CS280 - Computer Vision - Spring 2017 - Assignment 2
Due: Friday, Mar 8, 2019, 11:55pm

1 Hybrid Images

Here, we will create hybrid images using the approach described in the SIGGRAPH 2006 paper by Oliva, Torralba, and Schyns [3]. Hybrid images are static images that change in interpretation as a function of the viewing distance. The basic idea is that high frequency tends to dominate perception when it is available, but, at a distance, only the low frequency (smooth) part of the signal can be seen. By blending the high frequency portion of one image with the low-frequency portion of another, you get a hybrid image that leads to different interpretations at different distances. Some examples are shown in this gallery: `http://cvcl.mit.edu/hybrid_gallery/gallery.html`. You can also see a fun example on Stephen Fry's show: `https://www.youtube.com/watch?v=OlumoQ05gS8`.

We have included two sample images of your two favorite professors, `malik.png`, `papadimitriou.png`, as well as some starter code in `Matlab` and `Python` to load and align images. The alignment is important, as it affects the perceptual grouping (read the paper [3] for details). Chose a few pairs of images that you want to make into hybrid images as well. Try creating a variety of types of hybrid images (change of expression, morph between different objects, change over time, etc.).

1. Write a function `img_hybrid = hybrid_image(img1, img2)`. The function should:
   (a) Low-pass filter `img1`. Oliva, et al [3] suggest using a standard 2D Gaussian filter, but feel free to try your own favorite filter. The cutoff can be chosen with some experimentation.
   (b) High-pass filter `img2`. Oliva, et al [3] suggest using the impulse filter minus the Gaussian filter, which can be computed by subtracting the Gaussian-filtered image from the original image. The cutoff can be chosen with some experimentation and does not need to be the same as for the low-pass filter.
   (c) Linearly combine the two images.
2. In your report, include the following:
   (a) Briefly describe how this algorithm works. Write out the parametric form of the low-pass and high-pass filters you used, any parameters you chose, and include plots of the filters spatially.
   (b) Show your favorite result and illustrate the process through frequency analysis. Show the log magnitude of the Fourier transform of the two input images, the filtered images, and the hybrid image. You can compute and display the 2D Fourier transform with `imagesc(log(abs(fftshift(fft2(gray_image)))))` in `MATLAB` and `plt.imshow(np.log(np.abs(np.fft.fftshift(np.fft.fft2(gray_image)))))` in `Python`.
   (c) Show at least two more results, including one that does not work so well. Display your resulting hybrid images at various scales in order to simulate the viewer being closer or further away from the hybrid image. Briefly explain how you got the good results (e.g., chosen cut-off frequencies, alignment tricks, other techniques), as well as any difficulties and the possible reasons for the bad results. If you are so fortunate that everything that you try works well, try to figure out what shouldn't work.

2 Edge Detection

In this problem, build an elementary edge detector on some sample images from the Berkeley Segmentation Dataset [2]. We have provided a few images from the BSDS. You can find the human annotations by accessing the site `/www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/BSDS300/html/dataset/images/color/###.html`, where `###` is the image number. We will qualitatively compare the results to a 2014 state-of-the-art algorithm [1] as well. Also try using a few images from your personal collection.

1. **Finite operator** We will begin by using the humble finite difference as our filter in the x and y directions.

$$\mathbf{D_x} = \begin{bmatrix} 1 & -1 \end{bmatrix}, \mathbf{D_y} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \tag{1}$$

   Write function `[mag, theta] = difference_filter(img)`. Convolve the finite difference operator with the images, $\mathbf{D_x} * \mathbf{I}$ and $\mathbf{D_y} * \mathbf{I}$. Show the oriented gradient responses. Now compute the gradient magnitude at each location by taking the L2-norm over the R,G,&B channels in both the x and y directions and plot. Compute the gradient orientation at each pixel from the channel corresponding to the largest gradient magnitude. Explain how you computed the gradient orientation from the x and y filter responses. Visualize the gradient orientation at each pixel. Notice that the resulting gradients are very noisy. Remove some edges by trying a few thresholds and visualize the remaining gradient magnitudes and orientations.

2. **Derivative of Gaussian** We noted that the results with just the difference operator were rather noisy. Luckily, we have a smoothing operator handy: the Gaussian filter $\mathbf{G}_\sigma$. Write function `[mag, theta] = derivative_gaussian_filter(img, sigma)`. Convolve the finite difference operators with the 2D Gaussian filter, $\mathbf{G}_\sigma * \mathbf{D_x}$ and $\mathbf{G}_\sigma * \mathbf{D_y}$, and plot the filters.

   Then, as in part 1, convolve the image with the oriented filters, find the magnitude and orientation, and try a few thresholds. Make the same visualizations as you did in part 1. Compare the new results with the results from part 1, and note any observations.

   Note that smoothing the gradient map is the same as computing the gradient on the smoothed image. In other words, $\mathbf{G}_\sigma * (\mathbf{D_x} * \mathbf{I}) = \mathbf{D_x} * (\mathbf{G}_\sigma * \mathbf{I})$. Why is this the case, and why may it be important?

3. **Oriented Filters** Now try to improve your results using a set of oriented filters, rather than the simple derivative of Gaussian approach above. Implement function `[mag,theta] = oriented_filter(img)`, which computes the boundary magnitude and orientation using a set of oriented filters of your choice, such as elongated Gaussian derivative filters. Use at least four orientations and explain your choice of filters. Find a way to combine the individual filter responses into a magnitude and orientation map. Visualize the results as before.

4. Qualitatively compare the resulting edge maps you obtain to the 2014 state-of-the-art result from Isola, et al. [1], along with human annotations. What do you notice about the human annotations? What does your best algorithm do well? What does it struggle with?

5. **Image Straightening**

   We will now use edge orientation to rectify a slightly rotated image in order to vertically align it. The basic idea is that in an aligned image the orientation of most of the edges should be either vertical or horizontal. Given an input image, find a rotation transformation which results in an image whose edge orientation histogram peaks at vertical and horizontal edges. More concretely, implement the following steps

   (a) Compute an orientation map, as previously. Plot a histogram of the angles and visualize it.

   (b) Use this histogram to identify a rotation that shifts the peaks of the histogram to horizontal and vertical edges.

   (c) Apply the rotation and verify the histograms.

   Find two **natural** images, one where this procedure succeeds and another where it fails to align the image. For both images, provide:

   (a) Comparison of the histograms of the input image and the aligned image.

   (b) Comparison of the input and the aligned images.

3 Instructions

1. This assignment is to be done individually.

2. Please submit the assignment using bCourses. Upload the following files:

(a) `A PDF file`. The top of the first page should contain your name and student ID. The file should contain answers to all questions, supporting images, and **code for all functions specified in Problem 1 and 2**.

(b) `A tar/zip file`. The file should contain a `pdf` of your report, along with a subfolder `code`, with subdirectories for Problem 1 and 2.

3. The HW is due on: Friday, Mar 8, 2019, 11:55pm. Per course policy, you have 5 emergency slip days to cumulatively use throughout the semester.

4 Acknowledgements

Parts of this assignment are borrowed from Derek Hoiem's Computational Photography class (`https://courses.engr.illinois.edu/cs498dh/fa2011/`). The Mini Places Challenge was "heavily inspired" by the MIT assignment (`http://6.869.csail.mit.edu/fa15/project.html`).

References

[1] Phillip Isola, Daniel Zoran, Dilip Krishnan, and Edward H Adelson. Crisp boundary detection using pointwise mutual information. In *European Conference on Computer Vision*, pages 799–814. Springer, 2014.

[2] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 416–423. IEEE, 2001.

[3] Aude Oliva, Antonio Torralba, and Philippe G Schyns. Hybrid images. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 527–532. ACM, 2006.