CSC2503 Fall 2018: Homework 1

Niloufar Afsariardchi

October 17, 2018

1 Noise model

Important Note: I get a memory error when I run my code on CDF with GUI! Please run it without a MATLAB GUI using:" matlab -nodisplay -r findCellScript " command. I don't have any issue on personal laptop though...

We write e_k in terms of true measurement and the error:

$$\begin{aligned} e_k &= \vec{a}^T\vec{x}_k + b + \vec{x}_k^T\vec{x}_k && (1)\\ &= \vec{a}^T(\hat{\vec{x}}_k + \vec{m}) + b + (\hat{\vec{x}}_k + \vec{m})^T(\hat{\vec{x}}_k + \vec{m}) && (2)\\ &= \vec{a}^T\hat{\vec{x}}_k + b + \hat{\vec{x}}_k^T\hat{\vec{x}}_k + (2\hat{\vec{x}}_k + \vec{a})^T\vec{m} + \vec{m}^T\vec{m} && (3) \end{aligned}$$

The first three terms of equation (3) could be zero if we *fit* a circle to true edgel measurements of a *single* circle. These terms are deterministic. To write down the pdf of e_k in closed form, we rewrite (3)

$$e_k = b - \frac{1}{4}\vec{a}^T\vec{a} + (\vec{m} + (\hat{\vec{x}}_k + \frac{\vec{a}}{2}))^T(\vec{m} + (\hat{\vec{x}}_k + \frac{\vec{a}}{2})) \quad (4)$$

defining $\vec{c}_k = \vec{m} + (\hat{\vec{x}}_k + \frac{\vec{a}}{2})$, we have $p(\vec{c}_k) = \mathcal{N}(\hat{\vec{x}}_k + \frac{\vec{a}}{2}, \sigma^2\mathbf{I}) = \mathcal{N}(\hat{\vec{x}}_k - \vec{x}_c, \sigma^2\mathbf{I})$. Therefore,

$$e_k = -r^2 + \vec{c_k}^T\vec{c_k} \quad (5)$$

The probability distribution of $\vec{x}^T\vec{x}$ where is $\vec{x}$ is a Gaussian distribution with median $\vec{\mu}$ and unit variance is a non-central chi-squared [1] (χ^2) distribution with parameter $\lambda = \vec{\mu}^T\vec{\mu}$ and degree of freedom $k = 2$. Therefore,

$$\begin{aligned} \frac{\vec{c_k}^T\vec{c_k}}{\sigma^2} &\sim \text{non-central}\chi^2\Big(\frac{\vec{c_k}}{\sigma}, \lambda = (\hat{\vec{x}}_k - \vec{x}_c)^T(\hat{\vec{x}}_k - \vec{x}_c), k = 2\Big) && (6)\\ \vec{c_k}^T\vec{c_k} &\sim \sigma^2 \times \text{non-central}\chi^2\Big(\frac{\vec{c_k}}{\sigma}, \lambda = (\hat{\vec{x}}_k - \vec{x}_c)^T(\hat{\vec{x}}_k - \vec{x}_c), k = 2\Big) && (7) \end{aligned}$$

Then e_k has the distribution above but with its mean shifted for $-r^2$ as in equation (5).

[1] https://en.wikipedia.org/wiki/Noncentral_chi-squared_distribution

2 LS estimator

We first find the mean of e_k,

$$\begin{aligned} E[e_k] &= \vec{a}^T\hat{\vec{x}}_k + b + \hat{\vec{x}}_k^T\hat{\vec{x}}_k + (2\hat{\vec{x}}_k + \vec{a})^T E[\vec{m}] + E[\vec{m}^T\vec{m}] && (8)\\ &= \vec{a}^T\hat{\vec{x}}_k + b + \hat{\vec{x}}_k^T\hat{\vec{x}}_k + E[\vec{m}^T\vec{m}] && (9)\\ &= \vec{a}^T\hat{\vec{x}}_k + b + \hat{\vec{x}}_k^T\hat{\vec{x}}_k + 2\sigma^2 && (10)\\ &= \hat{e}_k + 2\sigma^2 && (11)\end{aligned}$$

where $\hat{e}_k = \vec{a}^T\hat{\vec{x}}_k + b + \hat{\vec{x}}_k^T\hat{\vec{x}}_k$ and in equation (8) and (10) we used the fact that $\vec{m}$ has zero mean. In equation (10), we used $E[X^2] = \sigma_X^2 + \mu_X^2$ formula. For the variance,

$$\begin{aligned} &\mathrm{Var}(e_k) = E[e_k^2] - E[e_k]^2 && (12)\\ &= E[(\hat{e}_k + \vec{q}_k^T\vec{m} + \vec{m}^T\vec{m})^T(\hat{e}_k + \vec{q}_k^T\vec{m} + \vec{m}^T\vec{m})] - E[e_k]^2 && (13)\\ &= \hat{e}_k^2 + 2\hat{e}_k E[\vec{m}^T\vec{m}] + E[(\vec{q}_k^T\vec{m})^2] + 2E[(\vec{q}_k^T\vec{m})(\vec{m}^T\vec{m})] + E[(\vec{m}^T\vec{m})^2] - E[e_k]^2 && (14)\\ &= \hat{e}_k^2 + 4\hat{e}_k\sigma^2 + \sigma^2\vec{q}_k^T\vec{q}_k + 2E[(\vec{q}_k^T\vec{m})(\vec{m}^T\vec{m})] + && (15)\\ &E[(\vec{m}^T\vec{m})^2] - \hat{e}_k^2 - 4\sigma^4 - 4\hat{e}_k\sigma^2 && (16)\\ &= \sigma^2\vec{q}_k^T\vec{q}_k + 2E[(\vec{q}_k^T\vec{m})(\vec{m}^T\vec{m})] + E[(\vec{m}^T\vec{m})^2] - 4\sigma^4 && (17)\end{aligned}$$

where $\vec{q}_k = 2\hat{\vec{x}}_k + \vec{a}$. Assuming $\vec{m} = [m_1 m_2]$, we write

$$\begin{aligned} E[(\vec{m})(\vec{m}^T\vec{m})]]_1 &= E[m_1(m_1^2 + m_2^2)] && (18)\\ &= \int_{m_1}(m_1^3 + m_1 m_2^2)\frac{1}{\sqrt{2\pi}\sigma}e^{\frac{(m_1^2+m_2^2)}{2\sigma^2}}dm_1 && (19)\end{aligned}$$

The odd moments of zero mean Gaussian distribution are zero, hence $E[(\vec{q}_k^T\vec{m})(\vec{m}^T\vec{m})] = 0$. Moreover, the variable $c = \vec{m}^T\vec{m}$ has the distribution $c \sim \sigma^2\chi^2(k=2)$, the mean and variance of chi-squared random variable are k and $2k$ respectively, where k is the degree of freedom[2]. Therefore $E[c^2] = \mathrm{Var}(c) + E[c]^2 = 4\sigma^4 + (2\sigma^2)^2 = 8\sigma^4$. Inserting these in equation (17) gives

$$\begin{aligned} \mathrm{Var}(e_k) &= \sigma^2\vec{q}_k^T\vec{q}_k + 8\sigma^4 - 4\sigma^4 && (20)\\ &= \sigma^2\vec{q}_k^T\vec{q}_k + 4\sigma^4 && (21)\\ &= 4\sigma^2(\hat{\vec{x}}_k - \vec{x}_c)^T(\hat{\vec{x}}_k - \vec{x}_c) + 4\sigma^4 && (22)\\ &= 4\sigma^2(\hat{e}_k + r^2) + 4\sigma^4 && (23)\end{aligned}$$

Now assume that we want to approximate the distribution $e_k \sim f(x)$ with the distribution $e'_k \sim f'(x) \sim \mathcal{N}(0, \sigma'^2)$. To find the best σ', we minimize KL divergence between the two distributions,

$$\mathrm{KL}(e_k||e'_k) = \int f(x)\log\frac{f(x)}{f'(x)}dx \quad (24)$$

[2] https://en.wikipedia.org/wiki/Chi-squared_distribution

Taking the derivative with respect to σ', we find

$$\begin{aligned}
\frac{\partial \mathrm{KL}(e_k||e'_k)}{\partial \sigma'} &= \int -f(x)\frac{\partial \log f'(x,\sigma')}{\partial \sigma'}dx && (25)\\
&= \int -\frac{f(x)}{f'(x,\sigma')}\frac{\partial f'(x,\sigma')}{\partial \sigma'}dx && (26)\\
&= \int -\frac{f(x)}{f'(x,\sigma')}\Big(\frac{1}{\sqrt{2\pi}\sigma'^2}\exp^{-\frac{x^2}{2\sigma'^2}} - \frac{x^2}{\sqrt{2\pi}\sigma'^4}\exp^{\frac{-x^2}{2\sigma'^2}}\Big)dx && (27)\\
&= \int -f(x)\Big(\frac{1}{\sigma'} - \frac{x^2}{\sigma'^3}\Big)dx && (28)\\
&= -\frac{1}{\sigma'}(1 - \frac{E[e_k^2]}{\sigma'^2}) && (29)
\end{aligned}$$

Equating above to zero,

$$\begin{aligned}
\sigma'^2 &= E[e_k^2] = \mathrm{Var}(e_k) + E[e_k]^2 && (30)\\
&= 8\sigma^4 + 4\sigma^2(\hat{e}_k + r^2) + 4\sigma^2\hat{e}_k + \hat{e}_k^2 && (31)\\
&= 8\sigma^4 + 4\sigma^2 r^2 + 8\sigma^2\hat{e}_k + \hat{e}_k^2 && (32)
\end{aligned}$$

When the measurements are taken from a *single* circle, the *true* positions would exactly lie on the best fit circle, meaning $\hat{e}_k = 0$, and for that specific circle of radius r, the variance is $\sigma'^2 = 4\sigma^2 r^2 + 8\sigma^4$.

(2) continued:

We write log-likelihood as

$$\mathcal{L} = -\log P(e_1, \dots e_N | \vec{a}, b) = -\log \prod_{i=1}^{N} P(e_i | \vec{a}, b)$$

$$= -\sum_{i=1}^{N} \log P(e_i | \vec{a}, b)$$

$$= -\sum_{i=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\sigma'^2}} e^{-e_i^2/2\sigma'^2} \right)$$

$$= -\sum_{i=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\sigma'^2}} \right) + \sum_{i=1}^{N} \frac{e_i^2}{2\sigma'^2} \quad (*)$$

where $\sigma'^2 = 8\sigma^4 + 4\sigma^2 r^2 + 8\sigma^2 \hat{e}_k + \hat{e}_k^2$
$= 8\sigma^4 + 4\sigma^2 \left(\frac{\vec{a}^T \vec{a}}{4} - b \right) - 8\sigma^2 (\vec{a}^T \hat{x}_k + b + \hat{x}_k^T \hat{x}_k) + (\vec{a}^T \hat{x}_k + b + \hat{x}_k^T \hat{x}_k)^2$

$$\frac{\partial \mathcal{L}}{\partial \vec{a}} = -\sum_{i=1}^{N} \sqrt{2\pi\sigma'^2} \cdot \frac{-1}{\sqrt{2\pi\sigma'^2}} \frac{\partial \sigma'}{\partial \vec{a}} + \sum_{i=1}^{N} \frac{2e_i}{\sigma'^3} \frac{\partial \sigma'}{\partial \vec{a}} + \sum_{i=1}^{N} \frac{2e_i}{2\sigma'^2} \frac{\partial e_i}{\partial \vec{a}}$$

$$= \sum_{i=1}^{N} \left(\frac{\partial \sigma'}{\partial \vec{a}} \left(\frac{1}{\sigma'} - \frac{2e_i^2}{\sigma'^3} \right) + \frac{2e_i}{2\sigma'^2} \frac{\partial e_i}{\partial \vec{a}} \right)$$

This seems to be very difficult to solve... also we don't have $\hat{x}_k$ (true measurements), so we can't really implement this...
I assume σ'^2 is constant now.

From equation (*):

$$\mathcal{L} = \text{constant} + \frac{e^T e}{2\sigma'^2} \Rightarrow O(p) = e^T e$$

Minimizing above $O(p)$ is the same as LS estimation.
Which make sense because for Gaussian noise and no prior distribution maximum likelihood is the same as LS.
We write e in a vector form using

$$e_i' = x_i^2 + y_i^2 - 2x_i x_c - 2y_i y_c - r^2 + x_c^2 + y_c^2$$

$$\vec{e} = \begin{bmatrix} x_1^2 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & y_2^2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \\ x_N^2 & y_N^2 & x_N & y_N & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -2x_c \\ -2y_c \\ x_c^2 + y_c^2 - r^2 \end{bmatrix} \begin{matrix} \} \vec{a} \\ \\ \\ \\ b \end{matrix}$$

$$= \begin{bmatrix} U_{N\times 2} & V_{N\times 3} \end{bmatrix} \begin{bmatrix} 1_{2\times 1} \\ \vec{a} \\ b \end{bmatrix} = \begin{bmatrix} U_{N\times 3} & V_{N\times 3} \end{bmatrix} \begin{bmatrix} 1_{2\times 1} \\ \vec{P} \end{bmatrix}$$

where the rows of U are $[x_i^2 \; y_i^2]$ & rows of V are $[x_i \; y_i \; 1]$, or homogeneous coordinates. P is $\begin{bmatrix} \vec{a} \\ b \end{bmatrix}$.

$$O(P) = e^T e = (U1_{2\times 1} + VP)^T (U1_{2\times 1} + VP)$$

$$= 1_{1\times 2} U^T U 1_{2\times 1} + P^T V^T V P + 2 1_{1\times 2} U^T V P$$

$$\frac{\partial O(P)}{\partial P} = 2 V^T V P + 2 V^T U 1_{2\times 1} \quad \Leftarrow \text{equating this to zero we find the closed form:}$$

$$\Rightarrow P = -(V^T V)^{-1} V^T U 1_{2\times 1}$$

3 Circle proposals

For this part, I implemented a small and fast variation of Hough Transform, below you can find the steps of my algorithm:

1. Constrained the range of possible r values to minimum of 0 and maximum of the difference between the Euclidean distance of the closest and furthest edgel positions to the origin, or more specifically $r_{\max} = \max_i \|\vec{x}_i\| - \min_i \|\vec{x}_i\|$.
2. Generated the possible r values by dividing the range above to r_{bin} equal spaces.
3. Similarly constrained the possible $[x_c, y_c]$ pairs to the range
   $x_c \in [\min_i ||\vec{x}_i||_1 - r_{\max}, \max_i \|\vec{x}_i\|_1 + r_{\max}]$
   and
   $y_c \in [\min_i \|\vec{x}_i\|_2 - r_{\max}, \max_i \|\vec{x}_i\|_2 + r_{\max}]$, where $\|\vec{x}_i\|_1$ and $\|\vec{x}_i\|_2$ are the first and second coordinates of the edgel position, respectively.
4. Refined the above ranges by $x_{c,\text{bin}}$ and $y_{c,\text{bin}}$
5. Constructed a grid (basically a matrix) V of possible of r, x_c, and y_c from the ranges above as parameters of Hough Transform
6. Started from $\vec{x}_i$ position, moved along the edgel normal $\vec{n}_i$ for r to find a position for the circle center $[x_c, y_c]$
7. Stored the votes for r, x_c, y_c in the grid (I casted the vote in the closest cell neighbor to r, x_c, y_c).
8. Repeated 5-6-7 for all possible r values
9. Repeated 5-6-7-8 for all $\vec{x}_i$ positions
10. Counted the votes in V and returned (r, x_c, y_c) of cells with numGuesses highest votes

The above algorithm is a mini version of a full Hough Transform, because I counted the votes assuming that the center of the circles is along the edgel normal, which may not be true considering the noisy measurements of the edgel positions and normals. Therefore, I did not parse the full parameter space for each $\vec{x}_i$. However, this approximation made the algorithm much easier to implement and less computationally intensive. Moreover, I got very reasonable results for initial proposals, I was able to propose well-matched circles for merging cells and capture all involved cells within the default numGuesses=15 proposal guesses. This algorithm was also successful in proposing reasonable circles for cropped cells near the image boundary.

In order for this to work well, I tested different refinement levels for matrix V, and realized that ranges should not be too refined because in that case the votes would be too low in each grid cell, similarly if the V is too coarse, then the resolution would become small and the circles would be off from the edgels. I experimented this with various values of r_{bin}, $x_{c,\text{bin}}$ and $y_{c,\text{bin}}$, and found that $r_{\text{bin}} = 40$, $x_{c,\text{bin}} = 80$ and $y_{c,\text{bin}} = 80$ give good results. Below you can see some of my results:

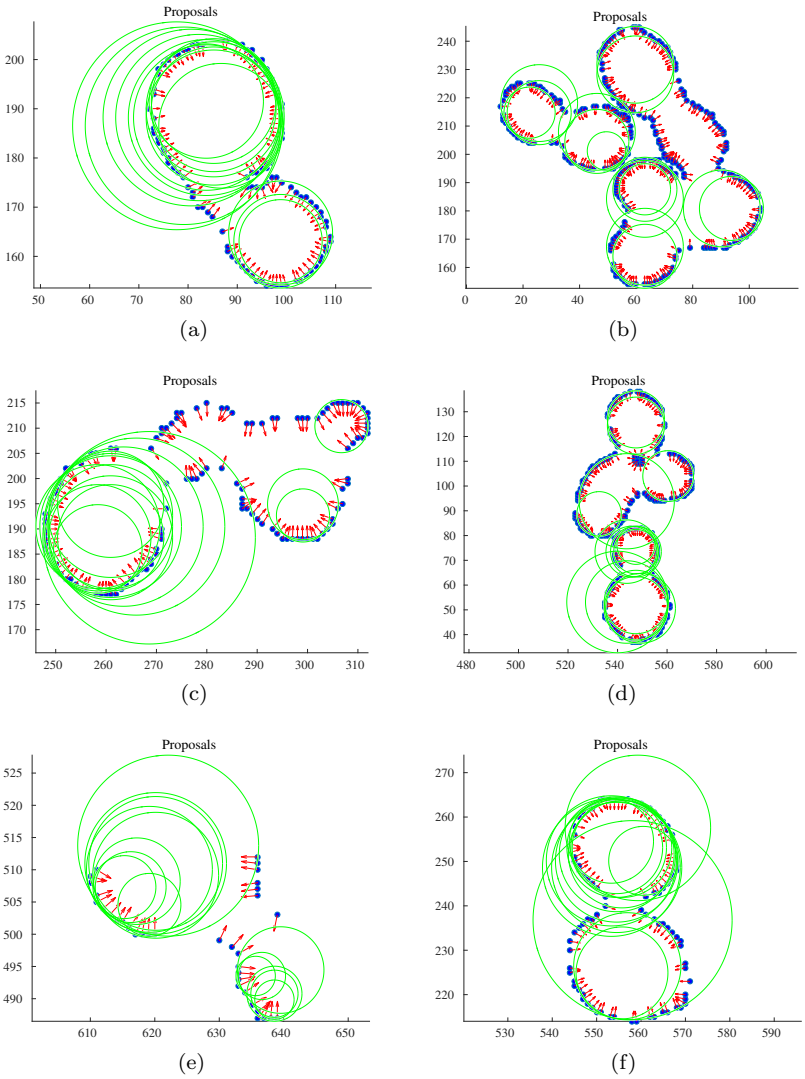


Figure 1: Sample of proposed circles

4 Circle selection

For picking the best proposed circle, I linearly combined several cost functions which show the best performances. Here are the three cost functions that I used:

1. GM: since we will be optimizing this estimator in the next question, I chose this functional form that depends on the total error. The good thing about GM is that it is robust to outliers.

2. Similarity: I used cost $= -\frac{(\vec{x}_i - \vec{x}_c).\vec{n}_i}{\|\vec{x}_i - \vec{x}_c\|}$ to use the information we have on the normal directions of the edgels. While GM tends to find a circle that is located in the middle of the blob (which has the lower error to all positions), this cost function cares about finding the normal direction right.

3. Circle Rank: since I implemented a mini version of Hough Transform in the previous question, and Hough Transform itself is a robust estimation method, I penalized on the index of circles, because I sorted all circles based on their votes for the output of getProposals function. This means that a circles at index 1 has much higher votes than a circles at index 15. **Note that if we are using another implementation of getProposals, other than what I implemented, the coefficient of this cost function must be set to zero, because in that case the ranks might not be as meaningful as here.**

I shifted and normalized all cost functions mentioned above to be in the range [0,1]. Then I re-normalized the costs this way:

$$\text{cost} \quad = \quad \frac{\text{cost} - \text{average(cost)}}{\text{std(cost)}}, \tag{33}$$

so that their scales become similar. Next, I linearly combined them and minimized the following,

$$\text{cost} \quad = \quad A \times \text{cost}_{\text{GM}} + B \times \text{cost}_{\text{Sim}} + C \times \text{cost}_{\text{Rank}} \tag{34}$$

I have tuned the coefficients A, B, and C experimentally and found $A = 1$, $B = 3$, and $C = 1$ give reasonable results.

Note: after reading the other questions, I realized that there should be a propobilistic element in this algorithm, because when the output of isGoodCircle is false, we want to have a new proposal. Therefore, I picked one of four (or the size of circles input, which ever is smaller) circles with minimum costs.

Here you can see a sample of best circle among the proposals:

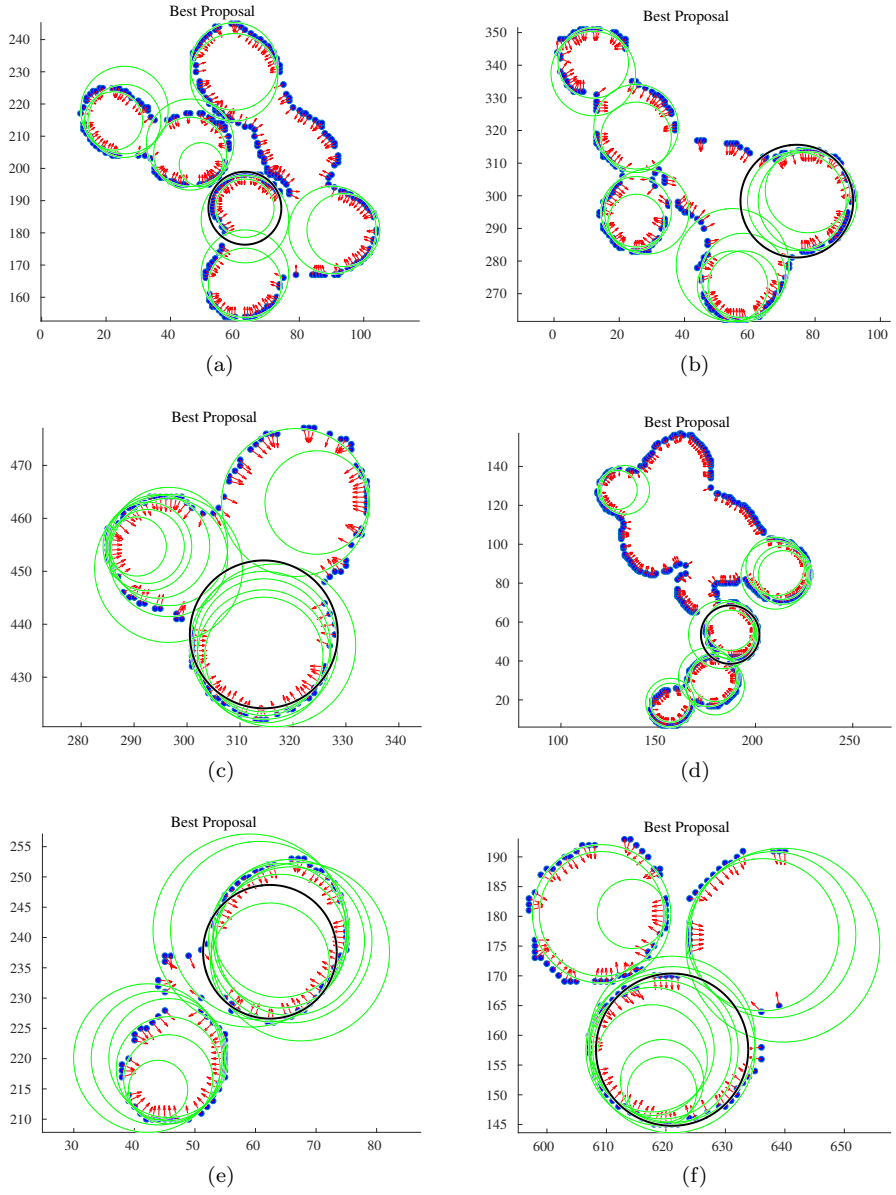


Figure 2: A sample of best circles among the proposals, shown in black color.

Question 5

The weights are defined as:

$$w_k = \frac{1}{e_k}\frac{\partial \rho(e_k)}{\partial e_k} = \frac{1}{e_k}\psi(e_k) =$$

Let's assume we have computed w_is for an iteration. We need to solve weighted least square problems, for which we should minimize:

$$O(p, w_1, w_2, \dots, w_N) = \sum_{k=1}^{k} w_k e_k^2 = e^T W e$$

; where W is a diagonal matrix with elements w_i along diagonal elements. Similar to Question 2, we find the closed form for P:

$$e = DP = \begin{bmatrix} U_{N\times 2} & V_{N\times 3} \end{bmatrix}\begin{bmatrix} 1_{2\times 1} \\ P \end{bmatrix} = U 1_{2\times 1} + VP$$

where the rows of U are $[x_i^2 \;\; y_i^2]$ & V are $[x_i \;\; y_i \;\; 1]$.

$$We = We 1_{2\times 1} + WVP$$

$$O(P) = e^T W e = 1_{1\times 2} U^T W U 1_{2\times 1} + P^T V^T W V P + 2 P^T V^T W U 1_{2\times 1}.$$

$$\frac{\partial O(P)}{\partial P} = 2P^T V^T W V + 2 1_{1\times 2} U^T W V$$

$$\Rightarrow P^T = -1_{1\times 2} U^T W V (V^T W V)^{-1}$$

$$\Rightarrow \boxed{P = -(V^T W V)^{-1} V^T W U 1_{2\times 1} = \begin{bmatrix} \vec{a} \\ b \end{bmatrix}}$$

5 Robust fitting

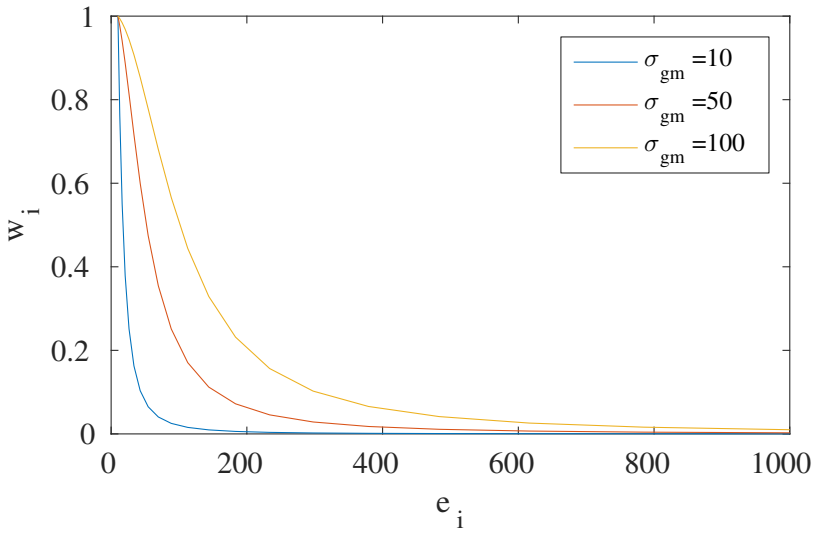
Continued:

Now I used the closed form formula for robustly fitting the circles:

1. Initialize $P = [-2\vec{x}_c; \|\vec{x}_c\|^2 - r^2]$
2. Update weights as $w_i = \frac{\psi(e_i)}{e_i} = \frac{1}{e_k}\frac{2\sigma_{gm}^2 e_k}{\sigma_{gm}^2+e_k^2} = \frac{2\sigma_{gm}^2}{\sigma_{gm}^2+e_k^2}$
3. Make diagonal matrix W with w_i along diagonal
4. Find leverage point and make their w_is zero (described below)
5. Solve linear system $AP_{\text{new}} = B$ where $A = V^T WV$ and $B = -V^T WU1_{2\times 1}$
6. Compute $\epsilon = \|P_{\text{new}} - P\|$
7. Update $P = P_{\text{new}}$
8. Repeat 2-7 until $\epsilon < 0.01$ or for niter=1000 times

To find the leverage points, I projected the high weight (i.e., above 0.85 quantile of the weight distribution) points on to a circle at $\vec{x}_c$ center with r radius. Next, I changed the coordinates to polar assuming $\vec{x}_c$ is the origin. Each weight w_i now has an associated θ_i. I sorted θ_is and computed the difference $\delta\theta_i = \theta_i - \theta_{i-1}$. I identified gaps in $\delta\theta_i$ distribution by picking $\delta\theta_i$s that are above 2σ (tuned experimentally) from the median of $\delta\theta_i$s. Inliers are then the largest set of points with no significant gaps in between. The leverage points are all the other high weight points that are not in the inlier set of points.

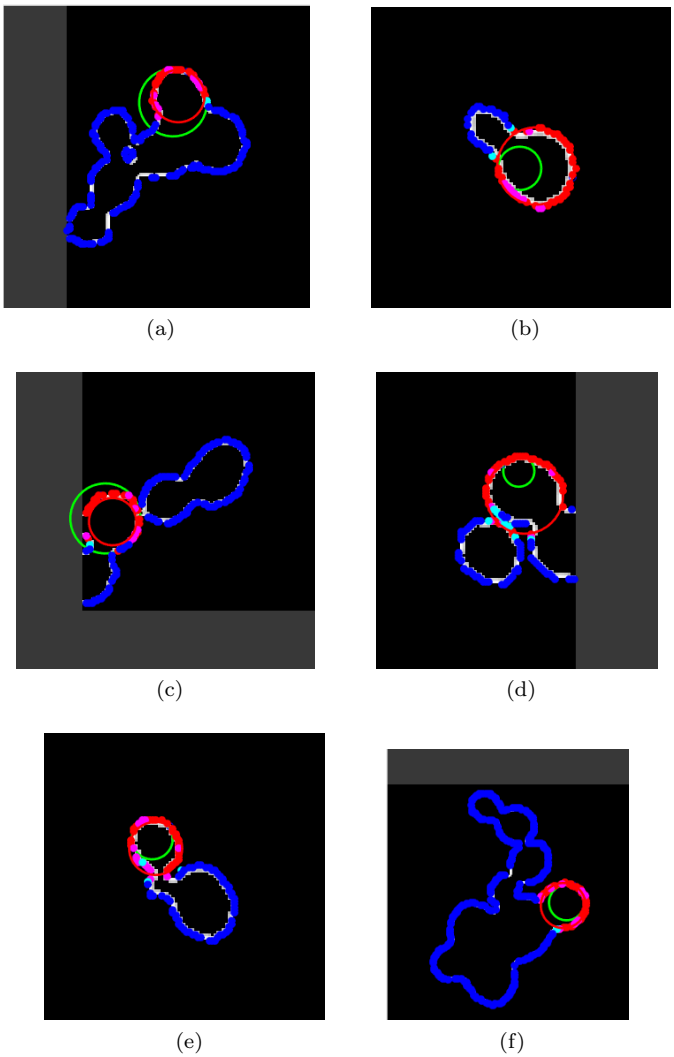
For picking the best σ_{gm}, I first plotted the weight versus error curve for different values of σ_{gm} as below:



This shows that for larger σ_{gm}s, we have higher tolerance for the points with high errors, while for smaller σ_{gm} the weights rapidly approach zero. I realized

that the optimal σ_{gm} is different for each set of edgels, depending on the level of variance of the inliers for each circle. Moreover, when σ_{gm} is very small such as 10, only the points with the lowest errors have significant weight. This means that the fitting is done based on only a small subset of inliers, which usually give small circles with poor fit. For $\sigma_{\text{gm}} \to 0$ all weights would be zero. Similarly, for high σ_{gm}, the weights are large and in particular, for $\sigma_{\text{gm}} \to \infty$, the weights would be all one, and the estimator would act like a normal LS estimator. I tested my code with σ_{gm} in range of [1,200] and found that $\sigma_{\text{gm}} = 24$ give reasonable results for most of the edgels.

Here are some of my results:



(a) (b) (c) (d) (e) (f)

Figure 3: A sample of fitted circles

6 Model update

For evaluating the performance of my code, I first set the output of isGoodCircle to true. I realized that in all cases, the code starts with a good a circle, however problems arise when the code attempts to fit the leftover points from previous runs. Specifically, the circles are often overlapping or even contained in other circles. While we want to remove many of these cases, the code must allow for some overlapping to happen, because 1) the boundaries between cells are not always clear and 2) there are buds from recently born cells that are splitting from another cell and thus overlapping with the parent cell. Here is my algorithm:

1. if nFound==0, return true
2. For each previous circle "CE" in the CircleEstimates:
3. Compute the distance d between the center of the proposed circle and CE
4. if $d \geq r + r_{CE}$, continue to the next CircleEstimate as these two circles are not overlapping
5. if $d < ||r - r_{CE}||$ return false because the circles are contained in each other
6. Compute the intersection surface of two circles A
7. if ($\frac{A}{\pi r^2} > 0.38$ or $\frac{A}{\pi r_{CE}^2} > 0.38$) and ($\frac{\pi r^2}{\pi r_{CE}^2} > 0.2$), return false
8. if the sum of overlap with all previous circles A is $\frac{A}{\pi r^2} > 0.4$ return false
9. else return true

I always return true for the first proposed circle as my it is a good fit in all of my test cases. This is because my bestproposal script returns a reasonable circle based on the available points.

In Step 7 of the above algorithm, the code returns false when the surface overlap between the proposed circle and any CE is above 38% ***and*** the surfaces ratio of the proposed circle to CE is larger than 20%. This ensures that it is fine for very small buds to have overlaps larger that 38% (I implicitly assumed that the buds are fitted after the main components of the blob which holds true for the way that I implemented my whole fitting algorithm).

Finally in Step 8, I check if the sum of the overlap with ***all*** CEs is less than 40% compared to the surface of the proposed circle. This condition ensures that we will not propose a circle to fit residual points surrounded by already fitted cells.

I tuned all these percentages and thresholds experimentally for the datasets we have here.

I used this[3] MATLAB function for computing the intersection surfaces in Steps 6 and 8 of the algorithm above.

[3] `https://www.mathworks.com/matlabcentral/fileexchange/15899-analytical-intersection-area-between-two-circles`

7 Brief evaluation

Important Note: I get a memory error when I run my code on CDF with GUI! Please run it without a MATLAB GUI using:“ matlab -nodisplay -r findCellScript ” command. I don’t have any issues on personal laptop though...

I extensively tested the above algorithm on the images provide to us. While $\sigma_{gm} = 24$, mentioned is Section 5, works well with the images 1,2, and 4, for image 3 it fails to provide good results. This is because in this figure there is a very large blob of cells for which we need to fit large circles, however small values of σ_{gm} tend to ignore the valid data points far away. Therefore for image 3 only, I increased σ_{gm} to 45. Note that my results are not deterministic and you may get slightly different results each time you run the code. Here are my fitted circles to four images:

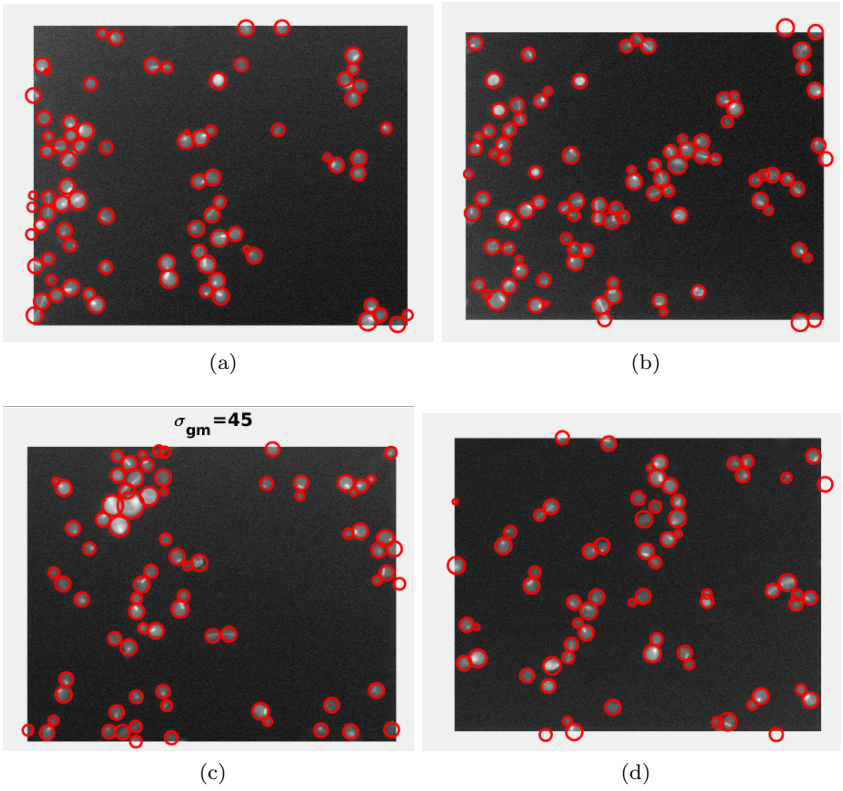


Figure 4: Fitted circles

As you can see above, the overall performance of the code is pretty good. Below are some cases that the performance is good. I show two cases with many cells packed in one edgel that were detected reasonably well as well as two cropped edgels that the code successfully fitted. Moreover, I display two cases where very small buds were captured by the code.

I also found small issues in the code which are depicted in Figure 6. This figure shows that in some cases (d and e panels) I have missed small buds, perhaps due to ill-treating leverage points. My code failed to fit some of the

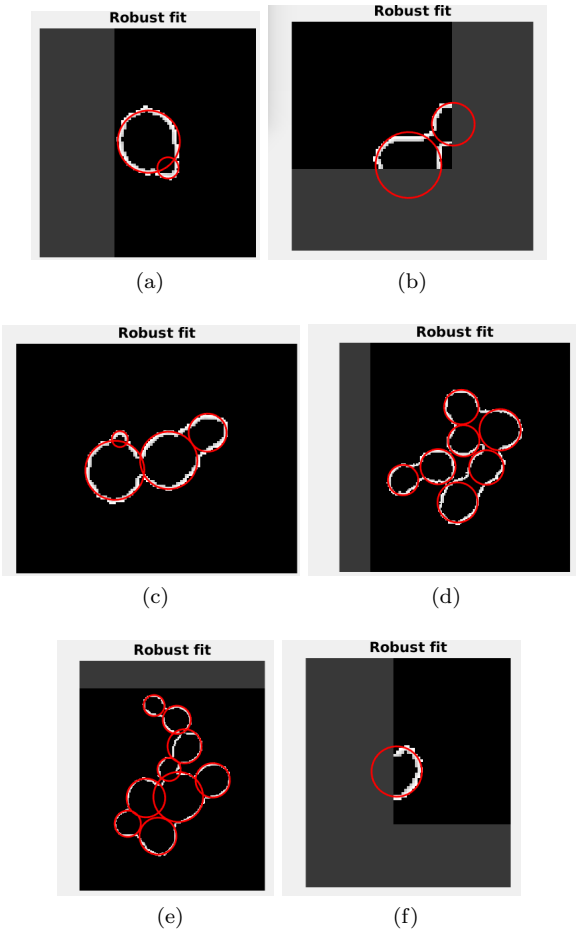
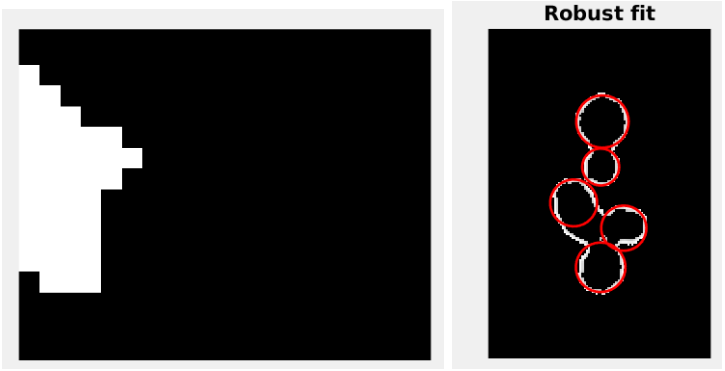


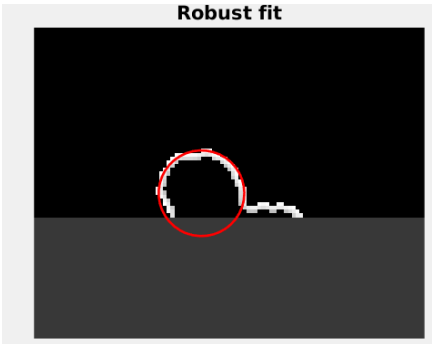
Figure 5: Well-fitted circles

greatly-cropped cells (a, c, and f panels) and several cells that are clearly not well represented by a circle (b and g panels). For missed buds, I believe I could increase the performance by improving the leverage point detection algorithm because some of the weights from larger cells must be excluded for fitting those buds well (see both d and e). For issues related to cropped images and cells not being circular, we have to move to fitting ellipses, because highly elliptical cells and lines from cropped images could be detected in that case.

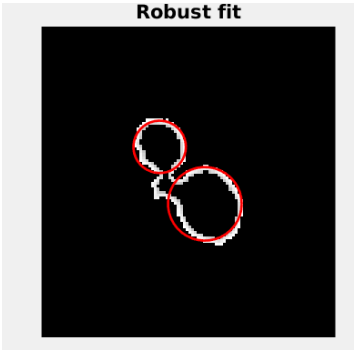


(a)

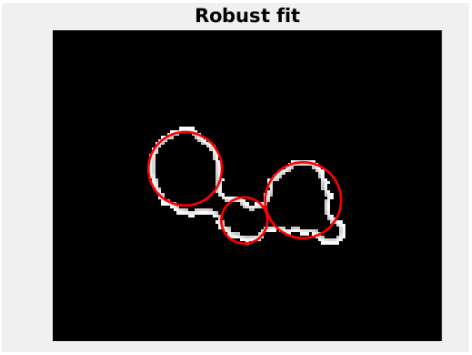
(b)



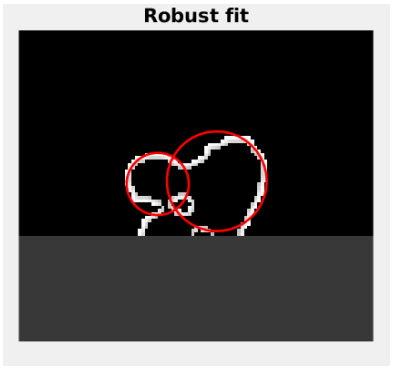
(c)



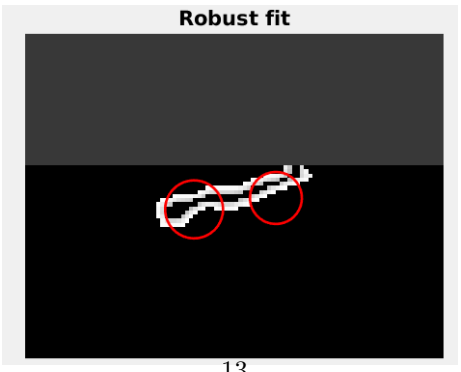
(d)



(e)



(f)



(g)

Figure 6: Poorly fitted cases