Colorization Using Optimization

Chris Gonterman

This project explored the techniques described in Colorization Using Optimization, the paper by Levin et al.[1] The algorithm proceeds by casting the problem as an optimization problem: pixels of similar intensities are likely to have similar colors, so deviations from that model should be minimized. This formulation lets us bring the general purpose tools for solving optimization problems to bear.

Algorithm

In order to proceed with developing the algorithm, we need to separate the pixel values into intensities and colors, and define what it means for them to be similar. Intensity is straightforward, as it maps naturally onto the real number line. Color can be dealt with by transforming the pixel value to a color space that separates color and intensity, such as YUV or HSV. Then, color similarity can be defined by the distance in the channels that represent color. For consistency, I'll use the YUV color space here, where Y corresponds to the intensity, and U,V together describe the color.

Using the intuition that similar pixels in the gray image should have similar colors, we can introduce the following distance function we wish to optimize.

$$J(U) = \sum_{\boldsymbol{r}} \left(U(\boldsymbol{r}) - \sum_{\boldsymbol{s} \in N(\boldsymbol{r})} w_{\boldsymbol{rs}} U(\boldsymbol{s}) \right)^2$$

[1] *A. Levin, D. Lischinski, Y. Weiss*, **Colorization Using Optimization**, ACM SIGGRAPH 2004

Here, U represents a color channel, **r** and **s** are both pixels, N(**r**) is the neighborhood of **r**, and lastly $w_{\boldsymbol{rs}}$ is a weight that captures the intensity similarity. Intuitively, this states that we want to minimize the square distance between each pixel's color and a weighted average of its surrounding pixels.

The weight $w_{\boldsymbol{rs}}$ can be derived by considering a particular model for the relationship between color and intensity. Assuming a local linear relationship between color and intensity yields the following weighting scheme:

$$w_{\boldsymbol{rs}} \propto 1 + \frac{1}{\sigma_{\boldsymbol{r}}^2}(Y(\boldsymbol{r}) - \mu_{\boldsymbol{r}})(Y(\boldsymbol{s}) - \mu_{\boldsymbol{r}})$$

In natural language, the weight for a given pixel is proportional to correlation between the intensities, normalized since these are used to perform an average. This requires computing the mean and variance for the neighborhood of every pixel.

Note that we can write $J(U)$ in matrix form as $J(U) = U^T(I - W)^T(I - W)U$, where I is the identity matrix and W is just the matrix form of the weights, i.e. $W_{rs} = w_{\boldsymbol{rs}}$. This is of the form $x^T Hx$, which tells us that this is a quadratic programming problem.

Note: Levin et al state that $J(x) = x^T(D - W)x$, where $x \sim U$ and $D \sim I$, but one can see that this is not quite correct. They also point out that minimizing $J(x)$ corresponds to minimizing $x^T Ax$ for a symmetric matrix A, but as $D - W$ is defined, it is not symmetric. The matrix I used above, however, is symmetric.

We can now let the user constrain the solution space by drawing swathes of color, thus pinning those pixels to the drawn color. In the quadratic programming problem we set up with our function $J(U)$, this corresponds to adding linear constraints. There are a variety of solvers for problems of this type.

Implementation

To implement this, I approached this with both C++ and Matlab. Finding solvers for C++ proved to be too unwieldy, so I focused my efforts on Matlab's quadprog routine for solving the general quadratic programming problem. One key consideration in solving the problem as specified is that a naïve implementation of the matrices would be beyond the memory capacity of any machine very quickly. As such, sparse representations, and algorithms that run on those representations, must be used. This turned out to be a limiting factor for some avenues.

My intent was to explore how various color spaces affected the resulting images. Matlab implements conversion for RGB, HSV, and YIQ (comparable to YUV). I began with YIQ due to its similarity to YUV. Some results for that are below. For HSV, the optimization routine tends to pick non-realizable solutions, such as negative hues and saturations. In order to fix this, one would need to add in inequality constraints to the quadratic programming formulation, which increases the complexity. Unfortunately, solutions to quadratic programming problems with the additional constraints aren't available for sparse problems, which is an essential factor to making the algorithm run reasonably. Unfortunately, I had to stick to implementing YIQ.

To make the routine run quickly, I did my tests on the following low resolution image:

There are some regions where there's a clear edge in the color image, but the corresponding region in the gray image is difficult to distinguish, such as the left and right edges of the t-shirt.

I used the following color strokes to pin down colors, and the image on the right is the result.

Future Directions

The formulation of the function to be optimized considers the color channels separately, which is the key reason why I wanted to explore using other color spaces that may provide better perceptual encapsulation of each channel. It would be even better if we could directly model the perceptual difference in the color in the function itself, rather than separating it into two channels and hoping that the resulting function maintains color consistency. In addition, there are better solvers that could be applied to the problem, resulting in more stable solutions that don't exhibit the unrealizable "colors," as well as being more efficient.