CSC487/2503—Foundations of Computer Vision, Fall 2008

Assignment 2: Image Pyramids and Robust Estimation

Due: 12:10pm, Mon., Oct. 27 (at the start of the tutorial)
This assignment is worth 15 marks towards your grade in this course.

The goals of this assignment are for you to gain some experience: 1) using Gaussian and Laplacian pyramids; and 2) using robust estimation to extract simple image models.

Download the handout code A2handout.zip from the homepage. These files will be used in the following two questions.

What to hand in. Write a short report addressing each of the itemized questions below (hand-written reports are fine). You can assume that the marker knows the context of the questions, so do not spend time repeating material in the hand-out, or in class notes. Also, email the completed Matlab files for each question to `mstefan@cs.utoronto.ca`. For the marker's convenience, please put the solution to the n^{th} question in subdirectory Qn.

---

1. **Laplacian Pyramids [10pts]:** Use the Laplacian pyramid to blend two images, using the algorithm described in the lecture notes (see imageTransform.pdf, page 19). That is,

   - Build Laplacian pyramids for the two images. Matlab code for Laplacian pyramids is available in `iseToolbox/pyrTools`. It's use is explained in `pyramidTutorial.m` and the handout code.
   - Given these two pyramids, follow the approach described in the lecture notes to generate a new, blended pyramid. Take care with the scaling of the values in the Gaussian pyramid. Note that the lecture notes assume that each level of the Gaussian pyramid of the image mask, namely $l_k(\vec{n})$, is in the range $[0, 1]$. However, the Gaussian pyramid generated by `buildgpyr` is scaled differently.
   - Reconstruct the image from the blended Laplacian pyramid.

   The apple and orange image from the lecture notes are provided as sample data. Generate a blended image similar to the one shown in the lecture notes. Explain how different values for the total number of pyramid levels (i.e., the variable `maxLevels` in the handout code) effect the results.

   Compare the above result with a simple blended average of the two images,

   $$I_b(\vec{x}) = (1 - b(\vec{x}))I_1(\vec{x}) + b(\vec{x})I_2(\vec{x}), \tag{1}$$

   where I_1 and I_2 are the apple and orange images, and $b(\vec{x})$ is the blurred mask `blurMask` produced in the handout code (with `maxLevels` set at 7). In your write up, clearly explain how $I_b(\vec{x})$ appears different from the result you obtain by blending the Laplacian pyramids. Explain the reason(s) for this difference.

2. **Robustly Fitting Circles [20pts].** A geneticist wishes to automatically count cells in microscope images, such as the one depicted in Fig. 1a, and take specific note of cells that are in various stages of splitting. Here we consider an application of robust estimation that will get us started on a solution for the geneticist.

   The handout code `findCellScript.m` reads in a microscope image of cells. We have already detected cell pixels (foreground), and have labelled connected components of these pixels. The resulting labels are read in as a second image. Canny edgels are computed on the foreground mask. The code then considers each connected segment separately.

   Your job is to fit circles to the edgels obtained from the boundaries of these segments. The idea is that each fitted circle should correspond to one "cell" including, perhaps, a small circle for a cell bud just

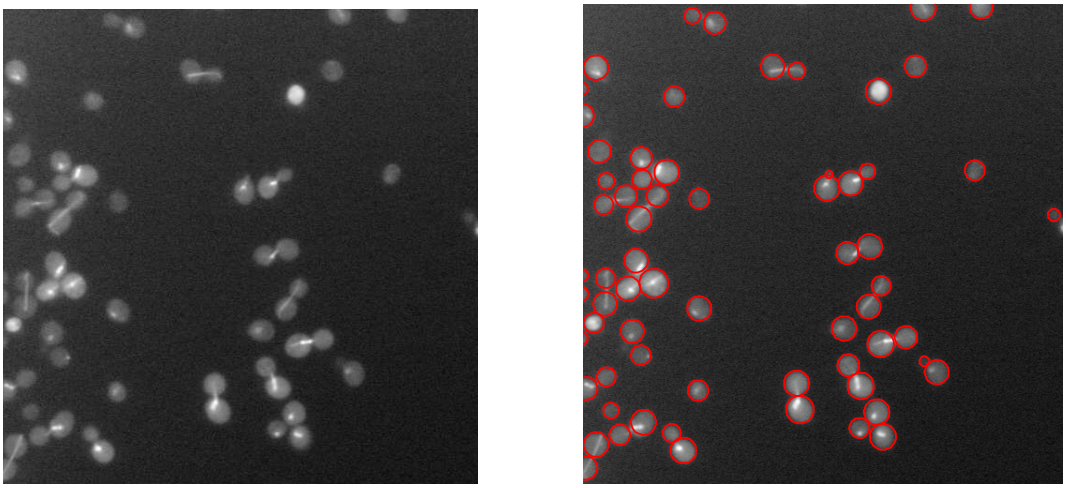

Figure 1: Cell image (a) with fitted cells (b) (view in colour).

being born by splitting off from another cell (see Fig. 1b). I realize that this is a vague definition of what constitutes a "cell". However, if you study Fig. 1b (perhaps by blowing it up in the electronic copy), I think it is intuitively clear what is meant by a "cell". Part of your job in this question is to decide on how to operationally define our intuitive notion of one cell.

In the following we denote a given edgel by $(\vec{x}_k, \vec{n}_k)$, where $\vec{x}_k$ is the image position and $\vec{n}_k$ is the edgel normal. The edgel normal points in the direction of increasing brightness of the foreground mask (i.e., towards the *inside* of the cells).

The general approach we use is based on robust estimation using the Geman-McLure (GM) estimator introduced in class,

$$\rho(e, \sigma_g) = \frac{e^2}{\sigma_g^2 + e^2}. \tag{2}$$

The error e here is the error in a given edgel, $(\vec{x}_k, \vec{n}_k)$ say, with respect to the circle centered at $\vec{x}_c$ having radius r. We can express this error as

$$e_k(\vec{x}_c, r) = \sqrt{[\vec{n}_c(\vec{x}_k) \cdot (\vec{x}_k - \vec{x}_c) - r]^2 + [1 + \vec{n}_c(\vec{x}_k) \cdot \vec{n}_k]\beta}. \tag{3}$$

Here

$$\vec{n}_c(\vec{x}_k) = (\vec{x}_k - \vec{x}_c)/||\vec{x}_k - \vec{x}_c||, \tag{4}$$

is a unit vector in the direction of the edgel position $\vec{x}_k$ from the circle center $\vec{x}_c$. The squared term in (3) measures the squared distance between the edgel position $\vec{x}_k$ and the circle specified by $(\vec{x}_c, r)$. The orientation term, namely $1 + \vec{n}_c(\vec{x}_k) \cdot \vec{n}_k$, involves the inner product of the edgel normal $\vec{n}_k$ with the circle's outward normal, $\vec{n}_c(\vec{x}_k)$. Since we are expecting the edgel normal to be pointing inwards, a perfect value for this inner product would be -1. For edgel normals differing from the radially inward direction by an angle θ (in radians), a Taylor series expansion of $\cos(\pi + \theta)$ shows that

$$1 + \vec{n}_c(\vec{x}_k) \cdot \vec{n}_k = 1 + \cos(\pi + \theta) = \theta^2/2 + O(\theta^4).$$

Thus, for small angular errors θ, this orientation error term is roughly proportional to the square of the angular error. Finally, the value $\beta > 0$ in (3) is a constant used to scale these orientation errors so that the average magnitude of the two error terms in (3) are roughly equal for typical inlier edgels.

A general solution strategy for fitting circles to this edgel data, using the above robust estimator, has been partially implemented in the handout code `findCellScript.m`. You only need to complete several M-functions, as described below. In general, you are allowed to make small changes to the handout code.

(a) **Circle proposals.** Your first job is to complete `getProposals.m` (in the `circleFit` subdirectory). When finished this function should return a $P \times 3$ array `circles`, where each row of `circles` corresponds to the parameters (x_c, y_c, r) of a circle. Here $\vec{x}_c = (x_c, y_c)^T$ denotes the image position of the center of the circle, and r denotes the radius of the circle. Your implementation of `getProposals` should attempt to produce up to `numGuesses` proposals.

If you have trouble with this step, you can begin by writing `getProposals` so that it simply generates proposals from moused in points (say a center point plus one point on the circle). You won't get any marks for this, but it will help you get started on other parts of the problem.

In your write up, clearly explain the algorithm you used to automatically generate circle proposals, along with the reasons why you chose it over other possibilities.

(b) **Circle selection.** The next step in `findCellScript.m` is to select the best circle from the proposed circles. For example, a good circle might be close to many edgels in the data. Implement your selection process in the function `bestProposal` so that it chooses a promising circle from the list of proposals. In your write up, clearly describe how you select the best circle (i.e., be specific about what you mean by "best"), and explain your reasons for this choice.

(c) **Robust fitting.** Derive an IRLS algorithm for estimating the circle parameters $(\vec{x}_c, r)$ which (locally) minimize the objective function

$$\mathcal{O}(\vec{x}_c, r) \;=\; \sum_k \rho(e_k(\vec{x}_c, r), \sigma_g). \tag{5}$$

Here e_k is as defined in (3). In deriving the update equations for $(\vec{x}_c, r)$, when you differentiate the objective function $\mathcal{O}(\vec{x}_c, r)$ you can treat the outward pointing normal $\vec{n}_c(\vec{x}_k)$, defined in (4), as if it was independent of $\vec{x}_c$. Setting this approximate gradient equal to zero then provides a weighted set of linear equations for $\vec{x}_c$ and r. As is standard in IRLS algorithms, the weights also depend on the unknowns $\vec{x}_c$ and r. Develop an IRLS algorithm which uses the solution at the previous iteration to evaluate both $\vec{n}_c(\vec{x}_k)$ and the weights, and then solves the resulting weighted linear equations for the updated values $\vec{x}_c$ and r.

In your write-up, include the derivation of the these equations for $\vec{x}_c$ and r, and clearly describe your IRLS algorithm. Also, explain how changes in an edgel normal $\vec{n}_k$ effects these equations. Does it change direction of the contribution to the gradient for this edgel, or just the weight?

Complete the function `fitCircleRobust` to implement this IRLS algorithm for robustly fitting a circle to your edgel data. In order to test your code, you can initially set `demoRobustConv` to `true`. This displays how your code behaves for each initial guess provided by `getProposals` in step (2a) above.

Experiment with different values of the robust estimator's scale parameter σ_g (`sigmaGM` in the code). For $\beta = 0$, what are the converged results when `sigmaGM` is large (e.g. 100)? What are the results like when `sigmaGM` is small (e.g. 0.1)? Set `sigmaGM` to an appropriate value between these two extremes. And then, finally, choose an appropriate value for β. In your write up, explain how you chose suitable values for these parameters σ_g and β.

Before continuing on, set `demoRobustConv` back to `false`.

(d) **Model update.** Finally, given the robustly fit circle, decide if it should be kept in the model (see the function `isGoodCircle`). If it is decided that it should be kept, then the handout code greedily removes the edgels from the data with sufficiently high weights. Steps 2a-d are then repeated to fit additional circles. The cell finder is now complete.

In your write up, describe on what basis your `isGoodCircle` function decides to keep a new circle, and explain why you chose this.

(e) **Brief evaluation.** Your completed code should now provide a very reasonable first cut at solving our geneticist's problem. It should be expected to miss cells that have been significantly cropped due to the side of the images. Also, small buds on some cells might be missed (see Fig. 1). Other than these two "failure modes", your algorithm should work very well. What other situations are observed to cause your algorithm to fail to find a plausible set of circles? Briefly describe what you might do to try to fix these remaining problems.