CSC487/2503—Foundations of Computer Vision, Fall 2007

Assignment 2: Robust Estimation of Image Models

*Due: 1:10pm, Thurs., Nov. 1 (*before *the lecture)*
This assignment is worth 15 marks towards your grade in this course.

Robust estimation of parameterized models. In your new job as a machine vision specialist, a geneticist drops by with microscope images such as the one depicted in Fig. 1a. She wishes to automatically count the cells in these images, and take specific note of cells that are in various stages of splitting. Here we consider two applications of robust estimation that together will get us started on a solution for the geneticist.

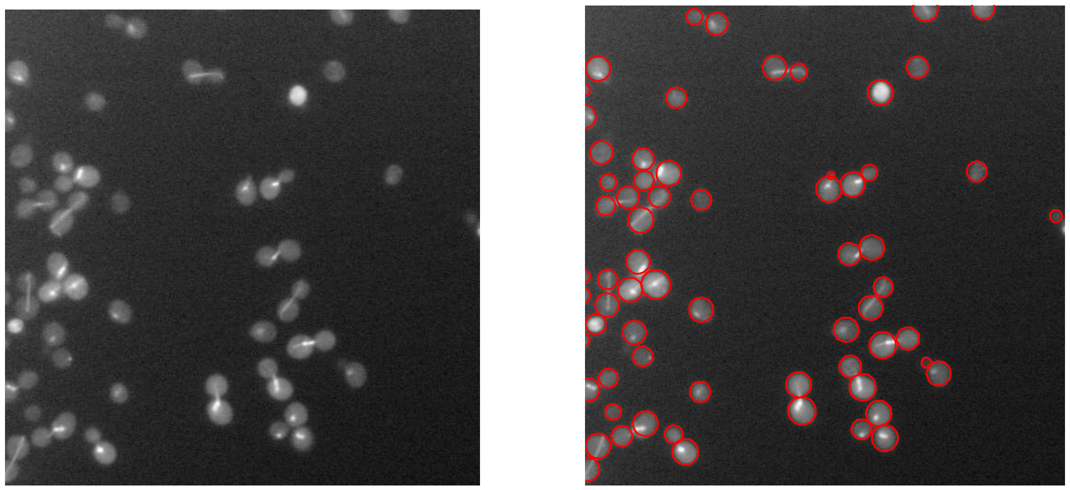

Figure 1: Cell image (a) with fitted cells (b) (view in colour).

What to hand in. Write a short report addressing each of the itemized questions below. The reports can be electronic, in the form of a PDF file. Alternatively, neatly hand-written reports including printed copies of the images output by your program are also acceptable. 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, your completed Matlab code should be in one directory A2/, and either zip or tar it up. Email the zip/tar ball to `mbrubake@cs.utoronto.ca`.

Download the starter code `cellFinder.zip` from the course homepage.

Undergraduates. Students registered in CSC487 (i.e., undergraduates) only need to do question 2 below, although I will give you a 50% bonus if you successfully complete both questions. Grad students will be marked on both questions.

1. **Smooth background model [20pts].** Our first task is to model the slowly varying background brightness in the microscope images. We will make use of this background model to differentiate between cell pixels and background pixels, and thereby detect cells. In the next question we will model the shapes of these detected cells. Specifically, our first goal is to find coefficients $\vec{c}$ such that

$$I(\vec{x}) \approx \sum_{k=1}^{K} B_k(\vec{x}) c_k.$$

The basis functions $B_k(\vec{x})$ are formed using Gaussians $G_k(\vec{x}) \equiv G(\vec{x} - \vec{x}_k, \sigma_B)$, having standard deviation σ_B and centered on points $\vec{x}_k$ arranged in a coarse 2D sampling grid. The spacing between the grid points $\vec{x}_k$ is set to be $0.8\sigma_B$ in both the x and y directions. In order to represent smooth functions, we set σ_B to be large, say $\sigma_B = 100$ pixels.

Given an image $I(\vec{x})$, consider the robust objective function

$$\mathcal{O}_d(\vec{c}) \equiv \sum_{\vec{x}\in R} \rho\left(e(\vec{x};\vec{c}), \sigma_g\right), \quad e(\vec{x};\vec{c}) \;\equiv\; I(\vec{x}) - \sum_{k=1}^{K} B_k(\vec{x}) c_k \tag{1}$$

Here R is the whole set of image pixels. Here we will set $\rho(e, \sigma_g)$ to be the Geman-McLure (GM) estimator introduced in class,

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

The vector of unknown coefficients $\vec{c} = (c_1, \ldots, c_K)^T$ for the smooth background is then chosen by minimizing this objective function $\mathcal{O}_d(\vec{c})$.

1a. Using the GM estimator $\rho(e, \sigma_g)$ in equation (1), derive an iteratively reweighted least squares algorithm for computing the coefficients $\vec{c}$. That is, given an initial guess $\vec{c}_0$, we iterate an equation of the form

$$D_\nu \vec{c}_{\nu+1} = \vec{b}_\nu. \tag{3}$$

Here D_ν is a $K \times K$ matrix and $\vec{b}_\nu$ is a K-vector, where K is the number of RBF kernels in (1). Both D_ν and $\vec{b}_\nu$ depend on the robust weights $w(e_\nu)$, which are evaluated using the current coefficient vector $\vec{c}_\nu$. The solution $\vec{c}_{\nu+1}$ of this matrix equation provides the updated coefficient vector.

Show the derivation of expressions for D_ν and $\vec{b}_\nu$ by following a roughly similar approach to the one for line estimation done in the class notes.

1b. Implement the iteratively reweighted least squares approach described in equation (3) above by completing the M-file `robustBack.m`. Note that many of the tools you need to do this are provided in the `rbf/` directory of the handout code. The implementation requires an initial guess for $\vec{c}_0$ and a value for the scale σ_g of the robust estimator. Use a least squares estimate for the initial guess (i.e., for $\nu = 0$ in (3), set all the weights used to form D_0 and $\vec{b}_0$ to be one).

For subsequent iterations, in order to set σ_g in the robust estimator, we need an estimate for standard deviation of the noise for the inliers. A simple way to do this is to approximate the errors for the inliers to be Normally distributed, and to assume the median absolute error $|e(\vec{x})|$ is approximately the median of the absolute values from this Normal distribution. To match the width of the GM weight function to the standard deviation of the noise, use

$$\sigma_g = 2\text{median}(|\vec{e}|). \tag{4}$$

This value of σ_g is updated each time the coefficients are updated in the IRLS algorithm.

Complete the IRLS implementation and test it on the images provided in the `images/` directory in the handout. Convergence may require 20 to 50 or so iterations.

In your write up, include the following images as displayed by your solution code: i) the original image, $I(\vec{x})$, ii) the estimated background image $b(\vec{x}) \equiv \sum_k B_k(x) c_k$, iii) the error image $I(\vec{x}) - b(\vec{x})$, and iv) the GM weight image $w(\vec{x})$. Comment on the behaviour of this algorithm. How well does it estimate the background for the images tested? Can you identify specific issues with the approach? Explain.

1c. On some images the converged value of σ_g (when the approach described in 1b. above is used) is rather large. In particular, it is significantly larger for `shirt03.pgm` than either of other two `shirt##.pgm` images. Explain why this happens.

In order to avoid this, consider a form of deterministic annealing where we force σ_g to decrease to some prespecified value $\sigma_{g,min}$. For example, on iteration $\nu + 1$ in (3) take

$$\sigma_{g,\nu+1} \;=\; \max\left[\sigma_{g,min}, \min\left\{2\text{median}(e_\nu), \rho\sigma_{g,\nu}\right\}\right], \tag{5}$$

where ρ is a constant less than one (e.g. $\rho = 0.9$). When $\sigma_{g,\nu}$ is significantly larger than $\sigma_{g,min}$, the above rule sets the new value $\sigma_{g,\nu+1}$ to be less than or equal to ρ times the previous value. That is, $\sigma_{g,\nu}$ is forced to decrease to $\sigma_{g,min}$. As in part 1b, this whole process can be started at $\nu = 0$ using the least squares solution (and, for $\nu = 1$, set $\sigma_{g,0} = 2\text{median}(e_0)$).

Comment on the change in the performance on the `shirt03.pgm` image due to this annealing.

1d. A second issue with the implementation in 1b is that, on some images, the data contributing to a particular RBF basis function may all (or mainly) have small weights (e.g., try `images/f0300crop.pgm`, for example). This situation causes the matrix D_ν in (3) to be nearly singular. To deal with these cases we add a "regularization" term $\mathcal{O}_r(\vec{c})$ to the objective function (1). The general idea is that this regularizer helps control the solution in these weakly constrained regions, but provides only a small bias on the solution in strongly constrained regions.

We consider the regularizer

$$\mathcal{O}_r(\vec{c}) = \alpha_r \sum_{k(i,j)} (c_{k(i+1,j)} + c_{k(i-1,j)} - 2c_{k(i,j)})^2 + (c_{k(i,j+1)} + c_{k(i,j+1)} - 2c_{k(i,j)})^2. \tag{6}$$

Here $c_{k(i,j)}$ denotes the RBF coefficient in the i^{th} row and j^{th} column of the spatial grid of RBFs. The sum in (6) is only taken over points (i, j) away from the boundary of the grid (so $c_{k(i\pm1,j\pm1)}$ are all defined). Finally, the constant α_r in (6) determines the relative importance of this regularization term, as compared to the so-called "data term" in (1).

Derive the form of the equations for $\vec{c}$ which minimize the regularized objective function

$$\mathcal{O}(\vec{c}) = \mathcal{O}_d(\vec{c}) + \mathcal{O}_r(\vec{c}). \tag{7}$$

Hint: The equations are only a small modification of the ones you previously derived in part 1a. Modify your IRLS solver from parts 1b and 1c to minimize this regularized objective function. Discuss the behaviour of the solutions for various values of α_r.

2. **Robustly Fitting Circles [20pts].** The handout code `findCell.m` first loads previously saved results from a solution of question 1. In particular, the image, the fitted background image (using $\sigma_B = 100$ in the RBF functions), and the weight image are recovered from a `.mat` file for the selected cell image. For a given cell image, the handout code detects cell pixels using the weight image, and clusters these detected cells into connected segments. Canny edgels are computed on results these segmentation results. 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 just 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, which points in the direction of increasing brightness. Let $\vec{t}_k = (n_{2,k}, -n_{1,k})^T$, which is a unit vector in the direction tangent to the edgel.

2a. **Circle proposals.** The first step in the inner-most loop in `findCell.m` is a call to `getProposals`. When finished this function should return a $P \times 3$ array `circles`, where each row of `circles` corresponds to the parameters (xc, yc, r) of a circle. Here $\vec{x}_c = (xc, yc)^T$ denotes the image position of the center of the circle (in the original image, not the cropped image), and r denotes the radius of the circle. In your solution you can change the number and type of `getProposals`' parameters, and also change the returned values (so long as the `circles` array described above is returned).

The approach we advocate for implementing `getProposals` is to randomly sample two of the edgels provided by its parameters. Use the position and orientation of these two randomly sampled edgels to propose a potential circle which passes close to these two image points, and with roughly the observed normals. (If you cannot solve for a suitable circle for a particular pair of edgels, then sample again.) Repeat this until you have `P = numGuesses` proposals. The remainder of `getProposals` in the handout code will then display your circle proposals.

2b. **Circle selection.** The next step in `findCell.m` is to select the best circle from the proposed circles. A good circle might be close to many edgels in the data.

One way to define this is to use the GM robust error function (2) on the error in a given edgel $(\vec{x}_k, \vec{t}_k)$ with respect to the circle centered at $\vec{x}_c$ having radius r, say $e(\vec{x}_k, \vec{t}_k; \vec{x}_c, r)$. We can express this error as

$$e(\vec{x}_k, \vec{t}_k; \vec{x}_c, r) = \sqrt{[\vec{n}_c(\vec{x}_k) \cdot (\vec{x}_k - \vec{x}_c) - r]^2 + [\vec{n}_c(\vec{x}_k) \cdot \vec{t}_k]^2 \beta}. \tag{8}$$

Here

$$\vec{n}_c(\vec{x}_k) = (\vec{x}_k - \vec{x}_c)/||\vec{x}_k - \vec{x}_c||, \tag{9}$$

is a unit vector in the direction of the edgel position $\vec{x}_k$ from the circle center $\vec{x}_c$. The first squared term in (8) therefore measures the squared distance between the edgel position $\vec{x}_k$ and the circle specified by $(\vec{x}_c, r)$. The second term involves the inner product of the edgel tangent with the normal to any circle centered at $\vec{x}_c$. It therefore is the square of the sine of the angular error in the edgel. The value β is a constant used to scale these angular errors so that the average magnitude of the two squared terms in (8) are roughly equal for typical (inlier) edgels.

The error in (8) can be simplified by rescaling with the norm $||\vec{x}_k - \vec{x}_c||$, and using (8, 9). We find

$$||(\vec{x}_k - \vec{x}_c)||e(\vec{x}_k, \vec{t}_k; \vec{x}_c, r) = \sqrt{[(\vec{x}_k - \vec{x}_c) \cdot (\vec{x}_k - \vec{x}_c) - r||(\vec{x}_k - \vec{x}_c)||]^2 + [(\vec{x}_k - \vec{x}_c) \cdot \vec{t}_k]^2 \beta}. \tag{10}$$

If we approximate the term $r||(\vec{x}_k - \vec{x}_c)||$ by r^2, we get the simpler scaled error,

$$\begin{aligned} s(\vec{x}_k, \vec{t}_k; \vec{x}_c, r) &\equiv \sqrt{[(\vec{x}_k - \vec{x}_c) \cdot (\vec{x}_k - \vec{x}_c) - r^2]^2 + [(\vec{x}_k - \vec{x}_c) \cdot \vec{t}_k]^2 \beta}, \\ &\approx re(\vec{x}_k, \vec{t}_k; \vec{x}_c, r). \end{aligned} \tag{11}$$

Finally, substituting $e \approx s/r$ in the GM estimator (2) we obtain a related estimator on s, namely

$$\rho(s_k, \sigma_g r) = \frac{s_k^2}{(\sigma_g^2 r^2 + s_k^2)}, \quad \text{where } \vec{s}_k \equiv s(\vec{x}_k, \vec{t}_k; \vec{x}_c, r). \tag{12}$$

Note the factor of r in the estimator's scale parameter $\sigma_g r$. We therefore seek circle parameters $(\vec{x}_c, r)$ which (locally) minimize the objective function

$$\mathcal{O}_c(\vec{x}_c, r) = \sum_k \rho(s_k, \sigma_g r). \tag{13}$$

In this part of the question we use (13) only to select the best circle proposal from our list of proposals (we consider finding local minima in the next part). We recommend that you use the sum of the robust weights associated with the estimator in (13) as a goodness measure for the proposed circles. For example, the proposed circle having the largest sum of weights might be selected as the winner. Alternatively, you might also take the circumference of the circle into account, i.e., big circles might be expected to have a larger sum of weights than little ones.

Implement your selection process in the function `bestProposal`. In your write up, clearly describe how you select the best circle (be specific about what you mean by "best").

2c. **Robust fitting.** Derive an IRLS algorithm for estimating the circle parameters $(\vec{x}_c, r)$ which minimize (13). Include this derivation in your write-up. Modify the function `fitCircleRobust` to implement this IRLS algorithm for robustly fitting a circle to your edgel data. Use the initial guess provided by part 2b.

2d. **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 (and why) your `isGoodCircle` function decides to keep a new circle. Finally, briefly describe any issues you noticed with the overall implementation.