CSC2503—Foundations of Computer Vision, Fall 2016

Assignment 2: Camera Obscura & Robust Estimation

Due: 2pm, Wednesday, November 16

This assignment has two parts that can be tackled independently. Robust estimation, needed for Part B, will be covered in class next week.

Part A: Build your own camera obscura

Your task is to turn a room, a box or any other box-shaped space whose lighting you can control into a real-life *camera obscura* (*a.k.a* pinhole camera) by letting light enter that space through a very small hole. If everything is done right, a sharp image of the scene outside your camera obscura will be projected upside down on the camera's back plane. The image may be quite dim but should be visible to the naked eye (make sure that the scene outside the camera obscura is *very* bright).

1. **Basic operation:** After you have constructed your camera obscura and formed a sharp image, use a digital camera or smartphone to take pictures of your setup and of the image being formed. Specifically, take pictures of (a) the scene(s) in front of your camera obscura, (b) the corresponding image formed on the camera obscura's back plane, and (3) your setup. Your pictures should be sufficient to convince the TA that you created the camera obscura yourself and that it behaves as expected.

   Since the images formed by your camera obscura will be dim, you will likely need to use a long exposure to record them with your digital camera. To avoid motion blur, ensure that your digital camera does not move at all during the exposure (*e.g.,* you may wish to mount your digital camera on a tripod). If your digital photos are too dim, you may linearly scale them to improve visibility.

2. **Experimenting with different pinhole shapes & sizes:** Now slightly increase the size of your camera obscura's pinhole so that it becomes a disc and the images it forms become blurry. As before, take a picture of the blurry image, of the scene outside and of the pinhole itself. Finally, repeat this process for a square-shaped, a cross-shaped and a hexagonal pinhole.

3. **Modeling lens-less image formation:** If done properly, the images in (2) will differ from each other and from the sharp image obtained with the tiny pinhole you used in (1). Derive an expression that relates the "ideal" image $I(x, y)$ of the scene, captured with an infinitely-small pinhole, to the image $I_{\mathcal{S}}(x, y)$ formed when the pinhole's shape is a 2D curve $\mathcal{S}$. Your derivation should assume the following: (a) the camera's front and back planes are parallel to each other; (b) the pinhole lies on the camera's front plane; (c) the scene is a Lambertian 2D plane that is parallel to the camera's front plane; and (d) the camera's back plane has infinite size, *i.e.,* x and y range from $-\infty$ to ∞. Finally, you should comment on why the Lambertian assumption and the scene's planarity and parallelism to the camera's front plane are essential in your derivation.

What to submit for Part A. Write a report that addresses the itemized tasks (1)-(3) above. Your report should be in PDF format and should include the images requested. LaTeX- or Word-formatted reports are preferred; a hand-written derivation that has been scanned to PDF is also fine *as long as it is very legible.* Assume the marker knows the context of all questions, so do not spend time repeating material in the hand-out or the class notes.

Part B: Robustly Fitting Circles

The goal of this part of the assignment is to gain experience using robust estimation to extract simple image models from noisy image measurements.

To begin: Download the starter code `A2handout.zip` from the course homepage.

What to submit for Part B. Write a short report addressing each of the itemized questions below. Pack your PDF report for Part B and the completed Matlab files for each question, along with the PDF report for Part A, into a file called `A2.zip`, and submit the zip file electronically.

Robustly Fitting Circles. A geneticist wishes to automatically count cells in microscope images, such as the one depicted in Fig. 1 (left). The geneticist is especially interested in cells at different stages of cell division. Here we consider an application of robust estimation that will get us started on a solution for the geneticist. Cells are actually elliptical but we'll begin by fitting circles.

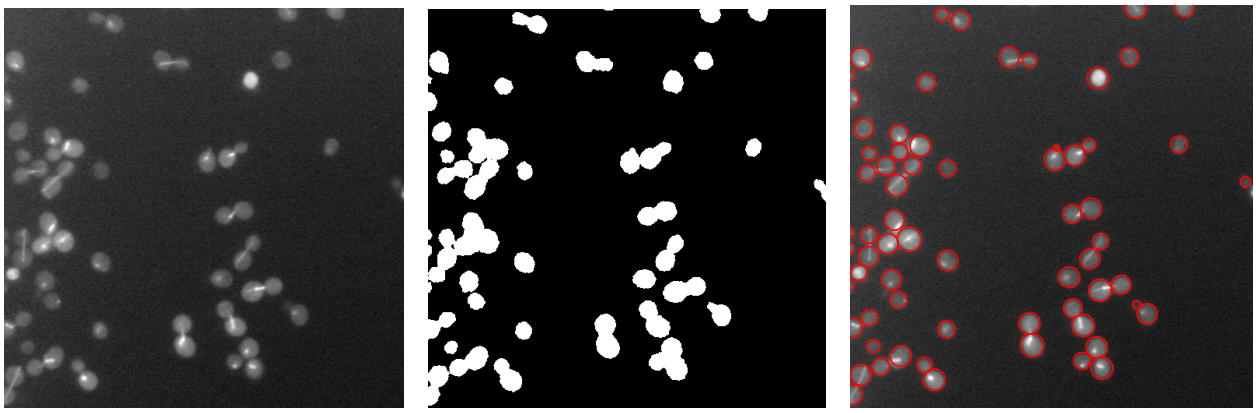

Figure 1: Cell image (left), foreground pixels (middle), and fitted cells (right, in red).

The main script in the handout code, called `findCellScript.m`, reads in a microscope image of cells (Fig. 1 (left)). We have already preprocessed the cell images to detect foreground cell pixels (Fig. 1 (middle)). We have also labelled the connected sets of foreground pixels, which are read in as a second image. Finally, the script computes Canny edgels from the foreground mask. The code then considers each connected component individually, looking for circular cells (see Fig. 1 (right)).

Your job is to fit circles to the edgels obtained from the boundaries of the connected components. Each fitted circle should correspond to one "cell" including, perhaps, a small circle for a cell bud just being born, by splitting off from another cell (see Fig. 1 (right)). This is a vague definition of what constitutes a "cell". However, if you study Fig. 1 (right) (e.g., by enlarging the electronic copy), it should be intuitively clear what is meant by a "cell". Part of your job in this assignment is to decide on how to operationally define our intuitive notion of one cell.

We aim to fit circles using edgel measurements. Let the k^{th} edgel be denoted by $(\vec{x}_k, \vec{n}_k)$, where $\vec{x}_k$ is the measured image position and $\vec{n}_k$ is the measured edgel normal. The edgel normal points in the direction of increasing brightness of the foreground regions Fig. 1 (middle); i.e., they point towards the *interior* of the cells.

Because cells occur in close proximity, a single connected foreground component will often contain

more than one cell. It is therefore natural to formulate circle fitting as a robust estimation problem. That is, a set of edgel measurements $\{\vec{x}_k, \vec{n}_k\}_{k=1}^N$ from a single connected component will often correspond to multiple cells. When estimating the parameters of one circle (cell) we can view the measurements from any other circle (cells) as outliers. To this end we will use the Geman-McLure (GM) estimator (to be introduced in class next week),

$$\rho(e, \sigma_g) = \frac{e^2}{\sigma_g^2 + e^2} \ . \tag{1}$$

In what follows we are going to take a greedy approach to this problem, finding one circle at a time. The approach comprises four main steps: 1) quickly generating a set of plausible circle hypotheses, 2) choosing one among them as an initial guess for optimization, 3) robust estimation to optimize the fit, and 4) the model update, for which we decide whether or not the optimized fit is sufficient and which edgel measurements are *owned* by the circle.

Below there are several questions, including some derivations and some programming. To simplify the programming, the assignment handout includes a partial implementation of the general strategy outlined above, i.e., `findCellScript.m`. For the programming parts of the assignment you need to complete several Matlab m-functions, as described below. You may also make small changes to the handout code.

1. **Noise model:** To estimate the parameters of a circle, we'll focus on the use of the edgel position measurements $\vec{x}_k$. The key constraint on position measurements is that the distance from the center to any position on the circle must equal the radius. For measurement $\vec{x}_k$, and a circle with centre $\vec{x}_c$ and radius r, we can write the error as

$$e_k \ = \ ||\vec{x}_k - \vec{x}_c||^2 - r^2 \ = \ \vec{a}^{\,T}\vec{x}_k + b + \vec{x}_k^{\,T}\vec{x}_k \ . \tag{2}$$

   where $\vec{a} \equiv -2\vec{x}_c$ and $b \equiv \vec{x}_c^{\,T}\vec{x}_c - r^2$. The change in parameterization, from $\vec{x}_c$ and r to $\vec{a}$ and b is convenient because e_k is linear in $\vec{a}$ and b. Of course it is straightforward to find the circle's centre $\vec{x}_c$ and radius r from $\vec{a}$ and b.

   To properly formulate an estimator we should understand the nature of the expected errors, e_k. To that end, let's assume that the observed positions (measurements) are equal to the true positions, denoted $\hat{\vec{x}}_k$, plus independent, isotropic Gaussian noise $\vec{m} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$; i.e.,

$$\vec{x}_k \ = \ \hat{\vec{x}}_k + \vec{m} \ , \quad \text{where } \ p(\vec{m}) = \frac{1}{2\pi\sigma^2}\, e^{-\vec{m}^T\vec{m}/2\sigma^2} \ . \tag{3}$$

   With this measurement noise model, what is the distribution of the errors e_k? How does it depend on the circle parameters $\vec{x}_c$ and r?

2. **LS estimator:** Suppose we approximate the distribution of the errors, e_k, as a mean-zero Gaussian (Normal) distribution. In other words, assume that the only source of error on the RHS of Eqn. (2) is additive, mean-zero Gaussian noise. What is a reasonable choice for the variance of the Gaussian noise?

   Assuming that you are given a set of independent measurements, $\{\vec{x}_k\}_{k=1}^N$, all of which arise from the same circle, a natural way to derive an estimator is to formulate the negative log likelihood of the measurements. The maximum likelihood estimator is found by minimizing the negative log likelihood with respect to the unknowns $\vec{a}$ and b. Beginning with this likelihood, derive a least-squares estimator.

3. **Circle proposals.** The first step of our greedy strategy is to generate a set of circle proposals for a given set of edgel measurements $\{(\vec{x}_k, \vec{n}_k)\}_{k=1}^N$. For this step you may find it helpful to use both the position measurements and the normal measurements.

   This requires that you complete the m-function `getProposals.m` (in the `circleFit` subdirectory). When finished this function should return a $P \times 3$ array named `circles`. Each row of `circles` corresponds to the parameters (x_c, y_c, r) of a circle, where $\vec{x}_c = (x_c, y_c)^T$ 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 efficiently produce up to `numGuesses` proposals. The idea is to quickly find initial guesses from which you can select one to be subsequently refined with the robust estimator. Since this will be run numerous times, it is essential that this step require minimal computational effort. 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.

   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.

4. **Circle selection.** The next step in `findCellScript.m` is to select the best circle from the proposed circles in the first step (Question 3 above). 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.

5. **Robust fitting.** Beginning with the *best* circle from the second step (Question 4) as an initial guess, the next step is to optimize the fit. To this end, derive an IRLS algorithm for estimating the circle parameters $(\vec{a}, b)$ which (locally) minimize the objective function

$$\mathcal{O}(\vec{a}, b) \;=\; \sum_k \rho(e_k(\vec{a}, b), \sigma_g) \;. \tag{4}$$

   Here e_k is as defined in (2). In your write-up, include the derivation of the these equations for $\vec{a}$ and b, and clearly describe your IRLS algorithm.

   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 the second step above. Experiment with different values of the robust estimator's scale parameter σ_g (`sigmaGM` in the code). Ideally, we would choose σ_g^2 to be equal to the variance of the errors e_k for inlier measurements. In practice, describe is a suitable way to choose σ_g for your implementation. What goes wrong when σ_g is much too large or much to small?

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

6. **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 with sufficiently high weights from the data. In your write up, explain how you would choose a good threshold to be applied to the weights. Also, describe on what basis your `isGoodCircle` function decides to keep a new circle, and justify your choice.

   These four steps (Questions 3-6) are then repeatedto fit additional circles. The cell finder is now complete.

7. **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 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.