Make The Old Pictures Alive! —

A Feature Matching Based Approach For Grayscale Image Colorization

Stephen Huang
Dachao Sun

Half-way Report of CPSC 863 Project
Instructor: Dr. James Z. Wang
School of Computing, Clemson University

March 31, 2015

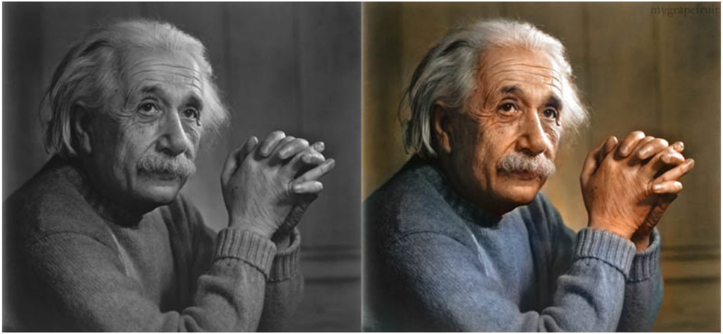
Image Colorization

- "Colorization": an intuitive action/process to add color on a pencil sketch or grayscale painting.
  – Generalization, as a term in digital image processing: **to convert a grayscale image to a color one** (intensity-only to RGB space).

- Goal and benchmark:
  "perceptually meaningful" and "visually appealing".

- Challenge: somewhat depens on the selection of reference image(s); the problem has no "visually correct" answer.

Examples



Albert Einstein

Quoted from "*15 Famous Photos in History Colorized*"
(`http://twistedsifter.com/2012/01/famous-photos-in-history-colorized/`)

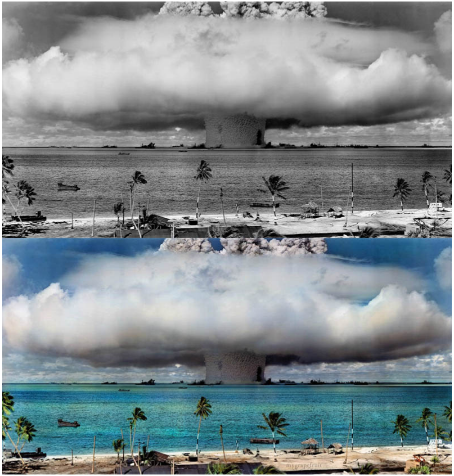
Examples

Joan Blondell
(August 30, 1906 – December 25, 1979)

Examples

Sophia Loren, Italian-French film star.

Examples



Mushroom-shaped cloud and water column from the underwater Baker nuclear explosion of July 25, 1946 ("Operation Crossroads").

Examples

Abandoned Boy After London Bombing WWII, by Toni Frissell.

Potential Application

Could be used for **efficient image storage** of color image:

Store only the intensity (single channel) value of color images, with proper categorization and corresponding reference images. Retrieve them back later to color images — kind of **compression**.

(can be concluded as part of the experiments of this project.)

Interactive v.s. Automatic

Two categories of colorization methods:

1. Interactive: color is inserted manually by users into a grayscale image, as a drawing process.
2. Automatic: color is taken from a **reference image** and transfered to the **target image**. Much more convenient but adjustments of parameters are often needed.

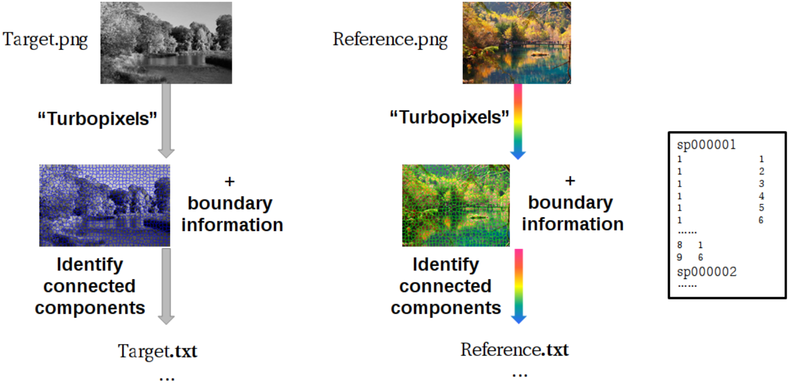
The Paper

The proposed by scholars Raj Kumar Gupta etc. in "*Image Colorization Using Similar Images*" in 2012 is an **automatic** approach, which includes four phases:

1. Do segmentation on both the reference and target images, using a method called "*Turbopixels*".
2. For each image segment, compute a 172-D feature that concludes: intensity (2), standard deviation (2), Gabor (40) and SURF (128).
3. Colorize each superpixel (segment) of the target image, by finding the "most similar" superpixel in the reference image.
4. Refinement: reassign the colors using a voting scheme.

Phase 1: Identify Superpixels

A preparation step, spliting both target and reference images into many "sub-images", which are colorized separately later.



"Turbopixels": A Fast Segmentation Approach

Proposed in 2009 by Alex Levinshtein, a PhD student at UofT, Canada.

"a geometric-flow based algorithm for computing **a dense over-segmentation of an image**, often referred to as **superpixels**."

1

TurboPixels: Fast Superpixels Using Geometric Flows

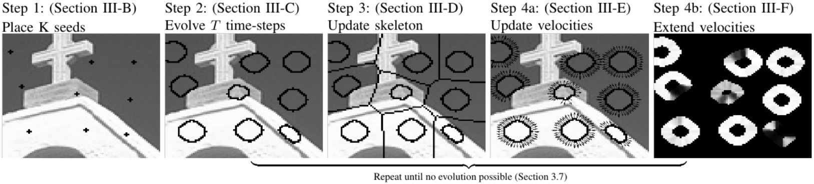
Alex Levinshtein, Adrian Stere, Kiriakos N. Kutulakos, David J. Fleet, Sven J. Dickinson
University of Toronto
Toronto, Canada
babalex,adrianst,kyros,fleet,sven@cs.toronto.edu

Kaleem Siddiqi
McGill University
Montreal, Canada
siddiqi@cim.mcgill.ca

***Abstract*—We describe a geometric-flow based algorithm for computing a dense over-segmentation of an image, often referred to as superpixels. It produces segments that on one hand respect local image boundaries, while on the other hand limit under-segmentation through a compactness constraint. It is very fast, with complexity that is approximately linear in image size, and can be applied to megapixel sized images with high superpixel densities in a matter of minutes. We show qualitative**

small, compact, quasi-uniform regions. Graph cut segmentation algorithms operate on graphs whose nodes are pixel values and whose edges represent affinities between pixel pairs. They seek a set of recursive bi-partitions that globally minimize a cost function based on the nodes in a segment and/or the edges between segments. Wu and Leahy [26] were the first to segment images using graph cuts, minimizing the sum of the edge weights

"Turbopixels": A Fast Segmentation Approach



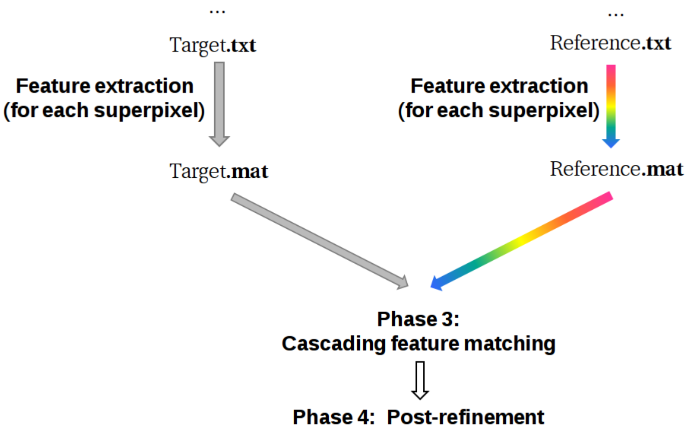
Original paper and reference code available on Dr. Levinshtein's website
`http://www.cs.toronto.edu/~babalex/research.html`

```
[phi,boundary,disp_img] = superpixels(img,
                                       (num_pixels/40)/1.5,
                                       0,
                                       [0,0,1]);
```

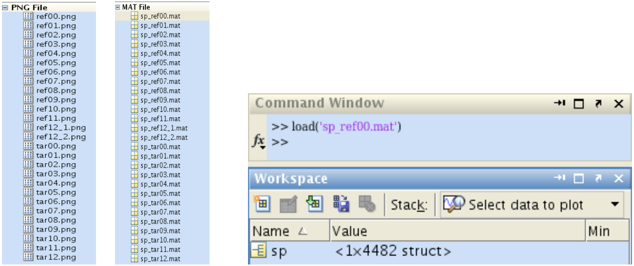
"Turbopixels": A Fast Segmentation Approach

$\approx$ 40 pixels per superpixel
boundaries and contours are well detected.

Phase 2: Feature Extraction



.mat for each image (target and reference)




```
>> sp(1000)
ans =
   coordinates: [1x71 struct] % with fields 'row' and 'col'
   row_min: 96
   row_max: 108
   col_min: 435
   col_max: 445
   neighbors: [863 878 901 964 982 999 1059 1067 1109 1132 1133]
   feature: [1x172 double]
```

Intensity	Std	Gabor	SURF
2	2	40	128

- Intensity (2)

$$f_1(i) = \frac{1}{n} \sum_{(x,y) \in S_i} I(x, y)$$

$$f_2(i) = \frac{1}{N} \sum_{\substack{j \in Neighboring \\ SuperpixelsofS_i}} f_1(j)$$

- Standard deviation (2)

$$g_1(i) = \frac{1}{n} \sum_{(x,y) \in S_i} Std\ (x, y)$$

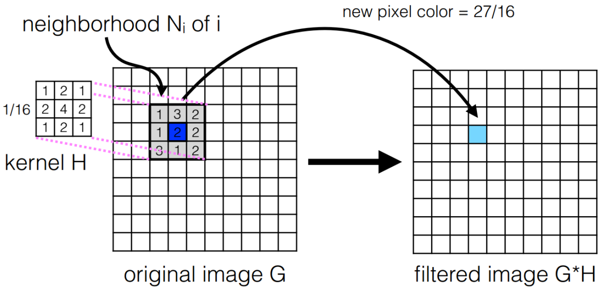
$$g_2(i) = \frac{1}{N} \sum_{\substack{j \in Neighboring \\ SuperpixelsofS_i}} g_1(j)$$

- Gabor ($40 = 5 \times 8$)

8 "orientation"s, $\theta = \frac{n\pi}{8}$ $(n = 0..7)$;

5 "exponential scale"s, $e^{i\pi}$ $(i = 0..4)$.

Each pair of the above defines a 5×5 square Gabor filter, which is applied to the whole image to get a "Gabor value" of each pixel.



(Quoted from Dr. Levine's CPSC 604 slides)

- Gabor: Def. of filter kernel weights #1

$$g(x, y;\ \theta, \sigma, T) = e^{-\frac{\hat{x}^2 + \hat{y}^2}{2\sigma^2}} \cos(\frac{2\pi\hat{x}}{T})$$

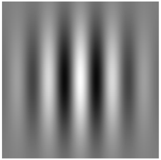
where

$$\hat{x} = x \cos\theta + y \sin\theta$$

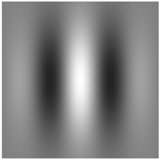
and

$$\hat{y} = -x \sin\theta + y \cos\theta$$

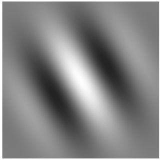
(x, y) are distances measured from the kernel center, θ is an angular orientation, σ is the standard deviation of the Gaussian curve, and T is the period of the cosine.



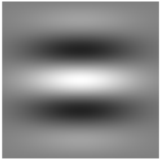
$(0, 50, 50)$



$(0, 50, 100)$



$(30, 50, 100)$



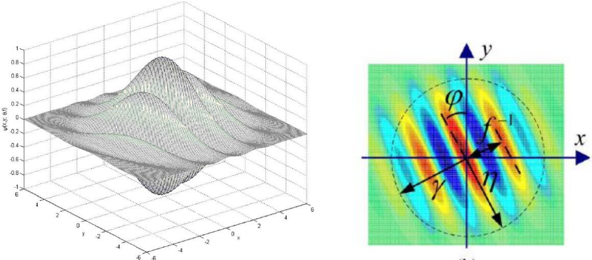
$(90, 50, 100)$

- Gabor: Def. of filter kernel weights #2

$$\psi(x, y; \varphi, f) = \frac{f^2}{\pi \gamma \eta} e^{-\left(\frac{f^2}{\gamma^2}\hat{x}^2 + \frac{f^2}{\eta^2}\hat{y}^2\right)} e^{2\pi j f \hat{x}}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}.$$

f: frequency of the sinusoidal plane wave; φ: counterclockwise rotation angle; γ: width of filter, parallel with the plane wave; η: width of filter, perpendicular to the plane wave.



(a) (b)

- Extended SURF (128)

Speeded **U**p **R**obust **F**eatures is a robust local feature detector, first presented by Herbert Bay et al. in May 2006. The standard version of SURF runs severl times faster than SIFT, another famous local feature detection algorithm.

Image registration, camera calibration, object recognition, image retrieval...

Speeded-Up Robust Features (SURF)

Herbert Bay [a], Andreas Ess [a], Tinne Tuytelaars [b], and Luc Van Gool [a,b]

[a] *ETH Zurich, BIWI*
Sternwartstrasse 7
CH-8092 Zurich
Switzerland
[b] *K. U. Leuven, ESAT-PSI*
Kasteelpark Arenberg 10
B-3001 Leuven
Belgium

Abstract

This article presents a novel scale- and rotation-invariant detector and descriptor, coined SURF (Speeded-Up Robust Features). SURF approximates or even outperforms previously proposed schemes with respect to repeatability, distinctiveness, and robustness, yet can be computed and compared much faster.

This is achieved by relying on integral images for image convolutions; by building on the strengths of the leading existing detectors and descriptors (specifically, using a Hessian matrix-based measure for the detector, and a distribution-based descriptor); and by simplifying these methods to the essential. This leads to a combination of novel detection, description, and matching steps.

The paper encompasses a detailed description of the detector and descriptor and then explores the effect of the most important parameters. We conclude the article with SURF's application to two challenging, yet converse goals: camera calibration as a special case of image registration, and object recognition. Our experiments underline SURF's usefulness in a broad range of topics in computer vision.

Key words: interest points, local features, feature description, camera calibration, object recognition
PACS:

1. Introduction

The task of finding point correspondences between two images of the same scene or object is part of many computer vision applications. Image registration, camera calibration, object recognition, and image retrieval are just a few.

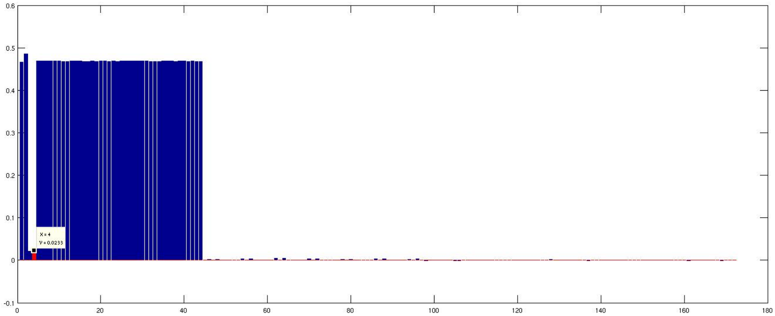
The search for discrete image point correspondences can be divided into three main steps. First, 'interest points' are selected at distinctive locations in the image, such as corners, blobs, and T-junctions. The most valuable property of an interest point *detector* is its repeatability. The repeatability expresses the reliability of a detector for finding the same physical interest points under different viewing conditions. Next, the neighbourhood of every interest

between the vectors, e.g. the Mahalanobis or Euclidean distance. The dimension of the descriptor has a direct impact on the time this takes, and less dimensions are desirable for fast interest point matching. However, lower dimensional feature vectors are in general less distinctive than their high-dimensional counterparts.

It has been our goal to develop both a detector and descriptor that, in comparison to the state-of-the-art, are fast to compute while not sacrificing performance. In order to succeed, one has to strike a balance between the above requirements like simplifying the detection scheme while keeping it accurate, and reducing the descriptor's size while keeping it sufficiently distinctive.

A wide variety of detectors and descriptors have already

172-dimensional feature



Phase 3: Cascade Feature Matching (pruning for similarity)

(3+ slides to be added here...)

Phase 4: Reassigning Colors: Image Sapce Voting

(1 slide to be added here...)

Evaluation: how "real" / "natural" is the result?

Naturalness: how well the artificial colorized image "mimics" the color of similar content in the real-world scene.

Such intuition may come to two ways of evaluation: **ground-truth error comparison** and **user study**.

Considering from an aesthetic perspective (not to be contrained by "conrrect answers"), we will only do the latter.

User Study

- One of a dozen of invited users is ready;
- A group of 4-6 color images are shown on the screen, half (but not always) of which are artificially-colorized by the approach in this project, while others are original color images.
- Let the user point out which one(s) of them is/are artificial.
- $Avg_{\texttt{users}} \frac{\#\ of\ \ correct\ \ selections}{\#\ of\ \ images\ \ in\ the\ \ group}$ denotes the naturalness of this group of color images.

An example: one out of the four is artificially colorized.

Further Experiments

1. Would the order of cascading (Gabor→SURF→etc.) affect the result significantly?
2. Which of the four features dominates? Any other better option(s) for the features **(performance vs. computational complexity)**?
3. Color image compression pipeline, based on the colorization approach.

"Milestone": Time Management

Milestone I.	Jan. 29	complete the proposal (concepts, tools, test data, etc.).
	Feb. 15	identify superpixels (phase 1), saved in `.txt` format
	Mar. 12	code feature extraction (phase 2), done with `.mat` format.
Milestone II.	Mar. 24	start with feature matching (phase 3).
	Mar. 31	half-way presentation.
	April 10	get preliminary results, fix problem(s).
	April 15	refinement and experiment.
Milestone III.	by **April 20**	finish writing the report (20–40 pages) and presentation slides.

"Milestone": Time Management

Milestone I.	Jan. 29	complete the proposal (concepts, tools, test data, etc.).
	Feb. 15	identify superpixels (phase 1), saved in `.txt` format
	Mar. 12	code feature extraction (phase 2), done with `.mat` format.
Milestone II.	Mar. 24	start with feature matching (phase 3).
	Mar. 31	half-way presentation.
	April 10	get preliminary results, fix problem(s).
	April 15	refinement and experiment.
Milestone III.	by **April 20**	finish writing the report (20–40 pages) and presentation slides.

“Milestone”: Time Management

Milestone I.	Jan. 29	complete the proposal (concepts, tools, test data, etc.).
	Feb. 15	identify superpixels (phase 1), saved in `.txt` format
	Mar. 12	code feature extraction (phase 2), done with `.mat` format.
Milestone II.	Mar. 24	start with feature matching (phase 3).
	Mar. 31	half-way presentation.
	April 10	**get preliminary results**, fix problem(s).
	April 15	refinement and experiment.
Milestone III.	by **April 20**	finish writing the report (20–40 pages) and presentation slides.