Image Colorization

Marcus Tan

May 4, 2021

This project implements the image colorization technique developed by Levin et al. ("Colorization using optimization", SIGGRAPH '04). It is motivated by the potential of enhancing grayscale images with colors and recoloring a colored image with desired colors. Coloring the entire image manually is extremely tedious because the object boundaries need to be respected and the color shading needs to follow the original image intensity. The method implemented here greatly simplifies image colorization and satisfies the two aforementioned conditions with minimal work.

Let the image be represented by a color space consisting of an intensity channel Y and two chrominance channels U. Let U_i and P_i be the unknown and known chrominance values of pixel i whereP_i is obtained from the color marks/scribbles provided by the user. U_i is then the solution of the following linear equations:

$$U_i = \sum_{j \in N(i)} w_{ij} U_j \quad \text{if pixel i is colored,} \tag{1}$$

$$U_i = P_i \quad \text{otherwise,} \tag{2}$$

for $i = 1, 2, ..., n$. $N(i)$ is a set containing the neighbors of pixel i, n is the total number of pixels. The weights w_{ij} are given by

$$w_{ij} \propto \exp\left[-(Y_i - Y_j)^2/(2\sigma_Y)\right], \tag{3}$$

$$\sum_j w_{ij} = 1, \tag{4}$$

$$\tag{5}$$

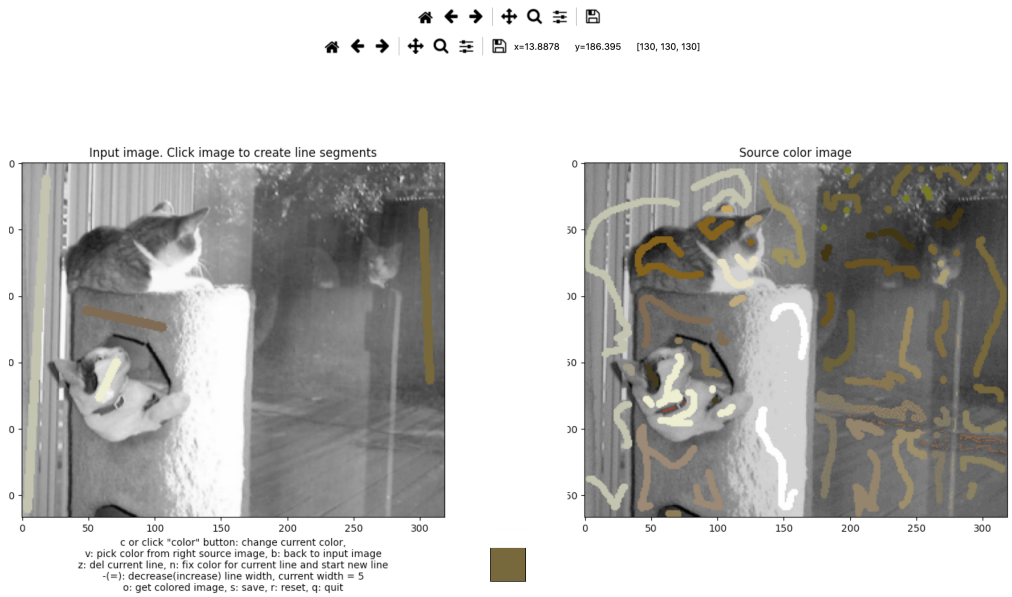
and σ_Y is the variance of the intensities for pixel i and its neighbors. These equations are set up in Python and solved using the Scipy exact linear sparse solver `spsolve`. The iterative linear solver can also be used.The OpenCV library is used to handle the images.

A GUI has been implemented using the Python Tkinter library that allows the user to draw colors on the input image at desired locations. The color can be obtained from two sources: (i) a color dialog box available on the OS (this GUI is only tested on Mac OSX Catalina) and (ii) colors from another source image. Fig. 1a shows the GUI and the color source image from the marked image provided by Levin et al (Fig. 1b). We use colors from the source image to draw colors on the grayscale version of the result image given by Levin et al (Fig. 1c). Our marked image and the result obtained from solving the above mentioned linear equations are shown in Fig. 1d and e, respectively. Note that our result is similar to that given by the paper, thus verifying the source code. A new example is provided in Fig. 2, where a vintage photo (from `https://izismile.com/2013/08/31/vintage_black_and_white_photos_that_capture_a_47_pics.html`) is colored using the OS color dialog box. Notice that the algorithm is able to color the objects in the image with minimal color bleed-out while ensuring the colors respect the original image intensities. The color bleed-out (for example on the left of the man on the left) can be eliminated by drawing the weaker color near the bleed-out location.

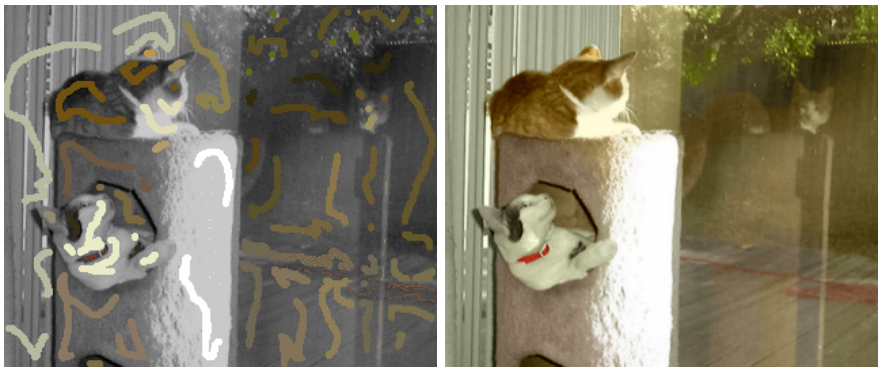
One of the challenges of implementing the GUI with Tkinter is the existence of bugs in `FigureCanvasTkAgg`, which embeds Matplotlib figures in the GUI and the difficulty in controlling object locations in the canvas. The GUI slows down significantly as more and more lines are drawn on the input image. More work is needed to improve the GUI.

Another challenge is the implementation of the conversion between YIQ and RGB spaces, which is not available in the OpenCV library. We encounter negative values when converting from YIQ to RGB space using the matrix

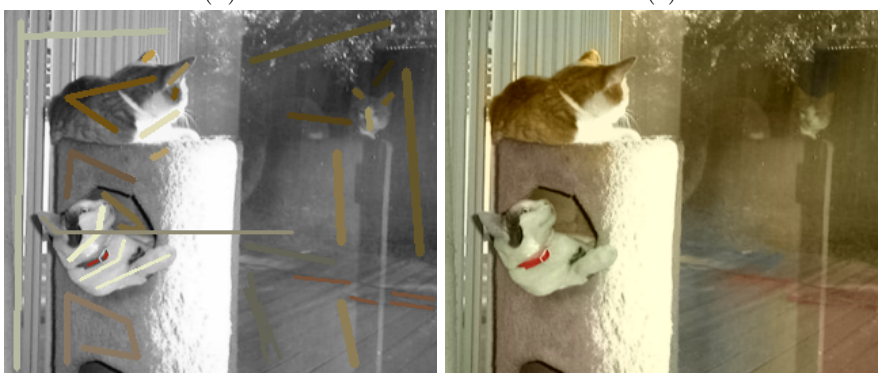
available online. MATLAB's `ntsc2rgb` does not have this issue but it is unclear how MATLAB achieves this. We eventually overcome this issue by using the OpenCV YUV color space.



(a)



(b) (c)



(d) (e)

Figure 1: (a) GUI showing the target grayscale image and the color source image, where the color can be extracted and drawn on the target image. (b) and (c): marked image and the resulting image from the paper by Levin et al. (d) Image marked by our GUI and (e) image colored by the optimization algorithm.

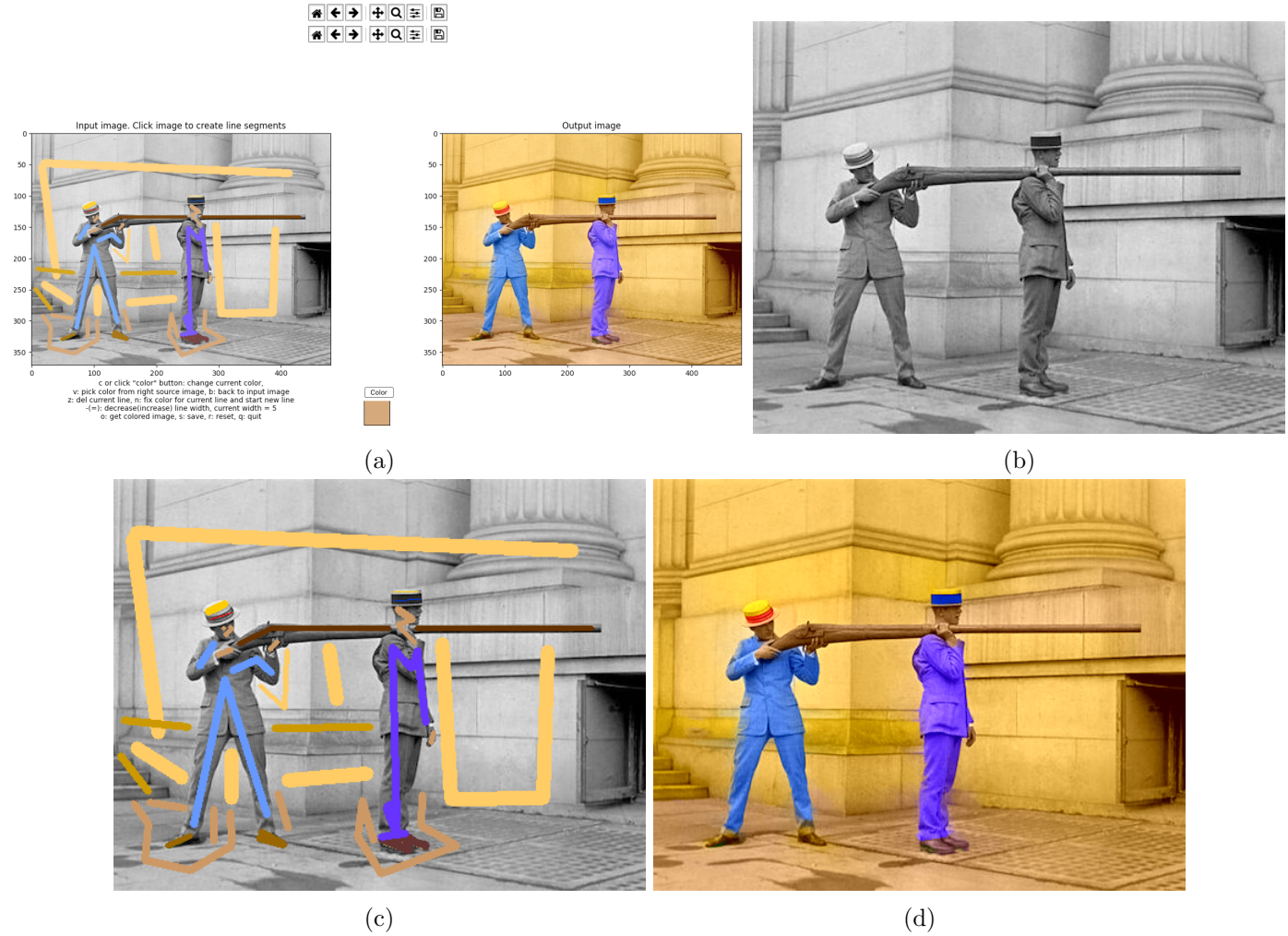


Figure 2: (a) GUI showing the target grayscale image with colors drawn on it and the output image. (b) Target grayscale image, (c) marked image and (d) image from the colorization algorithm.