REAL-TIME FACE SWAPPING APP WITH ADDITION OF OPTIONAL FILTERS

AI LAB: COMPUTER VISION AND NATURAL LANGUAGE PROCESSING
2022/2023

PETO BIRLEA ROXANA SORINA 1934897
SARACENO MICHELE 1905065
SINISI FEDERICA 1946981

1. INTRODUCTION

Computer Vision is a branch of Artificial Intelligence that allows computers to extract meaningful information from digital images, videos or other visual inputs in order to take actions or make recommendations based on that information. Computer Vision is similar to human sight, except for the fact that the latter has the advantage of lifetimes of context to train how to interpret what's in front of the eyes. A remedy to that problem is the training of the model that observes with lots of data, so it learns a pattern that can be applied to new data. The training of the model is done through techniques of Machine Learning, another field of Artificial Intelligence that deals with computers trying to learn how to process and react to data inputs.

Therefore Computer Vision can be applied, amongst other things, to detect human faces in visual inputs. In order to do that we used dlib, a toolkit containing a variety of machine learning algorithms, and in particular the facial recognition tools, such as a face detector and a shape predictor.

To add features to our project we also implemented a user interface that allows the user to select that reference image for the face swapping and optionally some filters, which are Blue Eyes, Cartoonize and Splash, that can be applied both to the swapped face and the plain captured image from the video camera.

2. IMPLEMENTATION OF FACE SWAPPING

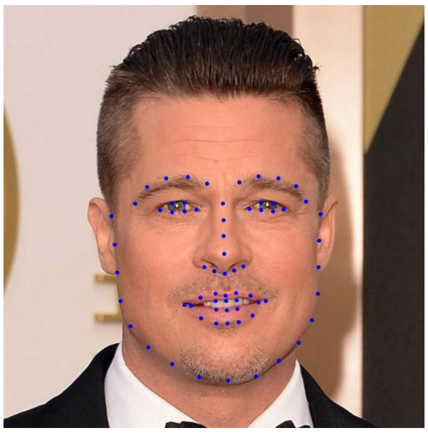
2.1. IMPORTS

The required libraries for this part of the project are OpenCV, Numpy and Dlib. While *OpenCV* is used to handle and modify the images and *Numpy* to work with multidimensional arrays, the *Dlib* library is used to facilitate the face detection and shape prediction of the face. From this library we got two functions: get_frontal_face_detector() and shape_predictor(). While the first function returns a detector that can point out faces on the image, the second one returns a detector that, given the image and the area of the detected face, can predict the 68 coordinates (x, y) that map the facial points on a person's face.

2.2. DETECT FACIAL LANDMARKS

In order to do the face swapping we need to detect the 68 facial points on both the reference image and the live camera (for simplicity, in case there are multiple faces, we use the biggest face detected in both cases), so for each detected point we store their coordinates in an array.

On the left the reference image, on the right the face to be replaced.

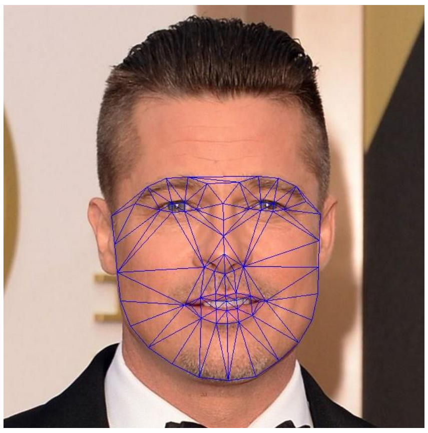


Landmarks of the face.

2.3. DELAUNAY TRIANGULATION

Since the reference image and the live camera have different size and perspective, other than the number of dimensions (2D for the first image and 3D for the second one) [1], we have to segment the area of the face into small parts that allow the swapping to be as smooth as possible and maintain any movements. The Delaunay triangulation is a mathematical concept that allows the grouping of discrete points into sets of three in such a way that no other point can be found inside the circumcircle of any triangle.

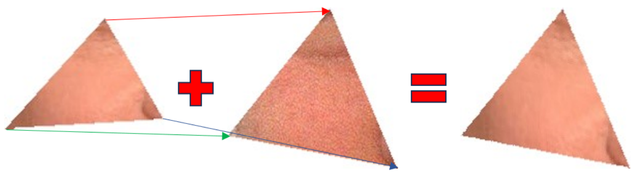
To do this, a convex hull of the landmarks is created, which is the smallest convex set that contains the points, and then used to create a planar subdivision on a set of 2D points, containing the bounding rectangle of the hull. Then we obtain a list of triangles, which we process by mapping the coordinates of the vertices into the point's indexes in the array of landmarks. We only do the triangulation once because the connection of the points on the live camera has to be the same as in the reference image.



Face triangulation using the 68 landmarks.

2.4. TRIANGLES WARPING

Afterwards, these triangles are used one at a time, for both the reference image and the live camera, to isolate the area of interest and within it create a mask where only the triangle is visible. Then the reference triangle is warped using as source points the original coordinates of the vertices and as destination points the coordinates of the vertices of the live camera triangle. By putting all the triangles together the face is rebuilt, this time with the size and perspective of the face detected on live camera.



Example of triangle warping.

2.5. OVERLAPPING OF FACES

In order to overlap the frame and the new face, two masks are created, one with only the swapped face, obtained in the previous part, and one with the background, eyes and mouth, so that the original parts are preserved. These are then put together by selecting only the colored parts from both face and background mask.

The reference face is added to the background.

2.6. IMAGES MERGING

For the purpose of better merging the images, since they can have different colorization and lighting, seamless cloning is applied to the result. The seamless clone method makes the composition look smoother and more natural.

Seamless cloning is applied to the result.

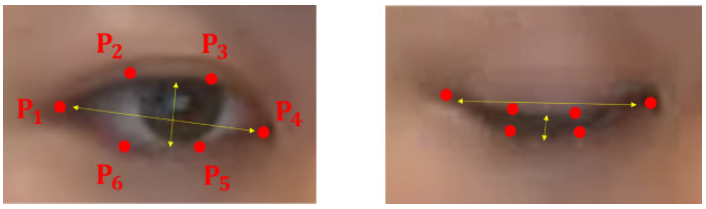
3. FILTERS

3.1 Blue eyes

This filter uses the facial landmarks, particularly the eyes landmarks, to change the iris of the eye. We resize the image of the blue iris using the eyes' landmarks and then apply a mask. To detect in real time when a person blinks we use the eye aspect ratio [4], which is obtained by finding the euclidean distances and using the following formula:

$$EAR = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

where p1,...,p6 are the points depicted in the image below. Due to the simplicity of the formula, it has excellent real-time performance but with the flaw that detection is difficult at the threshold and it causes the loss of accuracy when there is a sudden movement of the face or it is too far away.

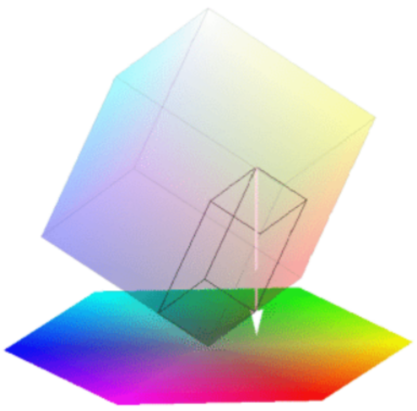


3.2 Color splash

This filter turns a colored image into grayscale except for one color that is still shown.

To detect the color, we first need to convert the image to the HSV (for hue, saturation, value) color space. *RGB* is suitable for color display, but not good for color scene segmentation and analysis because of the high correlation among the *R*, *G*, and *B* components. By high correlation, we mean that if the intensity changes, all the three components will change accordingly. Also, the measurement of a color in *RGB* space does not

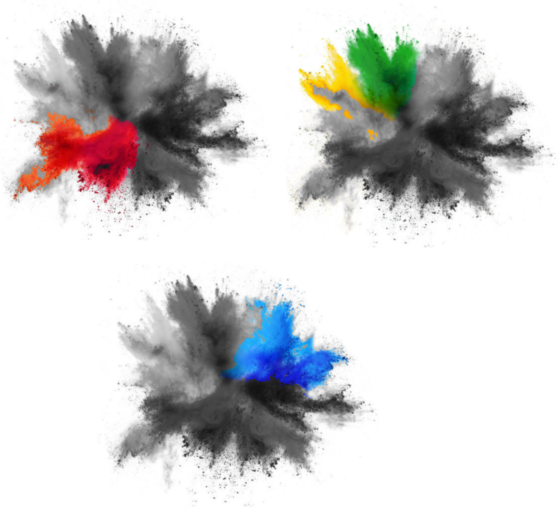
represent color differences in a uniform scale, hence, it is impossible to evaluate the similarity of two colors from their distance in *RGB* space.[2] HSV is often used in computer vision and image analysis for feature detection or image segmentation.
If we take a tilted RGB cube and project it, we obtain a hexagon with red, yellow, green, cyan, blue and magenta as its corners.


Projection of the RGB cube.

The Hue is the angle of the vector to a point in the projection with red at 0°. The Hue is also described as "Attribute of a visual sensation according to which an area appears to be similar to one of the perceived colors: red, yellow, green, and blue, or to a combination of two of them" [3].

The colored reference image.

In openCV the Hue ranges from 0 to 180 instead of 360, therefore red is mapped to the ranges of 0-10 and 160-180, blue is mapped between 120 and 140 and green and yellow to 20-80. If we make a mask with the inRange() function we can get the results shown below by joining the color region with the grayscale region (after converting it to 3 channels) of the image.


Results of the color splash filter.

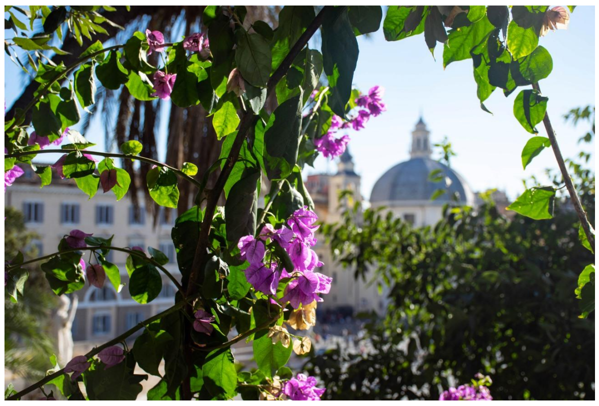
3.3 Cartoon

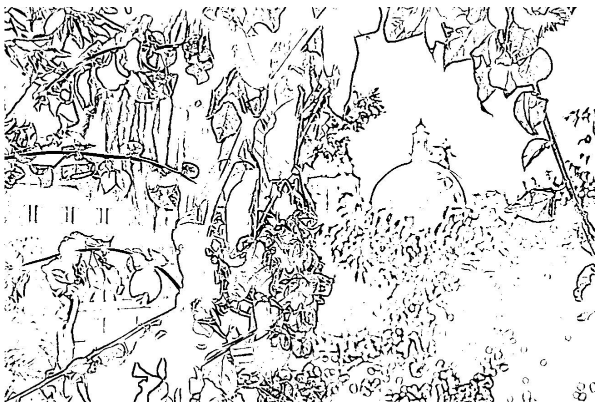
We saw this filter during the lessons. It uses the median Blur which is an averaging method usually used to preserve edges while removing noise. It calculates the median of the pixels in the kernel area and replaces the central element with this value.
After that we use the adaptive Threshold, which is a segmenting images method used to create a binary image. Each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally

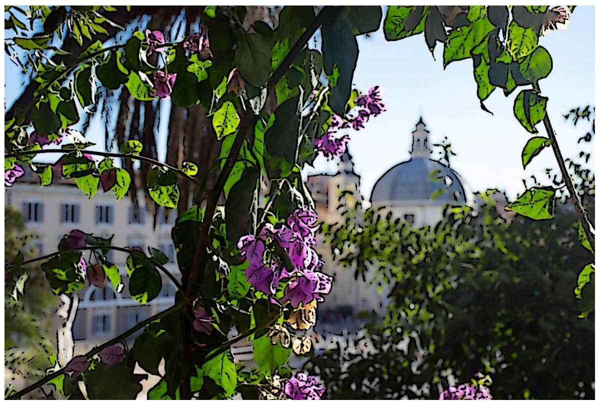
255). As opposed to global thresholding, the local (or adaptive) methods are particularly adapted to cases where images have inhomogeneous lighting. In this case, a neighborhood is defined and a threshold is computed for each pixel and its neighborhood.

We use the bilateral filter, which smooths the input image while preserving the edges, because it acts strongly on regions of uniform color, and lightly on regions with high color variance.

Lastly we add the edges to the smooth image we obtained with the bilateral filter.


The reference image.

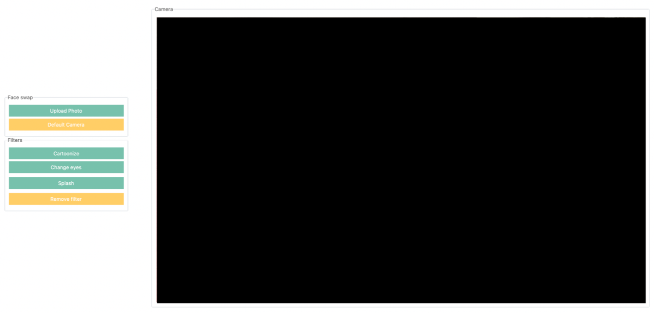

The edges of the image.


The cartoonized image.

4. USER INTERFACE

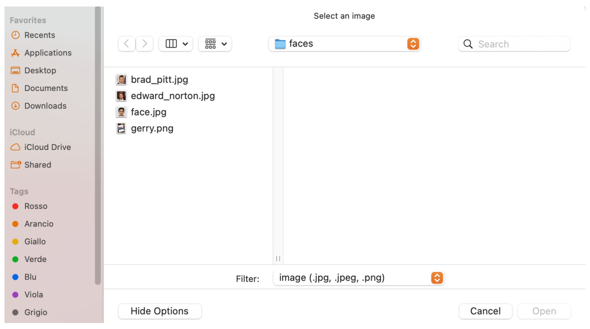
To avoid rerunning the code every time the user wants to choose a different filter or a different image to swap the face with, we decided to provide a graphical user interface.

We mainly used the library *ttkbootstrap*, which is an extension of *tkinter* that enables modern themes and aesthetics to the interface while keeping the same functionalities. To show images and camera frames with *tkinter* we need to convert them to the *ImageTk* format, with the use of the library *PIL* (Python Imaging Library).



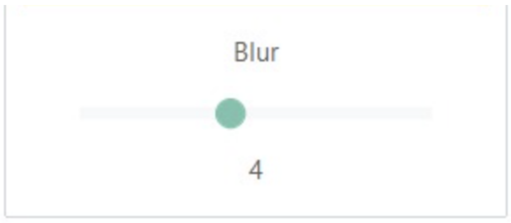
On the right side we have the camera section: here users can see their camera and the effects they apply to it.

On the left side we have the effects menu, divided in “Face Swap” and “Filters”.
In “Face Swap”, users can choose a photo, selected from the file system through a file explorer, to swap the face using the “Upload Photo” button.



Filter: image (.jpg, .jpeg, .png)

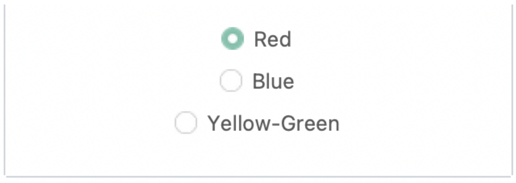
With the “Default Camera” button, the camera will turn back to normal canceling every effect that was applied on it (both face swap and filters).
In “Filters” the user can select a filter between the ones we listed in section 3 to apply to the camera. The filters can be applied one at the time, even on a face-swapped camera.
Some filters may have a value which can be chosen by the user.
For example, with the filter “Cartoonize”, the user can select the amount of blur they want to apply;



Blur

4

with “Splash” they can select the color to use for the filter between red, blue and yellow.



[✓ Red]
Blue
Yellow-Green

With the “Remove Filter” button, the user can remove any active filter they have on the camera without removing any face swap effect applied to it.

5. REFERENCES

1. A. Datta, O. K. Yadav, Y. Singh, S. S, K. M and S. E, "Real-Time Face Swapping System using OpenCV," 2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2021, pp. 1081-1086, doi: 10.1109/ICIRCA51532.2021.9545010.
2. H.D. Cheng, X.H. Jiang, Y. Sun, Jingli Wang, Color image segmentation: advances and prospects,Pattern Recognition,Volume 34, Issue 12, 2001, Pages 2259-2281, ISSN 0031-3203, https://doi.org/10.1016/S0031-3203(00)00149-7.
3. Fairchild, Mark D. (2005). *Color Appearance Models* (2nd ed.)
4. Akihiro Kuwahara, Kazu Nishikawa, Rin Hirakawa, Hideaki Kawano, Yoshihisa Nakatoh, Eye fatigue estimation using blink detection based on Eye Aspect Ratio Mapping (EARM), Cognitive Robotics, Volume 2, 2022, Pages 50-59, ISSN 2667-2413, https://doi.org/10.1016/j.cogr.2022.01.003.