University of Trento

Signal, Image and Video Final Project

Background Segmentation and Substitution

F. Diprima
federico.diprima@studenti.unitn.it

M. Farina
matteo.farina-1@studenti.unitn.it

15 April 2021

Contents

1 Introduction

The goal is to produce a segmentation map that imitates videocalls tools like the ones implemented in Google Meet or Zoom without using Deep Learning- or Machine Learning-based techniques. This document presents, step by step, the refining and incremental procedure that we have followed to achieve the final output:

1. The main idea started from a "simple" and intuitive frame differencing in the grayscale domain;
2. Then, this was revisited by migrating to a more effective implementation in the HSV colorpsace, where both brightness and saturation differences are taken into account leading to a boost in the quality of the final output.
3. Having found a safe place in the HSV domain, the third step consisted of adding user interaction by means of customizable trackbars used to configure the values for saturation and brightness thresholding. This turned out to be effective and led the algorithm to more stable results even in somehow complex environments.
4. Lastly, gaining stability at the cost of smoothness, users are given the possibility to use the Lucas-Kanade optical flow implementation in the OpenCV library and keep track of a satisfactory foreground mask over time. More on the meaning of this sentence will obviously be explained in the dedicated section.

Each script also makes use of three additional steps that, even though they are not strictly required, have proved to be very effective and useful in enhancing the quality of the algorithms and, above all, in handling noise:

- **Background Modeling:** the background is acquired and averaged over a time window;
- **Morphological Operators:** each mask in the scripts is passed through a series of closures and a final dilation;
- **Low-Pass Filtering:** Each acquired frame is convolved with a Gaussian Kernel, regardless of whether it should be used for background modeling or frame differencing. Moreover, in the case of saturation masks, due to their observed tendency to sparse blinking, a strong median filter is applied.

2 Grayscale domain

To start testing our algorithm we decided to work in the simplest possible color space, the grayscale. So, we converted the video captured by our webcam from BGR (default in the OpenCV Python library) to GRAY.

2.1 Create a background model

The first step to perform background segmentation is to create a background model of the scene. To this extent, it is important that the user leaves the acquired portion of environment recorded by the camera for the first 3 seconds of the video, approximately. In these few seconds our program takes care of capturing the surrounding background and applying a Gaussian filter on it, in order to make it more stable and robust to the most frequent noise settings. At the end of this step, an average of all the acquired frames is computed and used as the final background model.

2.2 Frame differencing in the gray-scale domain

After the creation of the background model the user is free to enter the scene. From now on, what our algorithm does is make a frame difference between the background model and the new scene captured by the webcam. In **figure 1** we can see the result of the frame difference in a particular moment of our recording:

- black pixels are those in which the difference is very small (close to zero) and should represent the pixels of the background;
- on the contrary, brighter pixels are those that have been subject to greater variations and are, thus, the ones that represent the user's entry into the scene.

Figure 1: Frame differencing in gray scale domain

2.3 Binary thresholding

After making the frame difference, the next step is to threshold it in order to obtain a binary image representing the foreground mask. With an OpenCV built-in function, this task can be accomplished in a very simple way. It is remarkable to say that for the choice of the most appropriate threshold value for our image we used the so-called **Otsu's method**, which calculates the optimal threshold to separate the two classes (black and white) by minimizing intra-class intensity variance and maximizing inter-class variance at the same time. In **figure 2** we can see an example output of this thresholding technique.

Figure 2: Binary foreground mask

2.4 Morphological operators

As we can see in the figure above, although it contains some empty spots, the mask is quite regular. To obtain this result we used some morphological filters, in particular a series of closures followed by a final dilation. It is also worth mentioning the choice of the structuring element used: it is a 3x5 ellipse that develops vertically. The idea was to have a structuring element which resembled as much as possible the generic shape of a human face, which is exactly the target object to be filled.

2.5 Final output

To view the final output we replaced the foreground mask with the scene captured by our cam and the background with an image of your choice. As we can see in **figure 3** the result obtained is acceptable but there are some imperfections that can be improved. In the next chapter we will try to work on a different color space to see if we can get better performances.

Figure 3: Output in gray scale domain

3 HSV domain

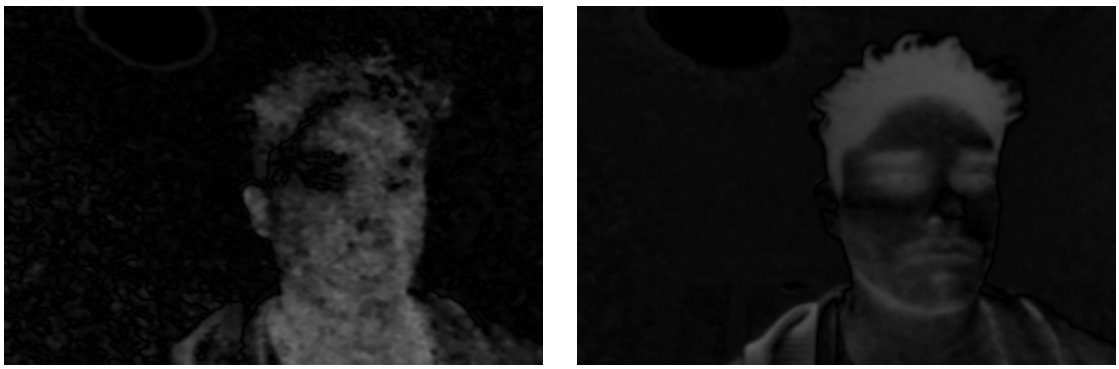
As anticipated in the previous chapter we decided to work in another colorspace in order to obtain better results. After several attempts, we chose HSV. HSV is a color space particularly oriented to the human perspective, being based on the perception of a color in terms of Hue, Saturation and Brightness (Value). The HSV color space is represented by a cone: the hue (H) is measured from an angle around the vertical axis, with red at 0 degrees, green at 120, and blue at 240. The height of the model represents the brightness (V) with zero being black and one being white. The saturation (S) instead goes from zero, on the model axis, to one on its surface, directed in an horizontal direction (orthogonal to the V axis).
With that said, the first step of the algorithm is conceptually the same as before: colorspace conversion is applied to the incoming frames to start working on the designated domain.

3.1 Create a background model in the HSV domain

The process of creating the background model is the same as before, the only difference is that now the averaged result is achieved combining three different channels instead of only one.

3.2 Frame differencing in the HSV domain

Also in the frame difference the concept is pretty much the same as before. However, we are in a color space composed of 3 channels: instead of a single difference image we actually have 3 of them (one for each component H, S and V). In **figure 4** we can see the frame difference on the Saturation (**4a**) and Value (**4b**) channels. We chose these two channels because they are the ones that best represent the difference between background and foreground.



(a) (b)

Figure 4: Saturation and Value differencing

3.3 Binary thresholding on the Saturation and the Value differences

Exactly as we did in the previous case (grayscale) we apply thresholding using Otsu's method. Obviously, because we are taken into account two different components, it is trivial but worth pointing out that in this case there are 2 different thresholded images, each with its inferred threshold value. In **figure 5** the results of this operation are shown.

(a) (b)

Figure 5: Saturation and Value binary masks

3.4 Combination of the two masks

Looking at the results obtained, we realized that the two masks are approximately complementary. Therefore, we decided to put them together in a single binary image by means of a bitwise-or operator. As you can see, the output represents the foreground very well.

Figure 6: Binary mask obtained from the combination of value and saturation

3.5 Median filtering and morphological operators

To improve the contours of our mask and make it more stable we have applied some simple operations:

- A median filter to remove salt noise in the saturation mask;
- A series of closures followed by a dilation, as in the previous Grayscale-based implementation.

,

3.6 Final output

To plot the final output we replaced our foreground mask with the scene captured by our cam and the background with an image of your choice. In **figure 7** we see the result achieved in the HSV domain. Comparing it with what we had achieved in the previous section, we can say that the performances are way better. Nevertheless, there are still small imperfections that we tried to solve in the next chapters.

Figure 7: output in HSV domain

4 HSV with trackbars

4.1 Interactive trackbars for Saturation and Brightness thresholds

To further improve the results we thought of adding a way to dynamically adjust the threshold values. The user can now simply use his/hers mouse and empirically find the value that provides him/her with the best possible foreground mask. In order to obtain this, we added 2 trackbars in the output: one to control the saturation threshold and the other to control the value threshold. In the following images we can see the evolution over time of the masks by interactively changing the threshold values. **Figure 8** contains the evolution of the saturation mask, **figure 9** contains the evolution of the brightness one and **figure 10** contains the final foreground mask after the typical filtering procedures illustrated in the previous sections (median and morphological operations).

(a) (b) (c) (d) (e) (f)

Figure 8: Saturation binary mask obtained by interactively modifying the threshold value

Figure 9: Brightness binary mask obtained by interactively modifying the threshold value

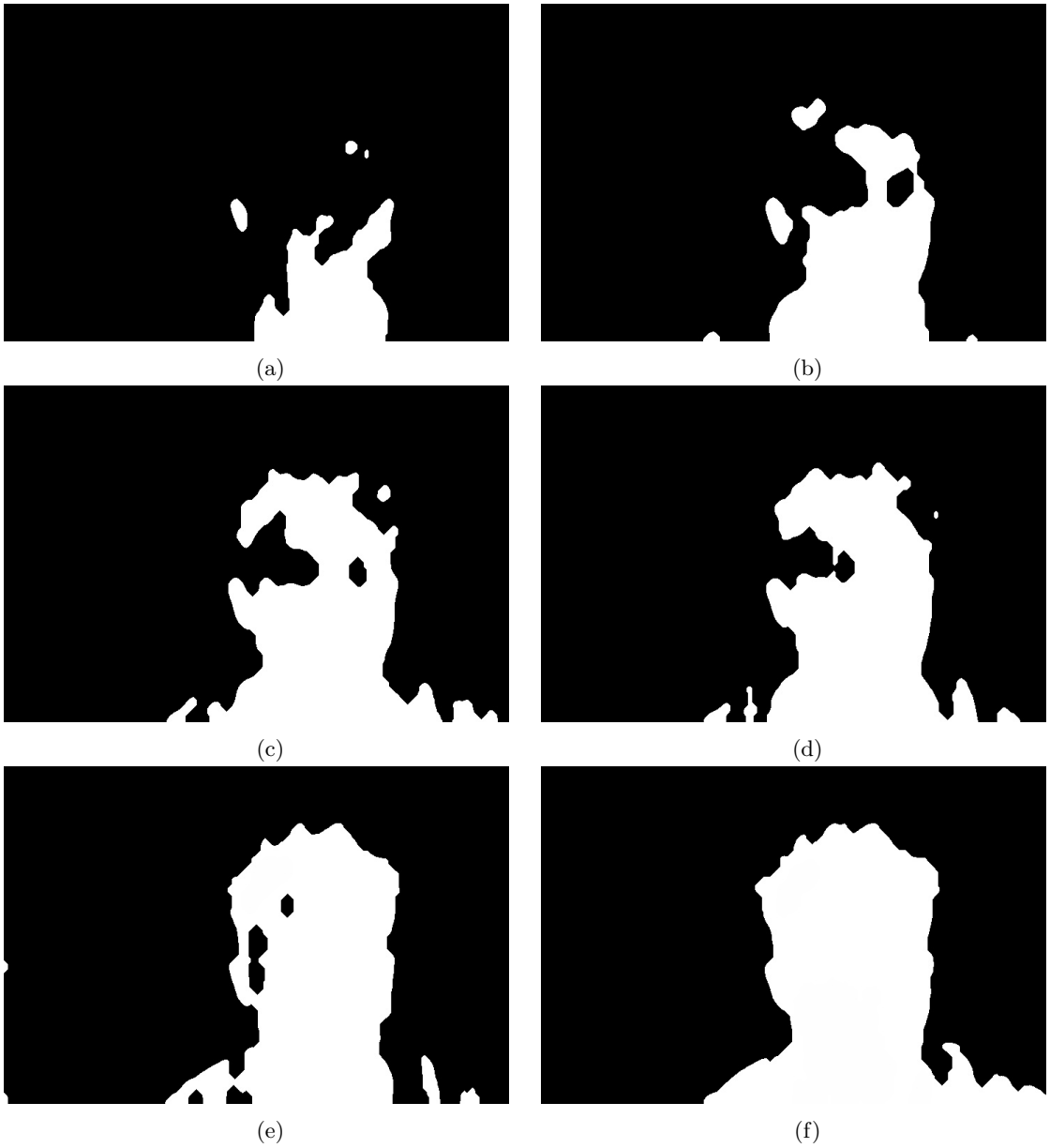
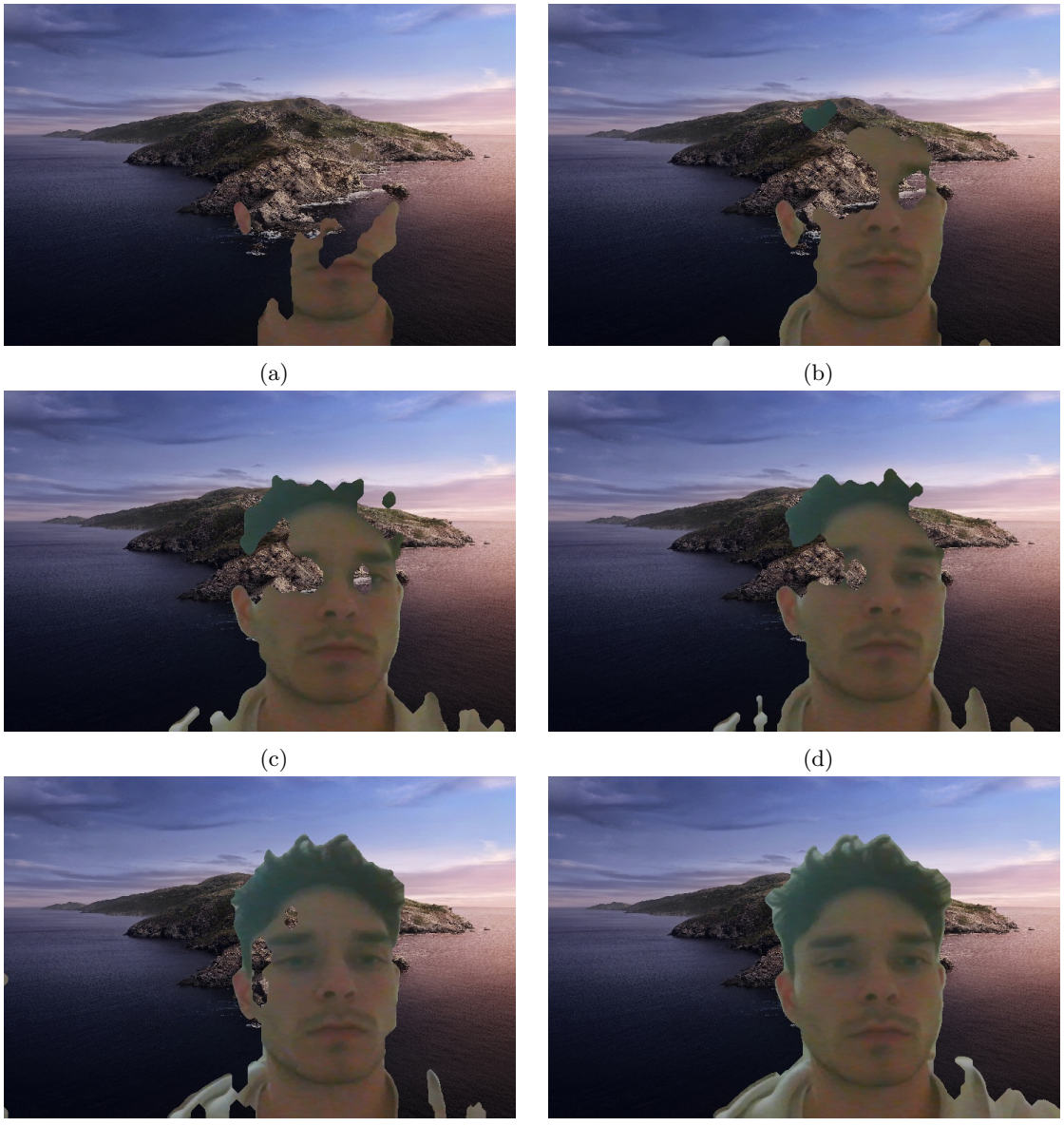


Figure 10: Combined binary mask obtained by interactively modifying the threshold value

4.2 Final output

After replacing our foreground mask with the scene captured by our cam and the background with another image, we can appreciate the final result in **figure 11**. By interactively choosing the best possible value of our thresholds we can achieve *almost* perfect results.



(a) (b) (c) (d) (e) (f)

Figure 11: Final output obtained by interactively modifying both the threshold values

5 HSV with optical flow

5.1 Lucas-Kanade algorithm

The last implementation consists of integrating the reasoning about the HSV colorspace and the idea behind the Optical Flow equations and theory. The main idea is to provide users with a way to save the foreground mask whenever they find it satisfactory enough. Therefore, to keep things intuitive, we thought of implementing this behaviour allowing users to press the 's' key, which reminds of the word 'save': when it is pressed, the current points of the binary mask start being tracked, frame after frame, making use of **the OpenCV implementation of the Lucas-Kanade method**.

A drawback, though, is that sometimes the algorithm loses track of too many points, resulting in an iterative shrinkage of the original mask. To bring things back to the original state, we implemented a functionality similar to the previous one, allowing users to press the 'r' key to 'reset' the background model and restart the algorithm without interrupting the runtime execution.

5.2 Convex hull algorithm

As mentioned before, the Lucas-Kanade method may happen to lose track of some foreground points over time. To tackle this, we integrated the OpenCV ConvexHull algorithm in the main script workflow, applying it to the set of points that are still tracked. The ConvexHull algorithm allows to extract the smallest convex polygon wrapping every point in the set it is given, providing a sharper though way more stable and coherent mask. Some examples of the output masks coming from the ConvexHull algorithm are shown in the images below.

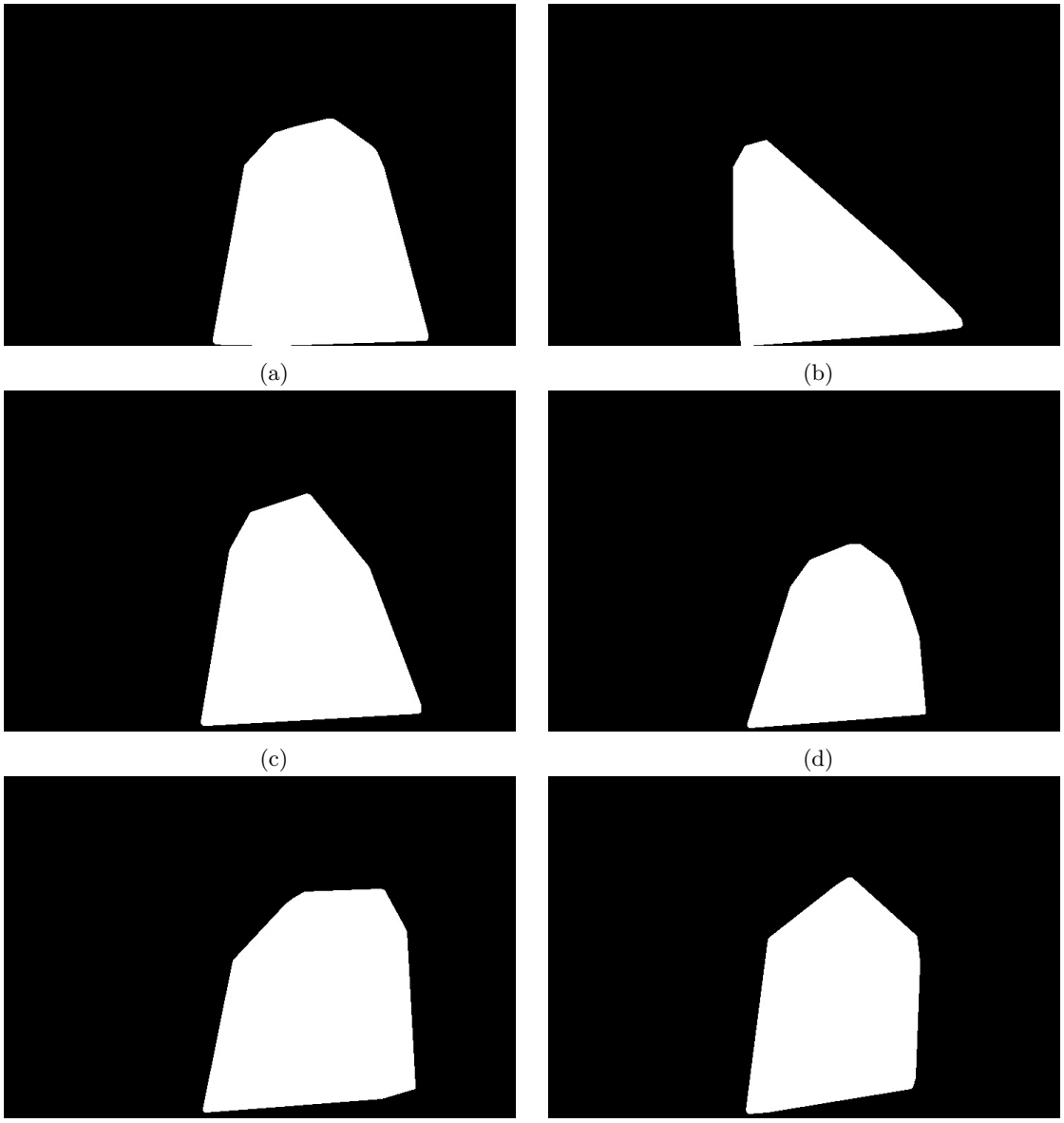
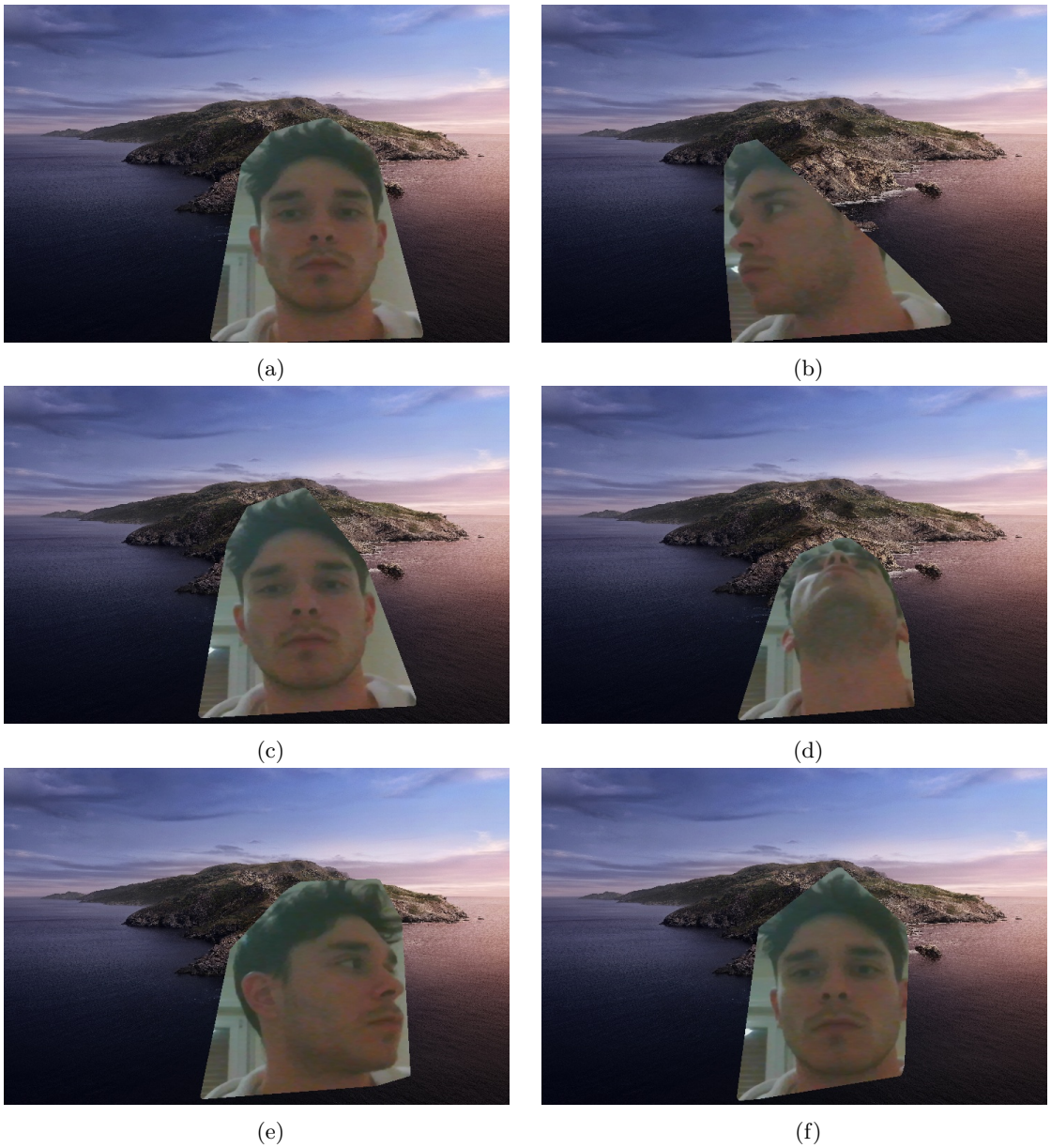


Figure 12: Binary masks obtained with optical flow algorithm after the convex-hull computation

5.3 Final output

As shown in the images below, the output of the latter algorithm is definitely more robust to noise and, above all, to changes in the illumination conditions. Additionally, an important concept such as spatial correlation between pixels is now taken into account. However, this comes at the cost of having a sharper mask, so with this configuration it is up to the end users to decide what is more desirable.



(a) (b) (c) (d) (e) (f)

Figure 13: Final output obtained with optical flow algorithm

6 Conclusions

Summing everything up, it has been shown that achieving nice and clean results is not trivial, but it can be done. The latter algorithms are robust to heterogeneous background settings, and have also found, by means of a smart application of morphological operators and/or optical flow equations, effective ways to deal with one of the most common and tricky problems in the image processing community, which is **illumination**.
However, it is also worth mentioning that, although it is satisfactory, the algorithm(s) is/are easily outperformed by the current state-of-the-art represented by Instance Segmentation with Deep Learning techniques. In fact, as opposed to them, our algorithm:

- requires "user cooperation", asking to stick out of the scene for an initial time interval;
- may include more noisy regions in the output frames.

Nevertheless, in scenarios where a trade-off between the usage of computational resources and quality results is needed, the proposed procedures are very good, effective and reliable choices.