUMANOĞLU 190315072**

**TOLGA TUGAN 170315050**

**DANIŞMAN**

**PROF. DR. BORA CANBULA**

**MANİSA 2021**

**TABLE OF CONTENT**

Page

FIGURE LIST 4

ABSTRACT 7

INTRODUCTION 8

1. FILTERS 9
   1. Apply Invert Function 9

1.2 Apply Sepia Filter Function 9

1.3 Apply Color Overlay Function 11

1.4 Apply Hue Saturation Function 14

1.5 Apply Circle Focus Filter Blur Function 14

1.6 Apply Threshold Mode Function 15

1.7 Apply Portrait Mode Function 16

2. REAL-TIME FACE DETECTION 17

2.1 Real-Time Face Detection Function 17

3. IMAGE FACE DETECTION 18

3.1 Image Face Detection Function 18

4. RECORDING VIDEOS 19

4.1 Record Video Function 19

5. INTERFACE 21

5.1 Interface File 22

5.2 Modifications of the Interface 23

6. AUTO BRIGHTNESS AND CONTRAST 23

7. IMAGE ENHANCEMENT 25

8. REAL-TIME VIDEO ENHANCER 29

9. CONCLUSION 31

REFERENCES 32

**FIGURE LIST**

**FILTERS**

Figure 1.1 Apply invert function

Figure 2.1 Apply sepia filter function

Figure 2.2 Verify alpha channel function

Figure 3.1 Operations in color overlay adjustments

Figure 3.2 Output of the interface.py

Figure 3.3 Output of the color overlay filter

Figure 3.4 Second output of the color overlay filter

Figure 3.5 Third output of the color overlay filter

Figure 4.1 Apply hue saturation function

Figure 5.1 Apply circle focus filter blur function

Figure 5.2 Alpha blend function

Figure 6.1 Apply threshold mode function

Figure 7.1 Apply portrait mode function

**REAL-TIME FACE DETECTION**

Figure 1.1 Real-time face detection function

**IMAGE FACE DETECTION**

Figure 1.1 Image face detection function

**RECORDING VIDEOS**

Figure 1.1 Record video function

Figure 1.2 Output 1

Figure 1.3 Output 2

**INTERFACE**

Figure 1.1 The interface file

Figure 2.1 Modifications of the interface file

**AUTO BRIGHTNESS AND CONTRAST**

Figure 1.1 Function to get an adjusted image

Figure 1.2 Functions to get new values and create a new image

Figure 1.3 Output

**IMAGE ENHANCEMENT**

Figure 1.1 Manual image enhancement

Figure 1.2 Manual image editor

Figure 1.3 Importing the file

Figure 1.4 Output

Figure 1.5 Second output

Figure 1.6 Original sample

Figure 1.7 Edited version

Figure 1.8 Output stored as a temp file

**REAL-TIME VIDEO ENHANCER**

Figure 1.1 Manual real-time video enhancement

Figure 1.2 Implementation of the function in the interface

Figure 1.3 Output

**ABSTRACT**

The main purpose of this project was to make a real-time face detection and facial recognition system and also apply filters to them. And by putting all of these options on our interface, it was to allow the user to choose. We tried to create a comprehensive and multifunctional user interface by presenting it in options such as video recording – resizing, etc. in our interface. In addition, we plan to make an image-face detection option, where the user chooses the image, according to the user's request. Then we have the plan to create a Machine Learning algorithm for training that image. And the algorithm decides whether the face in the image is the same as the face on the user's image. Then, on the advice of our advisor lecturer, he wants us to make a virtual camera and use our program’s output as an output in another program. We were planning to continue with the machine learning part because of our internship project's subject.

We have used several technologies involving around Opencv3. Taking advantage of the knowledge we have gathered during the Image Processing classes, we were able to apply real-time image filters.

By using OpenCV libraries, we managed to include a real-time face detection feature, with the help of cascades.

Including several more additions to our project; we are planning to implement a wizard tool-like feature that can detect a human face in real-time by perceiving the characteristics of every frame including brightness, contrast, saturation, and then automatically revise, improve every frame to get a better look.

Aside from our mainly focused function, we were able to add some small features to the project like video recording and image face detection.

And finally, to make the product show its specialties, we’ve gathered all our functions into an interface. From the interface, a user can reach every feature using buttons and exit from them individually whenever they like by hitting the designated button.

We expect this project to include various functions which combine image processing and face detection with the help of Opencv3 in a well-coordinated manner.

*Keywords: Real-Time Image Filtering, Real-Time Face Detection, Visual Improvement*

**INTRODUCTION**

The final aim of this project is to find an automated way to enhance real-time view which contains a human face. The project is written in Python Language using Open Source Computer Vision Library, also known as Opencv[1]. Along with that, we’ve used the numpy[2] library for holding the frame values of masks and overlays in image filtering, Tkinter library for creating our graphical user interface, and os library for interacting with the operating system to change the contents of a directory in video recording.

The project consists of five different python files, a default OpenCV library that comes with a cascade we specifically use, and a photo of a human face for testing the image processing function. Python files named filters, Interface, Real\_Time\_Face\_Detection, image\_face\_detection, and record\_video form all the necessary source codes.

All the functions are reached from the Interface, meaning that the other source files support this only executable file by getting imported and referred for calling their functions and assigning buttons to those functions using the Tkinter library[3].

1. **FILTERS**
   1. **Apply Invert Function**

This function takes a frame from the capture object which has been created by an OpenCV function called VideoCapture to get real-time visuals and inverts every bit in the input frame in a while loop to get a constant real-time negative filter effect. Like every filter we’ve used in this file, we convert the frame into grayscale to get a better performance before we apply the filters.

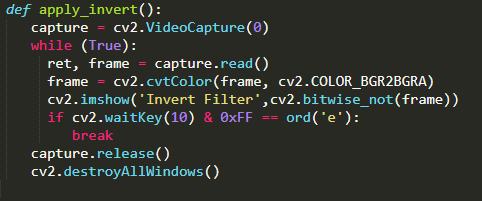


Figure 1.1 Apply invert function

**1.2 Apply Sepia Filter Function**

To get the sepia filter effect, we had to put a translucent overlay on our original frame. To create an overlay that consists of the blue, green, red, and alpha values, we used our verify\_alpha\_channel function to add an alpha channel into our frame. To get the color called sepia, we had to make research and use predefined numbers for the color values in our overlay. When creating the overlay, we’ve used the full method from NumPy to have an array that consists of the shape, color, and type we specified. We used 8-bit unsigned integers for the type, because of the compatibility reasons since Opencv reads the image in that particular type.

And then; using the addWeighted function, we managed to blend our original frame with the color overlay that has the values which form the color sepia. As a destination file for this operation, we used the same frame. So we had overwritten the original frame. We have set the intensity to 0.6, for translucency for the filter.



Figure 2.1 Apply sepia filter function

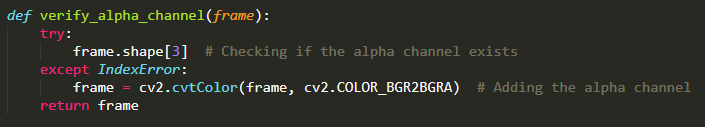


Figure 2.2 Verify alpha channel function

**1.3 Apply Color Overlay Function**

Inside the Interface.py file we added a mini interface which is the function above, it appears after pressing the Color Overlay Filter button. And that mini interface allows us to adjust the intensity, red, green, value values up to 255. So with this upgrade instead of adjusting manually in code, now we can apply intensity values with sliders thanks to the additional interface. In this code first, we assign all our variable types like Intvar() Doublevar() to our variables. We set the visuality and geometry of our buttons and interface(padx,pady geometry, row, column, etc.) We make sliders with Scale function and set the variables inside of that as a variable parameter. We adjust our words where is above the sliders(Intensity, Red, Green, Blue words for represented their sliders). Then with the apply button, we call the color\_overlay function with adjusted parameters thanks to the sliders. And we use blue\_value.get(), green\_value.get() like this for converting them to float inside of parameter.

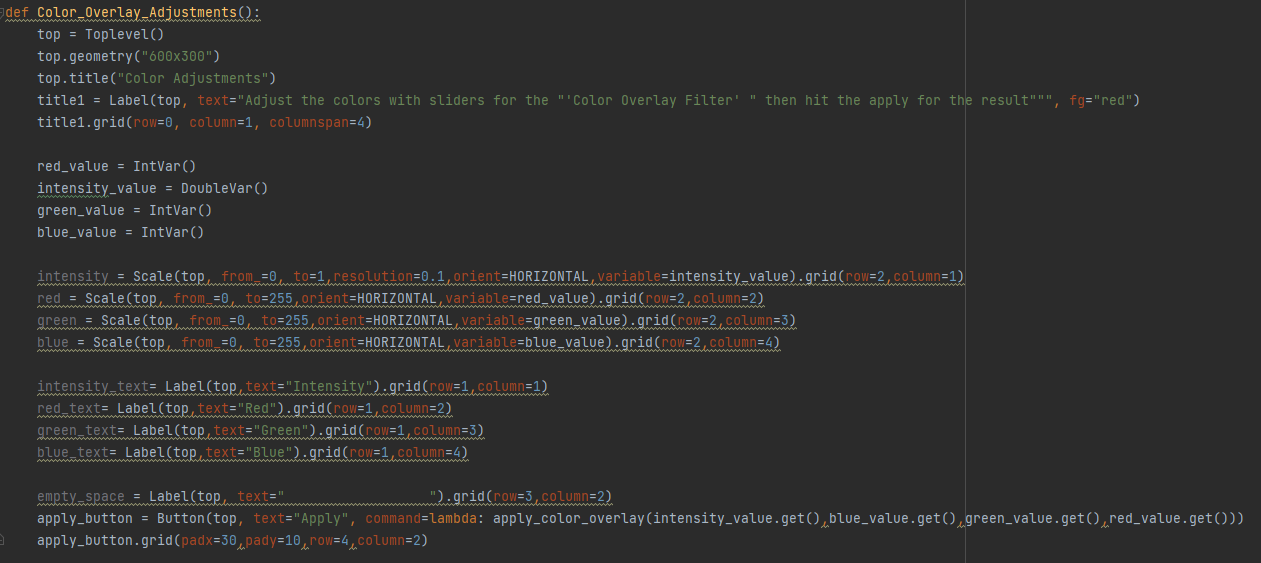


Figure 3.1 Operations in color overlay adjustments

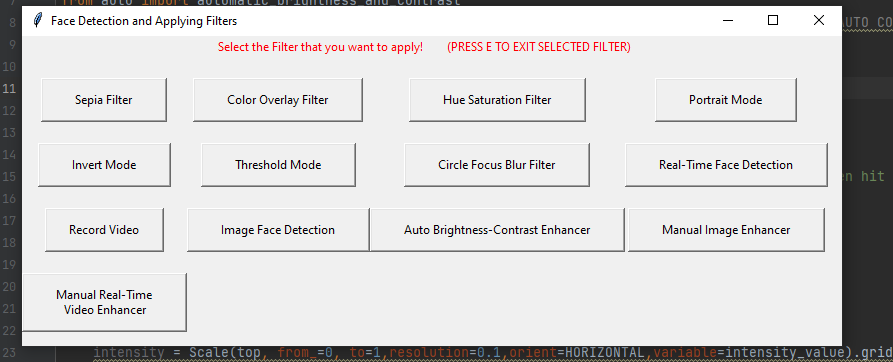


Figure 3.2 Output of the interface.py

So when we press the color overlay Filter button now we have a second mini interface that allows us to control color intensities. It appears like below. We can set intensity as a float variable from 0 to 10. The others are up to 255. After adjusting values with the slider when we press apply button our Default Webcam or external camera which is VideoCapture(0) appears on the screen. So when we press apply after adjusting red values to 218 the output looks reddish as we expected.

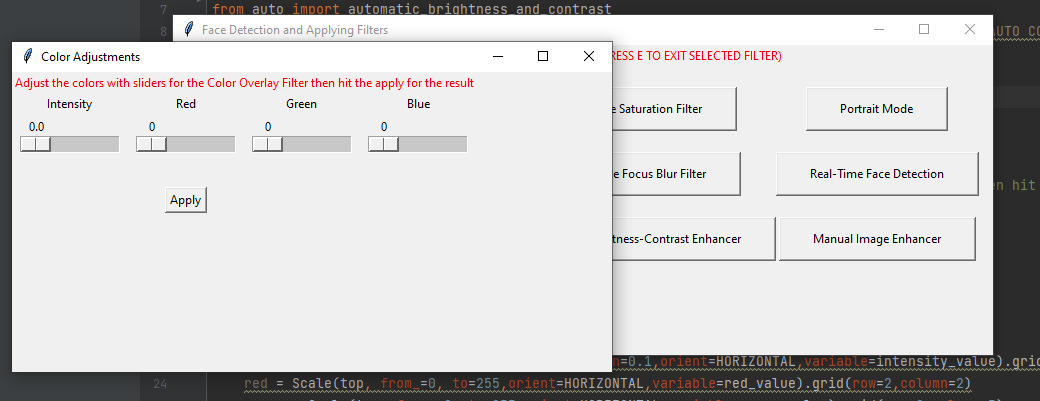


Figure 3.3 Output of the color overlay filter

So if we adjust all of them to the 255 and intensity to max (1.0). It is expected to be White in the below and that confirms it works properly.

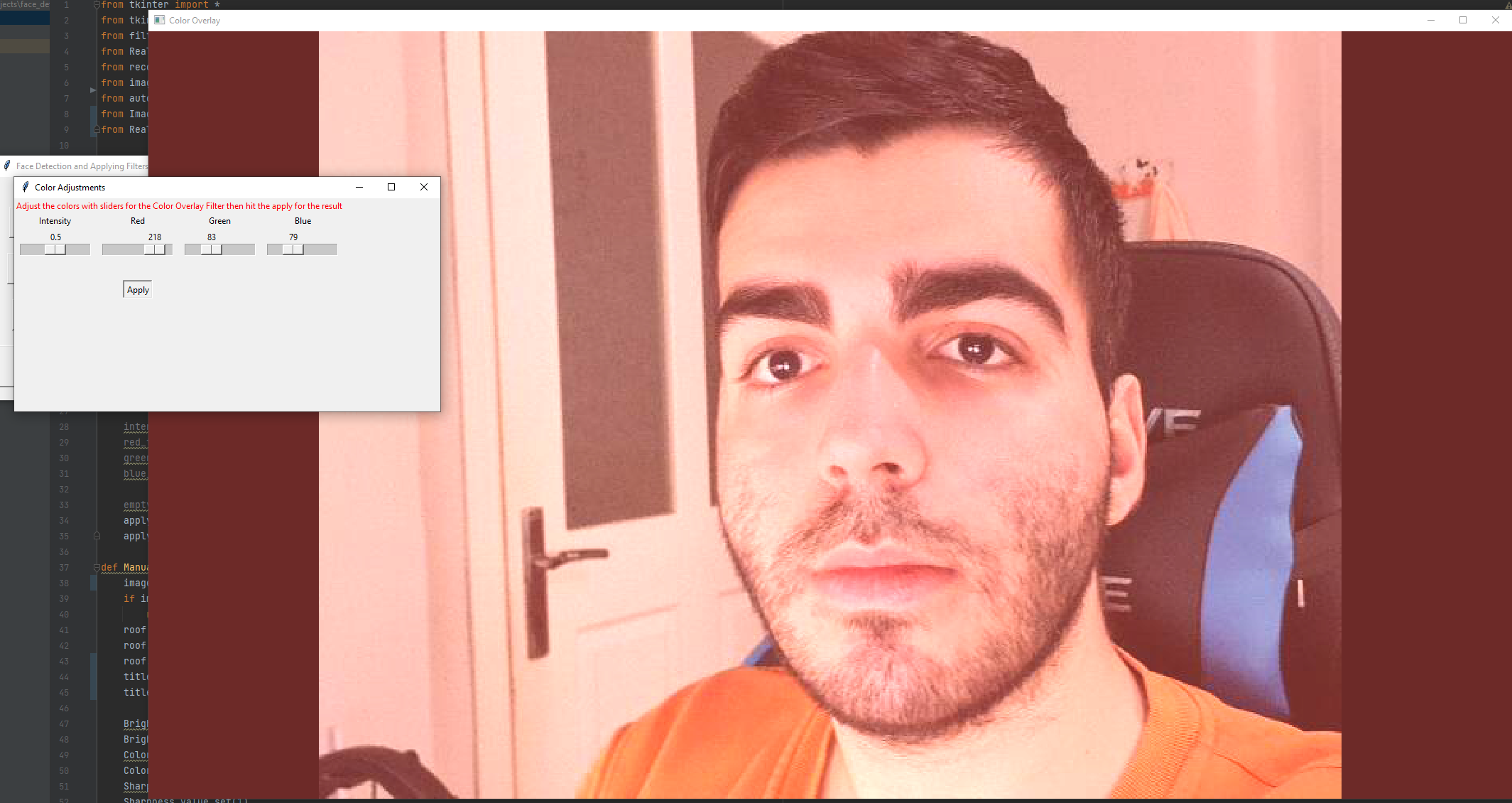


Figure 3.4 Second output of the color overlay filter

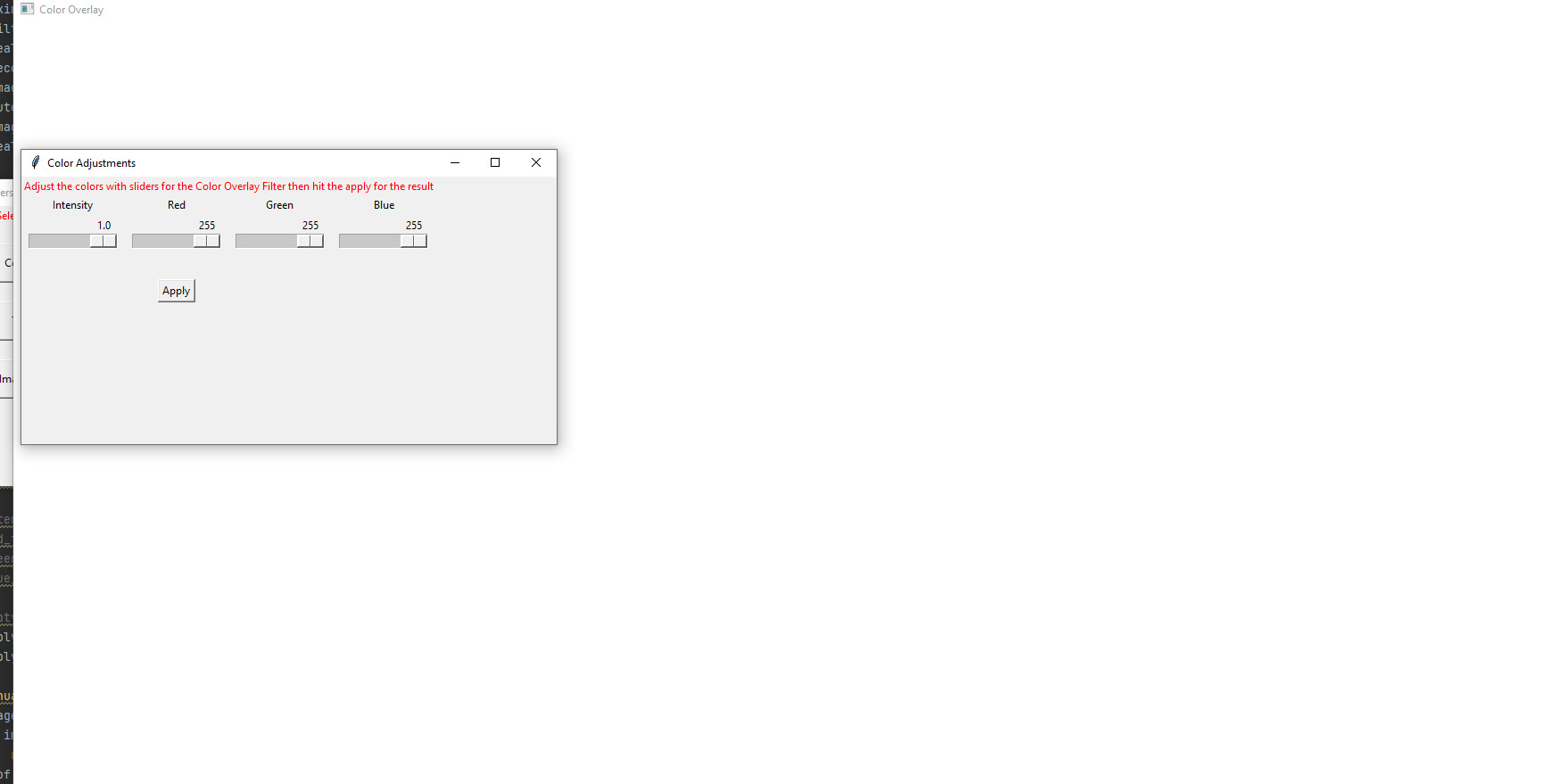


Figure 3.5 Third output of the color overlay filter

**1.4 Apply Hue Saturation Function**

Hue saturation value is a different color model than the blue, green, red color model which we usually use. This function takes a frame from the capture object as always, adds an alpha channel, then converts it into an HSV using the COLOR\_BGR2HSV method. Then splits it by using the split method to obtain the saturation and value. We assigned values manually into them using the fill method. After the splitting, we’ve merged those values and created the out frame. Before blending the out and frame, by calling the verify\_alpha\_channel we added the alpha channel and then finally displayed the result by calling the imshow method as always.

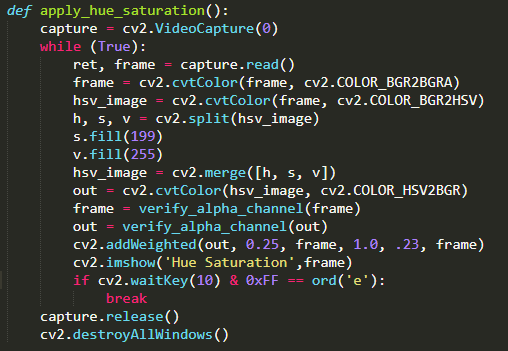


Figure 4.1 Apply hue saturation function

**1.5 Apply Circle Focus Blur Filter Function**

By creating this function, we targeted to achieve a frame that is fully blurred except for a circle right in the center. To get this frame, we had to blend the fully blurred frame with the circle mask we created. After getting the frame and adding the alpha channel into it, we’ve taken the full height and weight values, then divided them into 2 to get a circle radius. We’ve drawn a circle using the circle method around the radius. We blurred to the edge of the circle to get a smooth transition from the fully blurred area. We had to use the alpha\_blend function to get the output frame rescaled.



Figure 5.1 Apply circle focus blur filter function

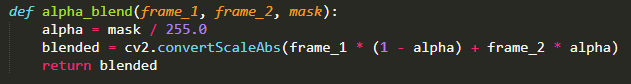


Figure 5.2 Alpha blend function

**1.6 Apply Threshold Mode Function**

We wanted to make a filter with the threshold values from the grayscale frame. After converting the original frame to grayscale, with the help of the THRESH\_BINARY method we were able to get our frame transformed to display threshold values.

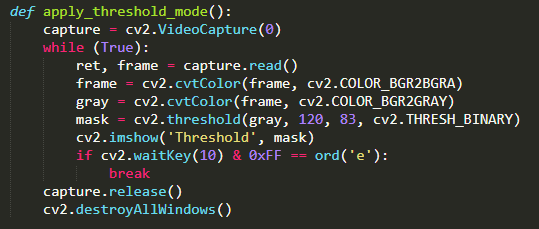


Figure 6.1 Apply threshold mode function

**1.7 Apply Portrait Mode Function**

With this function, we wanted to take the threshold function and combine it with the blurred frame to get a somehow caricaturistic visual. This way, we’ve used 2 different techniques we learned along with the project.

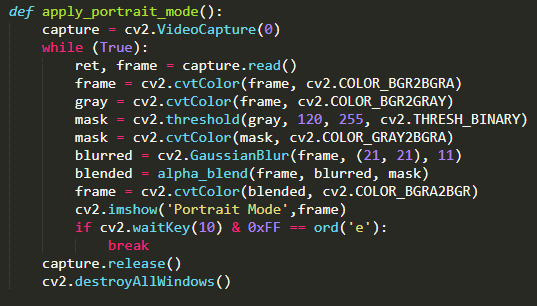


Figure 7.1 Apply portrait mode function

1. **REAL-TIME FACE DETECTION**

**2.1 Real-Time Face Detection Function**

Our goal was to combine image filtering techniques with real-time face detection. We’ve used the frontal face cascade classifier from the Opencv library. For drawing a rectangle around the face to indicate a positive detection, we first gathered coordinates of the face, and then by using the rectangle method, we were able to draw the rectangle of the color we picked. It is important to know that we’ve converted our frame into grayscale before using the cascade. The reason behind this is that the classifier can detect the human face without the color values easier. And also grayscale images cover less space, so it is preferred this way when applying these classifiers.

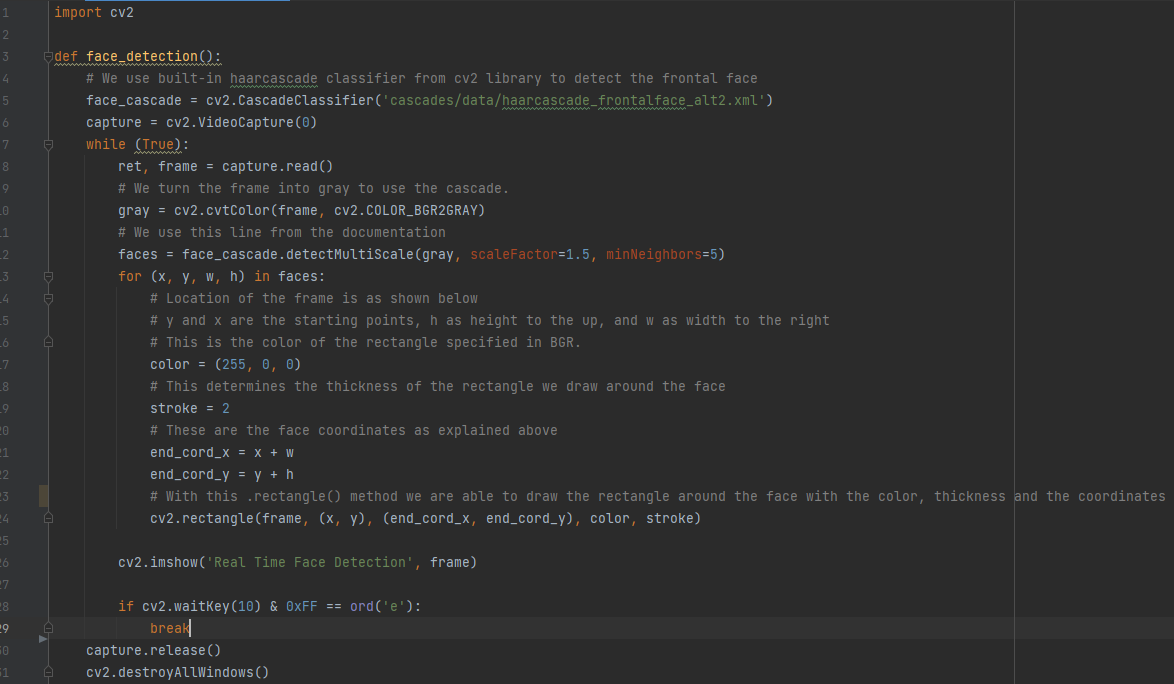
****

Figure 1.1 Real-time face detection function

1. **IMAGE FACE DETECTION**

**3.1 Image Face Detection Function**

As a small feature, we wanted to use the same frontal face cascade classifier to do image face detections. Just as we did with the real-time face detection, we converted our frame into grayscale then applied face cascade.

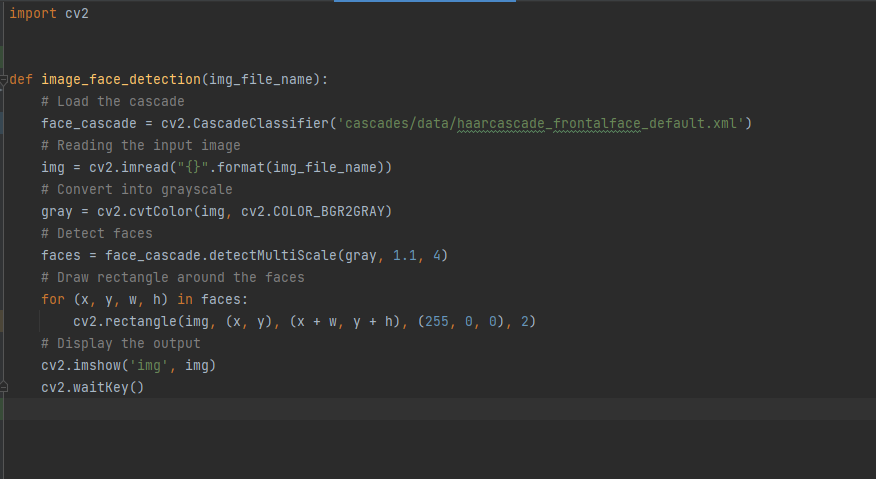
****

Figure 1.1 Image face detection function

1. **RECORDING VIDEOS**

**4.1 Record Video Function**

As another small feature, we wanted to keep track of the tests we did. To do this we created a record\_video function. Along with OpenCV, we had to import the os library as well since we were going to have access to the directories and write files. As a video format we picked MPEG-4 and audio video interleave for compatibility reasons with Opencv. We created a dictionary file called res\_sizes to get the height and width of the values for corresponding resolutions. A user can change the resolution when calling the get\_sizes function to get the right video sizes for the output file to write on. The get\_sizes function will call the change\_res function if the entered resolution is indeed in the dictionary file.

Again for compatibility reasons, we picked the video codec XVID, since it is suitable for working on both of the extensions and compatible with the Windows Operating System. In the get\_video\_type function, we set the default extension as audio-video interleave when there is not a name of the other extension in the entered output file name. We created the output file by using the VideoWriter method to write it under the while loop. Then finally to get a more practical utilization we imported file dialog from the Tkinter UI library to use the askdirectory() function to get a path by the user to save the output.

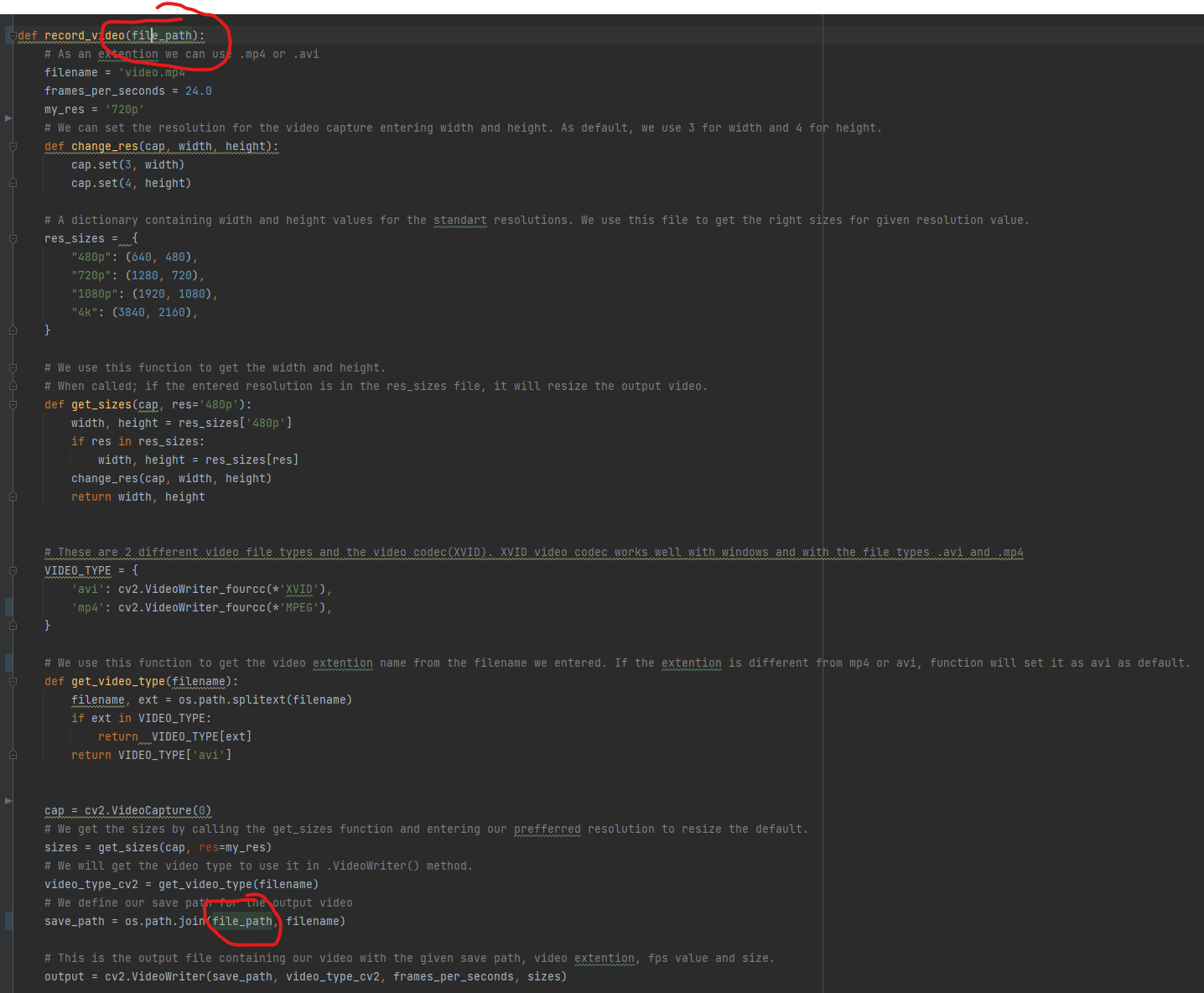


Figure 1.1 Record video function

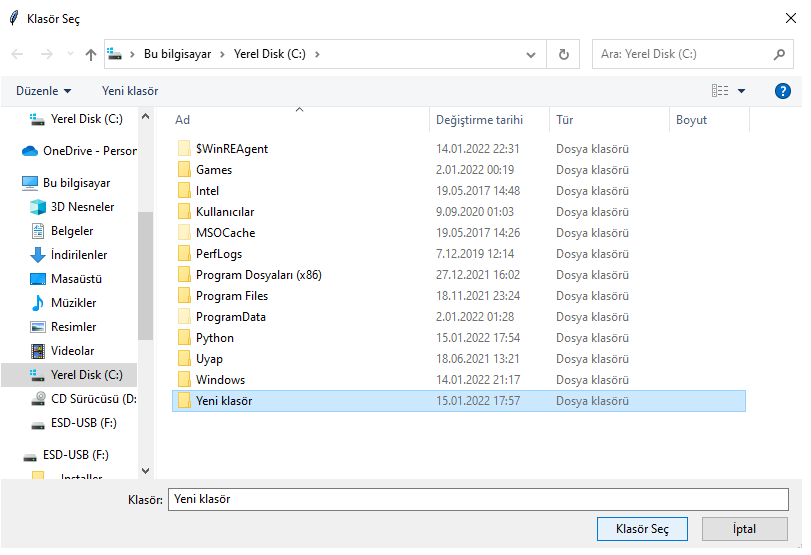


Figure 1.2 Output 1

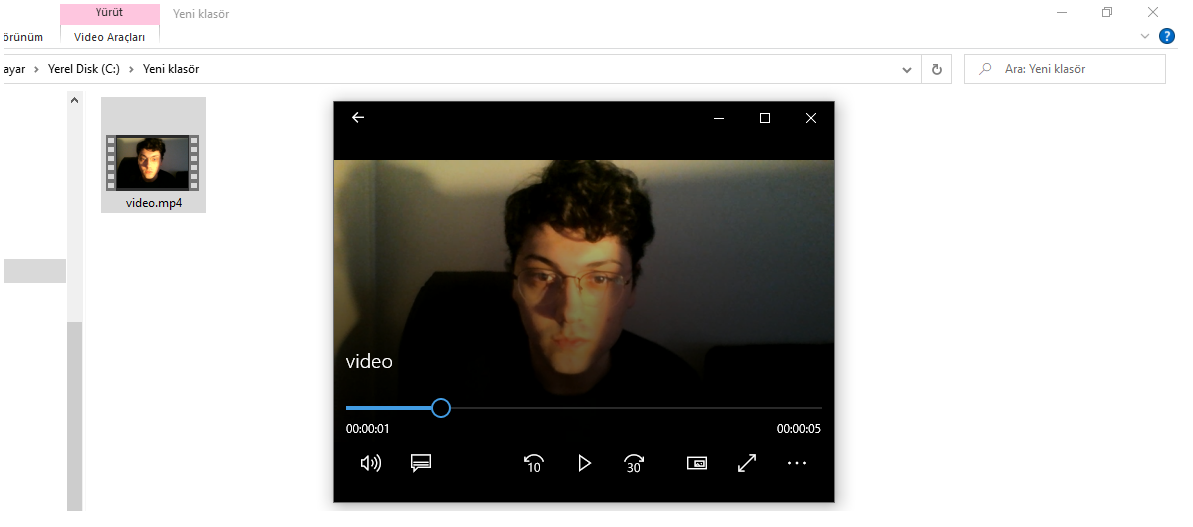


Figure 1.3 Output 2

1. **INTERFACE**

**5.1 Interface File**

We’ve used the Tkinter library to create our project’s graphical user interface. When calling the functions that we assigned to various buttons, if any of the functions had a parameter, we had to use Lambda to pass in the arguments.

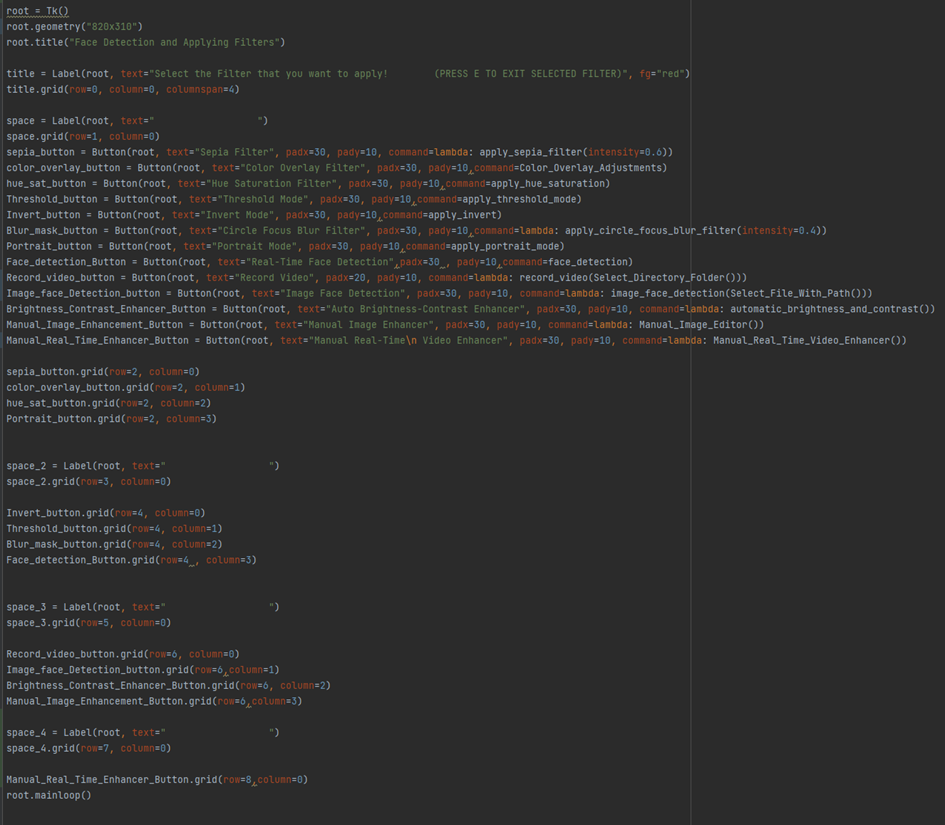


Figure 1.1 The interface file

**5.2 Modifications of the Interface**

We added these two functions inside of our Interface.py file for selecting Images to our Image\_face\_Detection and Manual\_Image\_Enhancer Tools or selecting directory For record\_video function for the path where we record.

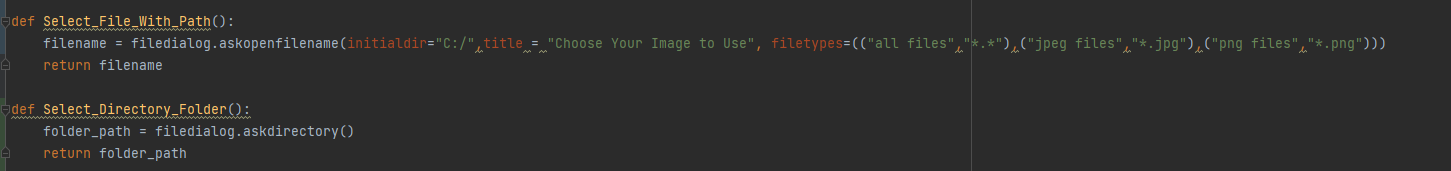


Figure 2.1 Modifications of the interface file

**6. AUTO BRIGHTNESS AND CONTRAST**

We wanted to create an algorithm that would read the frame and calculate its histogram values and evaluate it to get a better visual in terms of brightness and contrast. We consulted the document file of opencv[4] regarding the process of changing image values brightness and contrast which can be made for simple and single input frames. Although it is a popular topic to get useful information, it was quite a challenge to implement a way that we can execute in a while loop. That meant we could not handle the most complex derivations.

Essentially we’ve aimed to get evaluated alpha and beta values to operate on the input image which is a frame that has been taken in real-time to get the output image. To do that, firstly we converted the input into grayscale to achieve better performance. Then, using the OpenCV function calcHist() to get the histogram values of input. After storing these values temporarily to a list, we calculated the minimum and maximum gray values to get the alpha. And by using alpha we were able to get the beta value. Finally, as the following step, we performed the formula and display it.

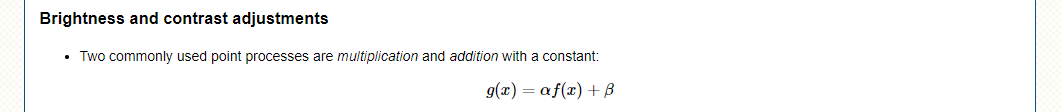


Figure 1.1 Function to get an adjusted image

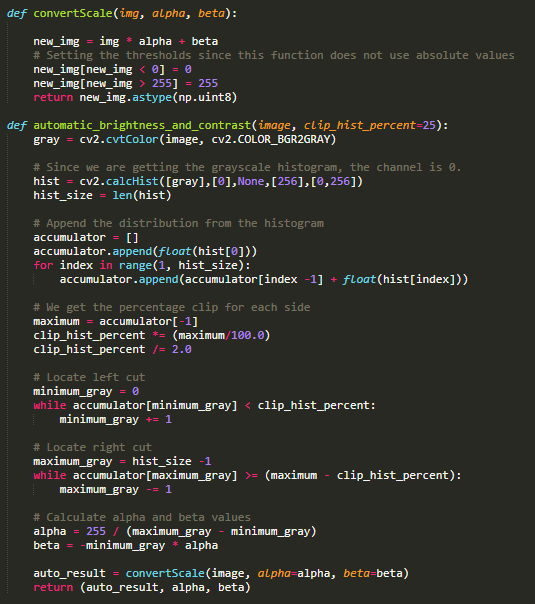


Figure 1.2 Functions to get new values and create a new image

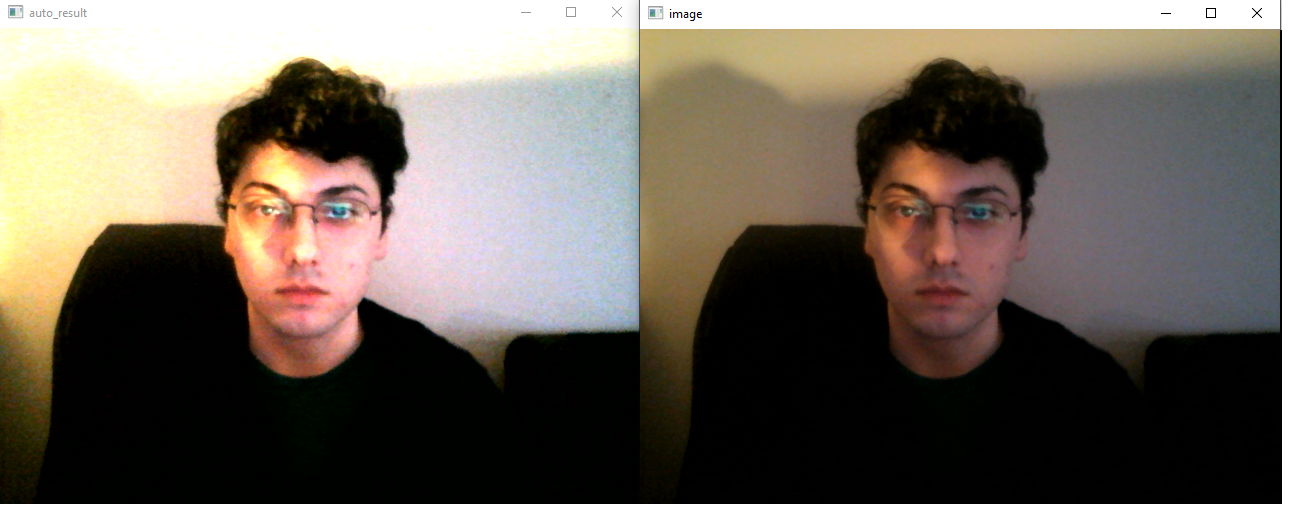


Figure 1.3 Output

**7. IMAGE ENHANCEMENT**

We tried to make an image editor look like photoshop but the basic version only related to enhancements. The enhancement part is combined into 4 parts which are saturation, contrast, sharpness, and brightness.

First, we ask the path to the user and let them select an image and it returns the path in the interface.py thanks to the Select\_File\_With\_Path() function. We kept enhancing over and over again and adding through assignments. We enhance the image then assign the other variable then enhance the enhanced image over and over again after brightness is added to the last image then we Show the result of the edited image.

First, we let users select the image from their storage and if the user selects nothing which is null then it does nothing and closes. We set the visuality and geometry again as we did before for our Color\_Overlay\_Filter. At the end of the code, we call our Manual\_Image\_Enhancement function as we call other functions. For that function, we assign parameters from our sliders value. It directs us to select an image file. After selecting a file and double-clicking. It redirects us to the mini interface that we created. I added brightness and color saturation. We can play like that until fine-tuning.

It lets us adjust values and it's set as default 1.0 which represents the original image. And we add a note on the bottom right corner for if we want to save an image we can simply press CTRL+S since it opened with Windows photo editor. Otherwise, it shows us a temp file, and when we close it nothing happens.

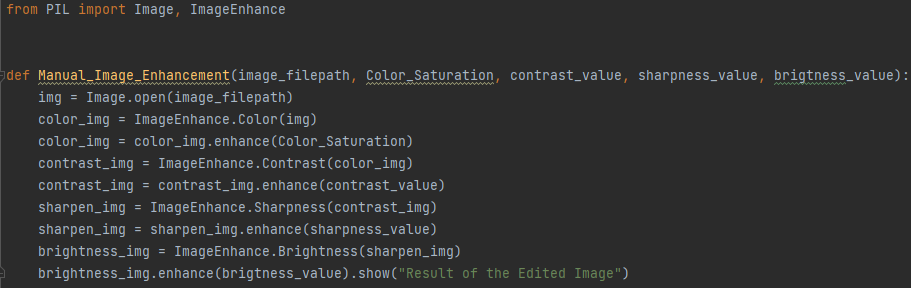


Figure 1.1 Manual image enhancement

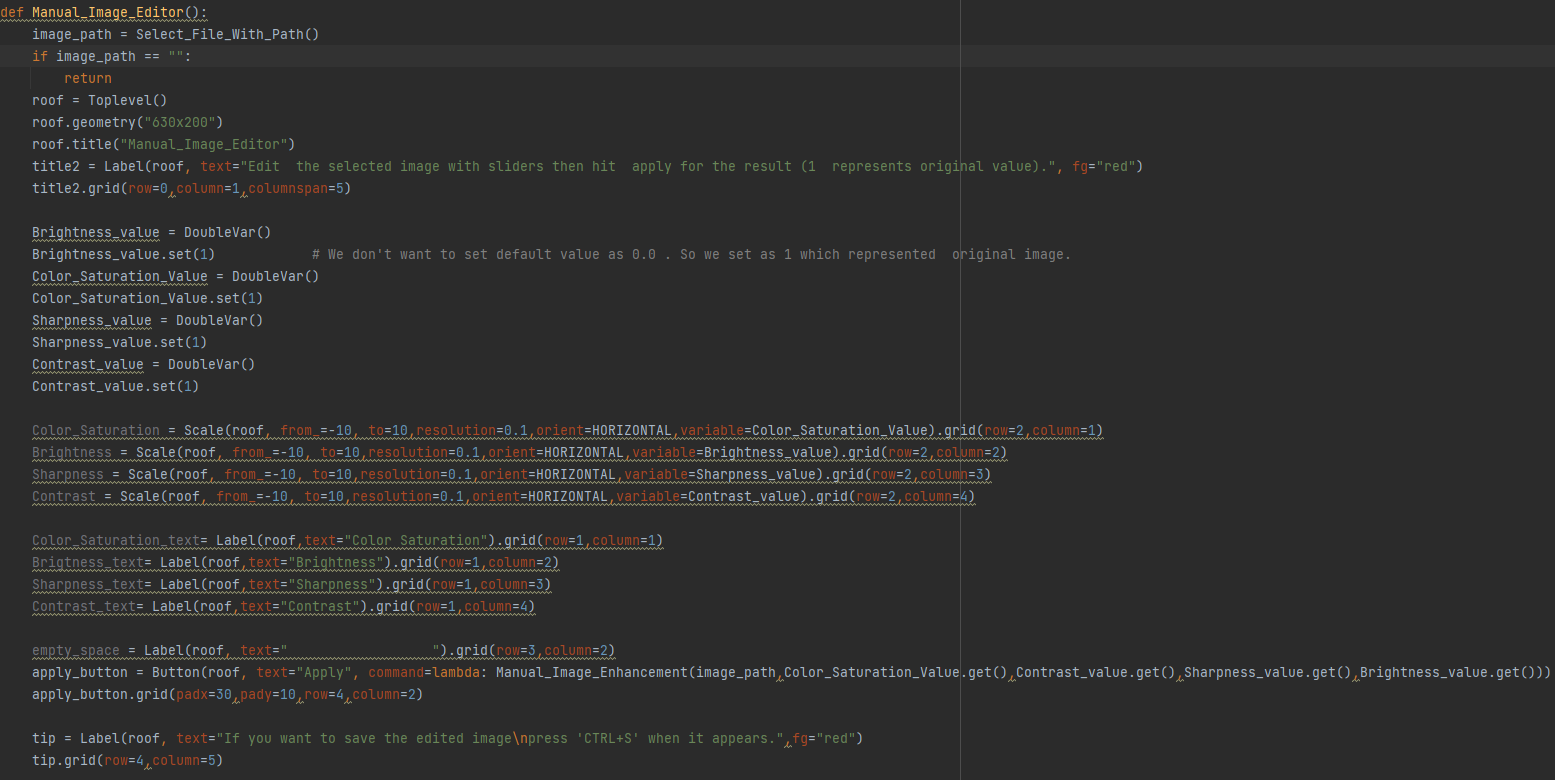


Figure 1.2 Manual image editor

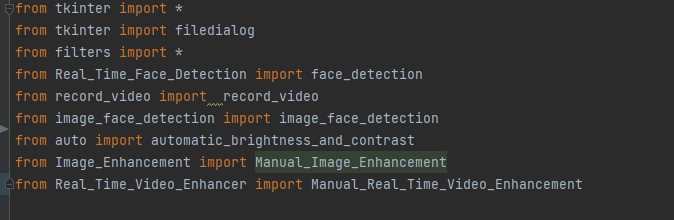


Figure 1.3 Importing the file

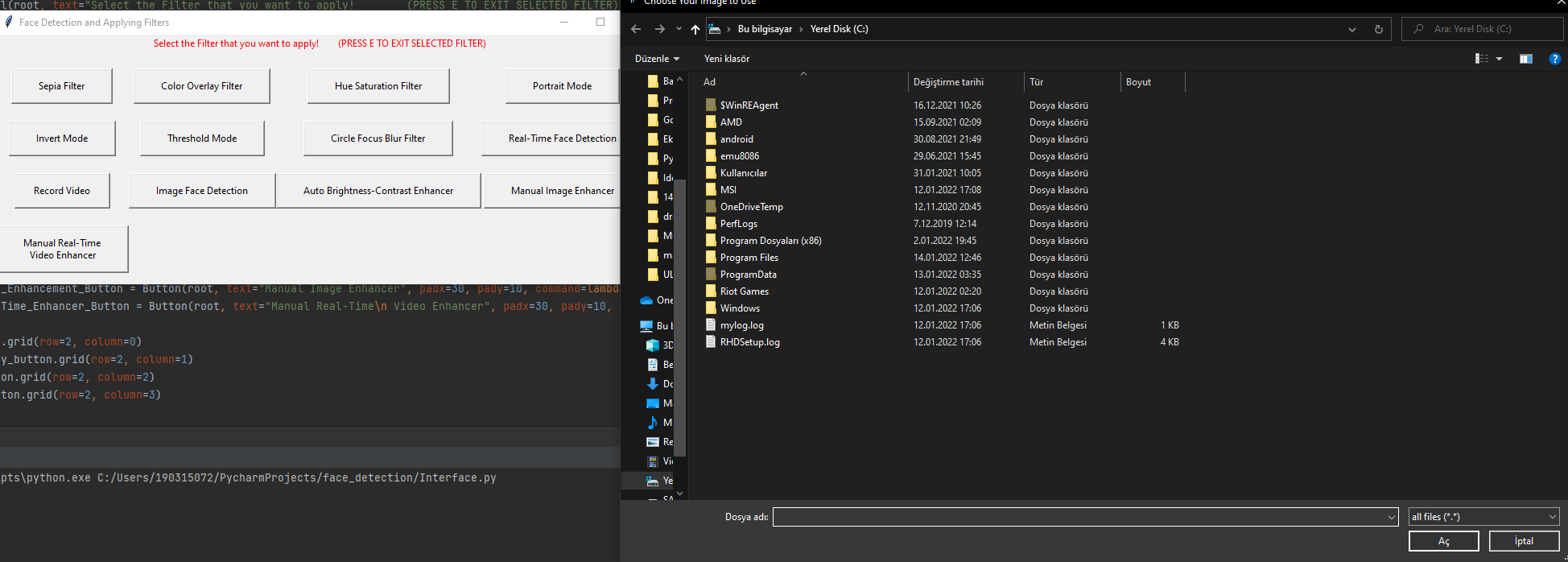


Figure 1.4 Output

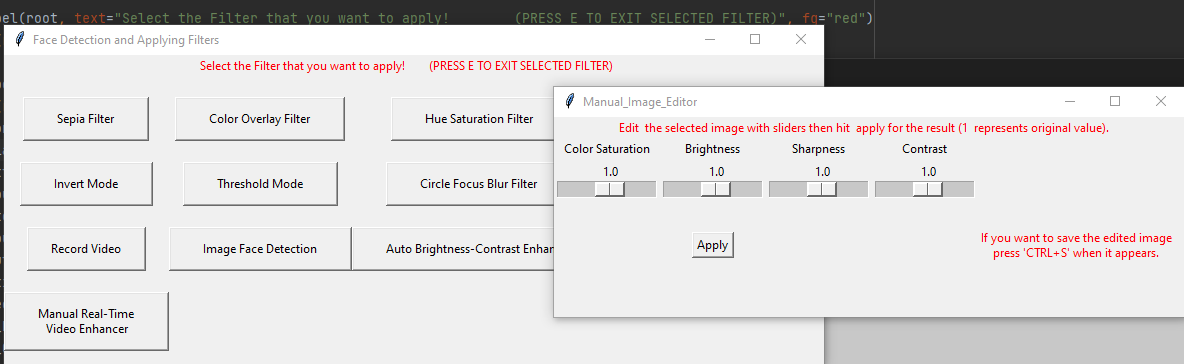


Figure 1.5 Second output



Figure 1.6 Original sample

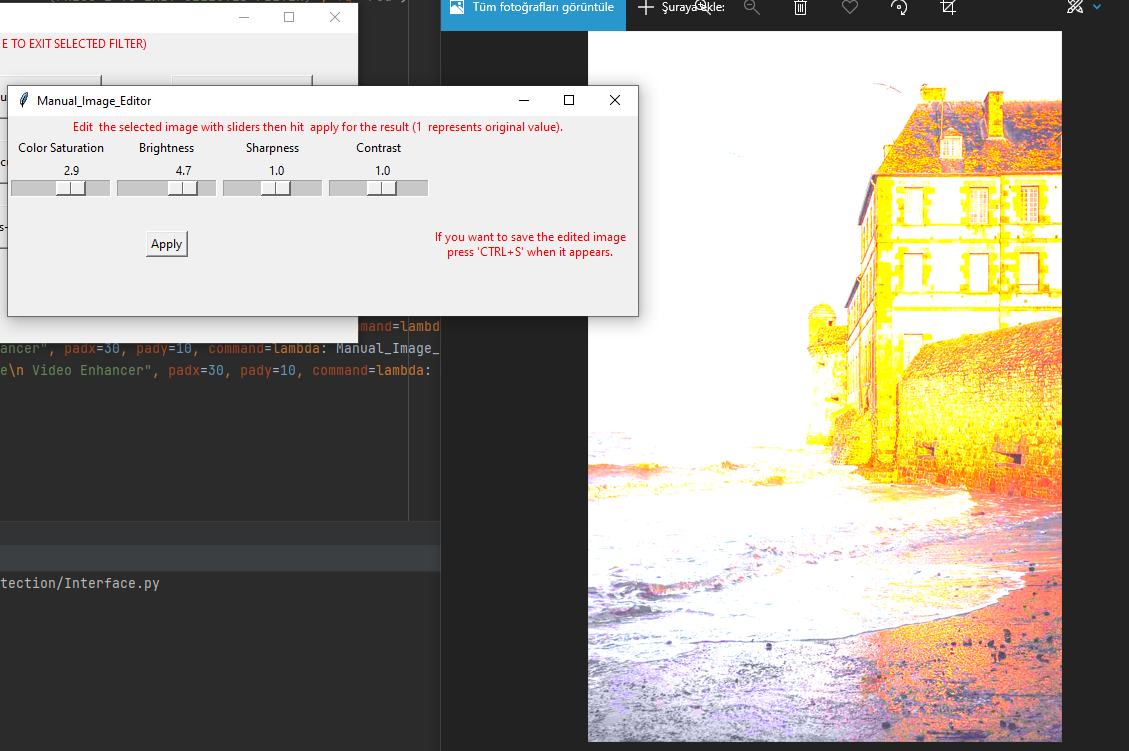


Figure 1.7 Edited version

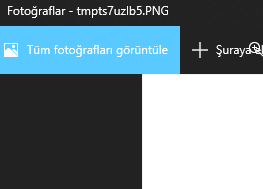


Figure 1.8 Output stored as a temp file

**8. REAL-TIME VIDEO ENHANCER**

It is the same logic as the Manual\_Image Enhancer but this is the real-time video version. There are tons of options but I select the options that give us the benefit of adjusting video quality. I can’t decide whether Should I add auto-focus, auto-exposure, and auto-white balance with tick options inside of the interface. But since they’re not working properly I decided to not add them. The zoom part seems interested to me so I add that. That’s the basic function first we set the parameters for our video. Then with a while loop, we make a video as frames added over and over I also set width height as default HD, and frame rate is 30.

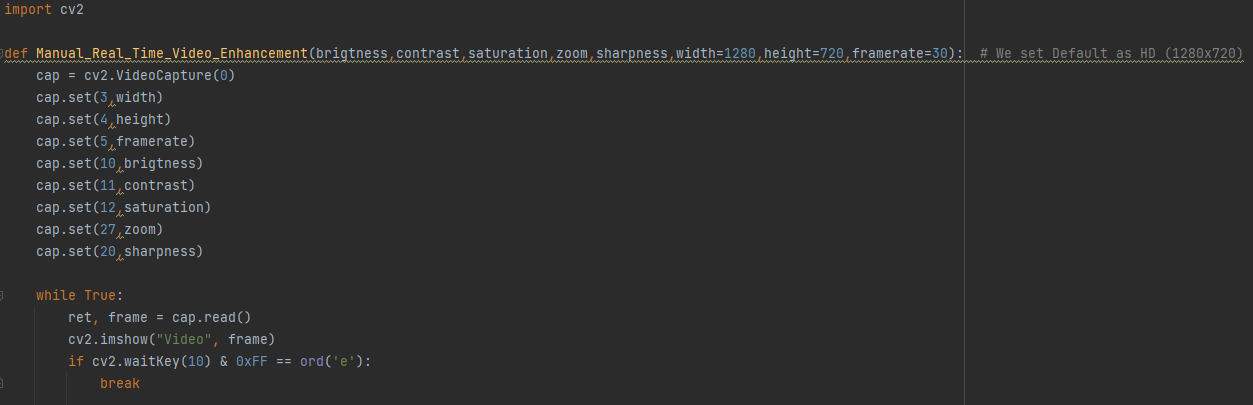


Figure 1.1 Manual real-time video enhancement

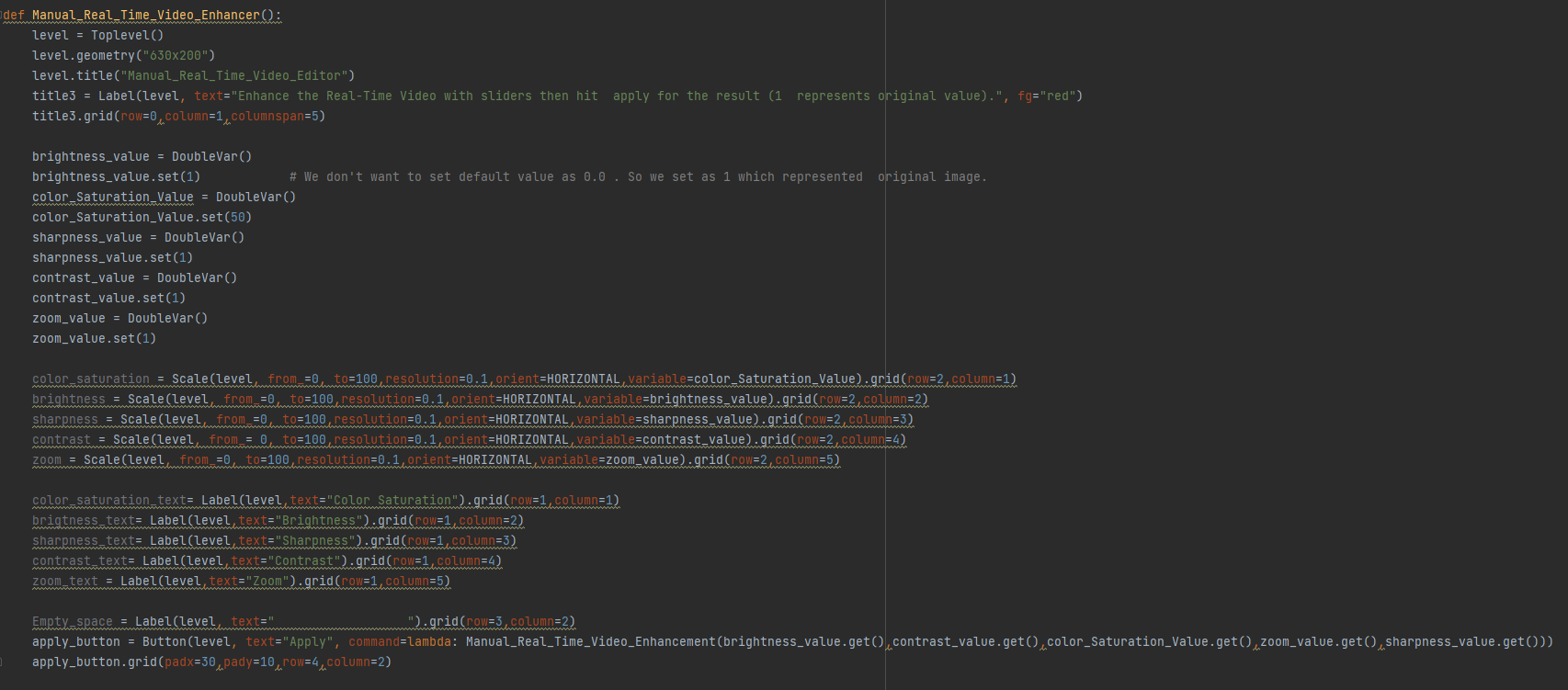


Figure 1.2 Implementation of the function in the interface

It’s the same logic as the others. I set default slider values to 1. So when we open sliders it appears as 1 first, then we adjust sliders for our pleasure. In apply\_button we call the function as we mentioned above. It shows up first we can set the values up to 100. When we apply our webcam shows up. But unfortunately, external Cam’s(such as connecting the phone with Droid Cam) doesn’t work because of encoding issues. I tried and it only works properly with a laptop webcam. External cam’s cause an issue.

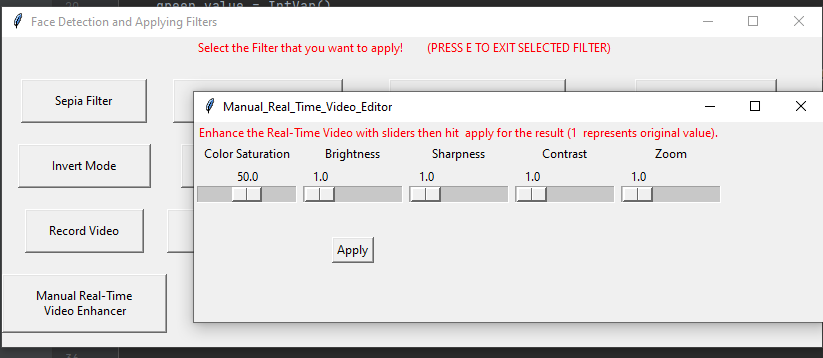


Figure 1.3 Output

**9. CONCLUSION**

We managed to get experience on different subjects like the implementation of various image filters using Opencv in real-time, implementing a cascade classifier, creating our graphical user interface with Tkinter, creating video output files in correspondence with Opencv, and getting instantaneously adjusted display. Also, we’ve gained abilities to handle diversified situations for single image manipulations in a real-time camera. Doing these manipulations with different tools we’ve created like sliders, allowed us a flexible control of both video and image display.

**REFERENCES**

**[1]** https://docs.python.org/3/library/tkinter.html

**[2]** <https://numpy.org>

**[3]** https://opencv.org

**[4]** <https://docs.opencv.org/4.x/d3/dc1/tutorial_basic_linear_transform.html>

**Code Links :**