Project in Advanced Algorithmics MTAT.03.238 2014 Fall

COLORIZATION USING OPTIMIZATION IN PYTHON

Gea Pajula[1], Kristjan-Julius Laak[1, *]

University of Tartu, Institute of Computer Science, *julius.laak@gmail.com

CONCLUSIONS

Colorization is the process of adding color to monochromatic images and footages. The aim of this project was to implement the colorization algorithm of the paper [1] in Python, and during the process, advance understanding of scientific computing with the open-source language. The authors of the project achieved the goals.

INTRODUCTION

Colorization is a term that describes any computer-aid method for adding color to black-and-white images and footages. The process generally involves large amount of user input to assist automatic image segmentation with complex boundaries (e.g. fuzzy hair). Thus, being expensive and time-consuming, most of the monochrome movies and TV-series have not been colorized.

The algorithm in the paper [1] presents a simple technique to automatically colorize both still images and moving sequences with a relatively small amount of user interference. Instead of manual segmentation and tracking of the regions, the method uses the idea that nearby colors in space and time have similar brightness levels.

Although the authors of [1] provide a MATLAB implementation of the method, in this project, a Python version of the algorithm is provided. The main purpose of this project is to examine the algorithm thoroughly and master skill in scientific computing with Python.

ALGORITHM

The algorithm works in YIQ color space where the $Y(p)$ component represents brightness, and the $I(p)$ and $Q(p)$ components represent the chrominance information of pixel p.

For each uncolored pixel p_0 of the scribbled image, variance of the intensities of it's window area are calculated by

$$\sigma^2 = \frac{1}{N}\sum_{i=0}^{N}(Y(p_i) - \frac{1}{N}\sum_{i=0}^{N}Y(p_i))^2,$$

where p_i, $i = 1, \dots, N$, are the neighbours of p_0. Then, for each p_i the similarity to the pixel p_0 is calculated using weighting function

$$w \propto e^{(Y(p_0)-Y(p_i))^2/2\sigma^2}.$$

Next, all the calculated weights are added to matrix A. Last, linear systems $A \times x_I = b_I$ and $A \times x_Q = b_Q$ are solved using the user colored pixels $I(p)$ and $Q(p)$ to finish up the colorization process.

RESULTS

Our Python implementation gives more weight to the p_0 window area pixels that have similar intensities with the pixel p_0. **Figure 1A** shows two grayscale images color-scribbled manually by the user. **Figure 1B** shows the color images obtained from the Python implementation of the algorithm of [1]. **Figure 1C** provides comparison by the original version of the same images.

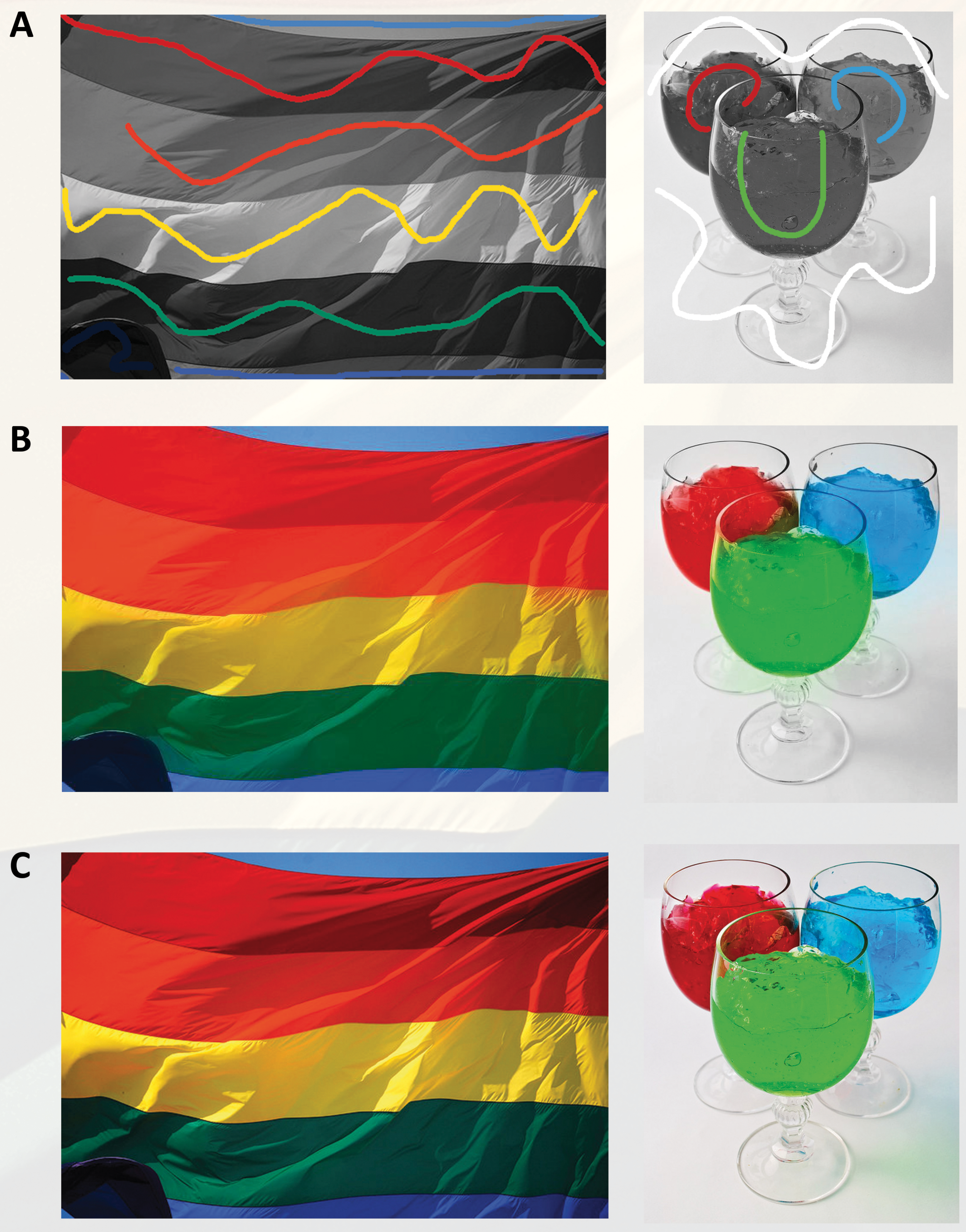

Figure 1 Colorization example. **A** Input monochrome image with user aided scribbles. **B** Resulting color image using the Python implementation of the algorithm. **C** Original image in comparison.

REFERENCES

[1] Anat Levin, Dani Lischinski, and Yair Weiss. 2004. Colorization using optimization. *ACM Trans. Graph.* 23, 3 (August 2004), 689-694.