UoT CSC2503 - Foundations of Computer Vision Fall 2014

Assignment 2: Robust Estimation

Jia Ming Liang
October 28, 2014

1 Noise Model

Given that $\vec{m} \sim N(0, \sigma^2 I)$ is an isotropic Gaussian noise , we know that $\vec{x}_k$ is also isotropic Gaussian because $\vec{x}_k = \vec{m} + \hat{\vec{x}}_k$ and $\hat{\vec{x}}_k$ is just a constant. This constant changes the mean so $\vec{x}_k \sim N(\hat{\vec{x}}_k, \sigma^2 I)$. The variance of this distribution remains unchanged. Let $\vec{z}_k = \vec{x}_k - \vec{x}_c$ then $\vec{z}_k$ is also isotropic Gaussian because $\vec{x}_c$ is also a constant. Hence, $\vec{z}_k \sim N(\hat{\vec{x}}_k - \vec{x}_c, \sigma^2 I)$. Again, variance stays the same. Since $\vec{z}_k$ is a vector, $||\vec{z}_k||^2$ would be equal to $z_{xk}^2 + z_{yk}^2$.

Since $e_k = ||\vec{z}_k||^2 - r^2 = z_{xk}^2 + z_{yk}^2 - r^2$ and both z_{xk}^2 and z_{yk}^2 has the Gaussian distribution (with none zero mean) square then e_k satisfies a **non-central chi-square distribution with 2 degrees of freedom**. The non-zero parameter λ is defined to be $\sum_{i=1}^{n} (\frac{\mu_i}{\sigma_i})^2$. The mean of the non-central chi-square distribution is defined to be k+λ where k is the degrees of freedom. And variance is $2(k + 2\lambda)$. Since $-r^2$ is just a constant, the mean of e_k becomes $k + \lambda - r^2$. Finally plugging everyhing in:

Mean:

$$E[e_k] = k + \lambda - r^2 = 2 + \frac{(\hat{x}_{kx} - x_{cx})^2}{\sigma^2 I} + \frac{(\hat{x}_{ky} - x_{cy})^2}{\sigma^2 I} - r^2 \tag{1.1}$$

$$E[e_k] = 2 + \frac{r^2}{\sigma^2 I} - r^2 \tag{1.2}$$

because $(\hat{x}_{kx} - x_{cx})^2 + (\hat{x}_{ky} - x_{cy})^2 = r^2$, so λ is just $\frac{r^2}{\sigma^2 I}$

and Variance:

$$var[e_k] = 2(k+2\lambda) = 4(1+(\frac{r^2}{\sigma^2 I})) \tag{1.3}$$

Now we see that e_k is independent of $\vec{x}_c$, but r contribute to both mean and variance.

2 LS estimator

If we approximate the distribution of e_k to be mean-zero Gaussian noise. A reasonable choice for the variance of the Gaussian noise would be the variance of the error σ_g. From question one, the e_k variance is $4(1+(\frac{r^2}{\sigma^2 I}))$.

To derive a least square fit, use the joint log likelihood function

$$L(e_k) = \prod_{e_k \in X} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{e_k^2}{2\sigma^2}} \tag{2.1}$$

We then take the negative log likelihood:

$$-ln(L(e_k)) = -ln(\prod_{e_k \in X} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{e_k^2}{2\sigma^2}}) \tag{2.2}$$

$$-ln(L(e_k)) = ln(\sigma\sqrt{2\pi}) + \sum_{k=1}^{k} \frac{e_k^2}{2\sigma^2} \tag{2.3}$$

To maximize the likelihood we minimize the negative log likelihood. To do this we take $\frac{\partial - ln(L(e_k))}{\partial P} = 0$ where P is the parameters we want to find (in our case, a and b in e_k):

$$0 = \frac{\partial - ln(L(e_k))}{\partial P} = \sum_{k=1}^{k} \frac{\partial}{\partial P} \frac{e_k^2}{2\sigma^2} \tag{2.4}$$

$$0 = \sum_{k=1}^{k} \frac{\partial}{\partial P} (\vec{a}^T \vec{x}_k + b + \vec{x}_k^T \vec{x}_k)^2 \tag{2.5}$$

$$0 = \sum_{k=1}^{k} \frac{\partial}{\partial P} (\vec{a}^T \vec{x}_k + b + \vec{x}_k^T \vec{x}_k)^2 \tag{2.6}$$

$$0 = \sum_{k=1}^{k} \frac{\partial}{\partial P} ([\vec{x}_k \ \ 1][\vec{a} \ \ b]^T + \vec{x}_k^T \vec{x}_k)^2 \tag{2.7}$$

Setting P = $[\vec{a} \;\; b]^T$

$$0 = \sum_{k=1}^{k} 2[\vec{x}_k \;\; 1]([\vec{x}_k \;\; 1]P + \vec{x}_k^T \vec{x}_k) \tag{2.8}$$

Let A be K X 3 matrix where each row is $[\vec{x}_k \;\; 1]$ and B be K X 1 matrix where each row is the result of $\vec{x}_k^T \vec{x}_k$. We then use least square to solve for P.

$$AP = -B \tag{2.9}$$

3 CIRCLE PROPOSALS

To generate circle proposal I used the hough transformation algorithm. The reason why I choose to use this algorithm is because it easy to implement, it is bounded, it handles gaps between discontinuous data (missing edges for circle), and it is not very sensitive to noise. And more importantly, for simple objects like circle the computation time is very small. RANSAC on the other hand is not bounded, and there is a chance where optimal solution is not proposed (unless we run every single possibilities which is not very feasible for computation time).

Below is the description of my algorithm:

1. Initialize the hough space accumulator array with 0 (with parameters x_c, y_c and r). This is a 3 dimensional array with size xmas by ymax by rmax, where xmas and ymax is the max x and max y in the edgels position and rmax is the maximum radius which I set to the diagonal sample edgels (see figure 3.1).

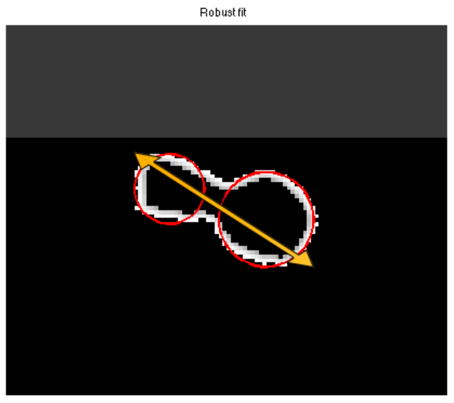

Figure 3.1: Yellow line show the diagonal (rmax) of the sample data

2. For each edgel (x, y) we do the following:

   a) compute the angle using the edgel normal (nx,ny) using arctan(ny/nx). Depending on which quadrant the normal is pointing toward we add or subtract π accordingly to find the correct angle.

   b)
   ```
   for r=rmin: rmax
       x0 = x - r*cos(angle)
       y0 = y - r*sin(angle)
       accumulator(x0, y0, r) = accumulator(x0, y0, r) + 1
   end
   ```

3. Find the local maxima of the accumulator and use that as the proposal circle center x_c, y_c and radius r.
   Below are some note on finding the local maxima

   a) we can set the bin size [width height] to reduce the resolution to find a clearer peaks by adding up the peaks in the bin size area and set the center of that area as the location of the peak. To large bin size will reduce resolution but too little we will see too many peaks. My bin size is 2x2 pixels see figure 3.2 for results. For some images we need to change the bin size to 4x4 to get better results especially for images with larger circles, then smaller bin size will have too much noise.

   b) set threshold for peaks so that we can find up to numGuesses peaks. We dynamically change this threshold to avoid infinite loop (not enough peaks) and over detect (too many peaks).

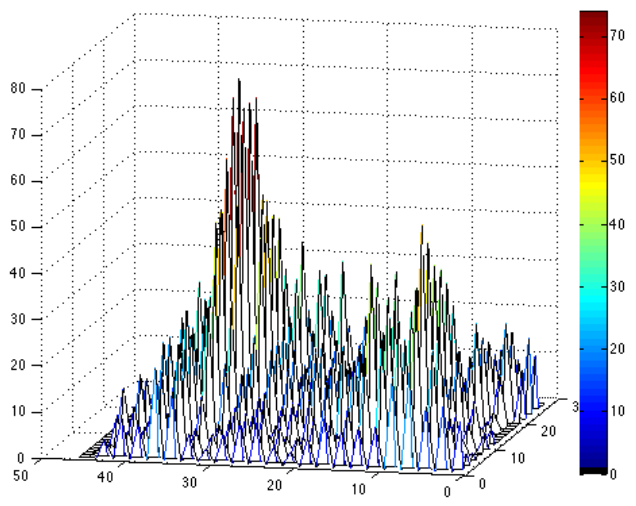

Figure 3.2: Sum of r dimension accumulator(x,y) peaks plot (detected for figure 3.1)

4 Circle selection

Given K proposed circles, I choose the circle with the smallest error when summing up the error function. In our case we use the Geman-Mclure (GM) estimator, $\rho(e,\sigma_g) = \frac{e^2}{\sigma_g^2+e^2}$.

My algorithm:

1. For every circle sum up $\rho(e_k,\sigma_g)$ for every edgels
2. Choose the circle with smallest sum of $\rho(e,\sigma_g)$ as the

The reason to choose this circle as the best circle because the best circle should have the smallest error in all the proposed circles. This circle is more likely to have more points lying on the circles. We then can eliminate these points in later steps since these points are most likely to be outlier for other circles.

5 Robust Fitting

We use best circle parameters $\vec{x}_c$ and r as initial parameters to IRLS to do robust fitting. To derive an IRLS algorithm for estimating circle parameters ($\vec{a}$,b), we minize the objective function $O(\vec{a},b) = \sum_k \rho(e_k(\vec{a},b),\sigma_g)$. To do that, simply taking $\partial O(\vec{a},b)/\partial P = 0$ where P is the vector $[\vec{a}\ \ b]^T$

$$0 = \frac{\partial O(\vec{a}, b)}{\partial P} = \sum_k \frac{\partial \rho(e_k(\vec{a}, b), \sigma_g)}{\partial P} \tag{5.1}$$

$$0 = \sum_k \frac{\partial \rho(e_k(\vec{a}, b), \sigma_g)}{\partial e_k} \frac{\partial e_k}{\partial P} \tag{5.2}$$

We can first find $\frac{\partial \rho(e,\sigma_g)}{\partial e}$:

$$\frac{\partial \rho(e, \sigma_g)}{\partial e} = \frac{\partial}{\partial e} \frac{e^2}{\sigma_g^2 + e^2} \tag{5.3}$$

$$\frac{\partial \rho(e, \sigma_g)}{\partial e} = 2e_k(\frac{\sigma_g}{\sigma_g^2 + e^2})^2 \tag{5.4}$$

We let

$$w(e) = (\frac{\sigma_g}{\sigma_g^2 + e^2})^2 \tag{5.5}$$

then

$$\frac{\partial \rho(e_k, \sigma_g)}{\partial e_k} = 2 \cdot e_k \cdot w(e_k) \tag{5.6}$$

We then find $\frac{\partial e_k}{\partial P}$. First we do some variable subsitition to e_k

$$e_k = \vec{a}^T \vec{x}_k + b + \vec{x}_k^T \vec{x}_k \tag{5.7}$$

$$e_k = [\vec{x}_k \;\; 1][\vec{a} \;\; b]^T + \vec{x}_k^T \vec{x}_k \tag{5.8}$$

Let P = $[\vec{a} \;\; b]^T$

$$e_k = [\vec{x}_k \;\; 1]P + \vec{x}_k^T \vec{x}_k \tag{5.9}$$

And

$$\frac{\partial e_k}{\partial P} = [\vec{x}_k \ \ 1] \tag{5.10}$$

Plugging everything back into equation 5.2

$$0 = \sum_k e_k \cdot w(e_k) \cdot [\vec{x}_k \ \ 1] \tag{5.11}$$

$$0 = \sum_k w(e_k) \cdot ([\vec{x}_k \ \ 1]P + \vec{x}_k^T \vec{x}_k) \cdot [\vec{x}_k \ \ 1] \tag{5.12}$$

$$\sum_k w(e_k) \cdot [\vec{x}_k \ \ 1]P \cdot [\vec{x}_k \ \ 1] = -\sum_k w(e_k) \cdot \vec{x}_k^T \vec{x}_k \cdot [\vec{x}_k \ \ 1] \tag{5.13}$$

We can rewrite the above expression in matrix form. First we let W be a K × K matrix with $w(e_k)$ in the diagonal. Let A be K × 3 matrix where each row is $[x_{kx} \ \ x_{ky} \ \ 1]$ and let B be K × 1 matrix with $\vec{x}_k^T \vec{x}_k$ each row. Finally, we have our IRLS function:

$$(A^T \ \ W \ \ A)P = -A^T \ \ W \ \ B \tag{5.14}$$

We then solve the above using least square method to get P. P will contain $\vec{a}$ and b. W then find $\vec{x}_c = -\vec{a}/2$ and $r = \sqrt{b - \vec{x}_c^T \vec{x}_c}$.

Algorithm:

1. Use initial proposal parameters to form matrix A, W, and B in equation 5.14
2. Solve equation 5.14 to get P
3. Using the P solved in step 2. as new parameters to iterate step 1. and 2. again
4. we loop step 1. to 3. to maximum 30 times in case it does not converge, and we break the loop after 4 initial iterations if we detect the difference between the new calculated P and the previous P is smaller than 0.5 to speed up computation time because we are dealing with pixels so we do not need accuracy below an integer value

To σ_g I first set it to 1 then increase by a factor of 10. Below are the sample fit for σ_g = 1, 10, 100, and 1000:

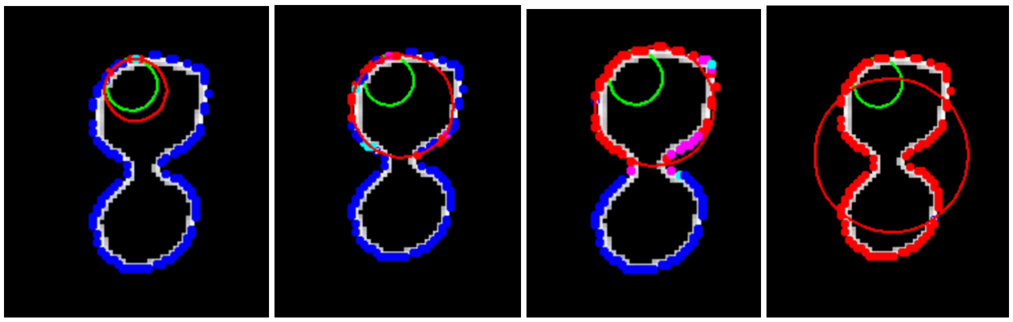

Figure 5.1: From left to right:σ_g = 1, 10, 100, and 1000

From image 5.1 we can see that smaller σ_g tends to produce smaller circle because as σ_g become very small our ρ function becomes just 1 which means further points give no weight on the error hence the fitting doesn't seem to change much from the original proposal. If σ_g is very big we can see that the robust fitting will try to cover every points because variance so big that the error tolerate points very far away. We can see that σ_g between 10 and 100 seems to give better results. Further narrowing down σ_g I found σ_g=35 (to 40) give the best overall result.

6 MODEL UPDATE

To decide whether a robust fitted circle is a good circle I use the following criteria:

1. **Using the weight:** If nFound = 0 (i.e. the first circle to be found for the current working set of edgels), we sum up the weight for all edgels then we check whether this weight pass some threshold. We do this for the first circle because the initial circle has most edgels to use for calculating weight. After we remove these edgels the future circles will have less edgels so the sum of weight will tend to be smaller than the threshold even if they fit those points.

   To find the threshold, we use the circumference of the fitted circle $c = 2\pi r$. We know that the edgels could be discontinuous and the weight on each edgels are less than 1 so we set the threshold $T = \alpha c$ where α is a constant that I manually set to allow some tolorant for the fitted circle. After testing my program I found that $\alpha = 0.2$ would give the most optimal result. It will tolerate some slightly elliptical cell too (see figure 6.1).

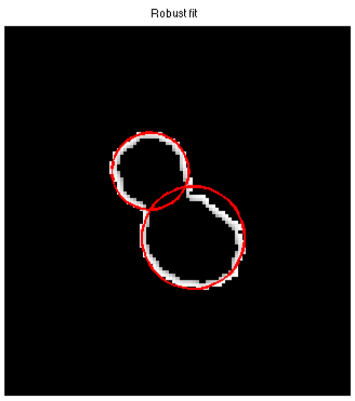

Figure 6.1: Fitting on slightly elliptical shape

Althought it was not asked for in our assignment, I actually set T = $\alpha\frac{2*\pi - croppedAngle}{2*\pi}c$ to tolerate circle fitting that are cut by the boundary of the image (see figure 6.2). To find croppedAngle, I hardcoded the dimension of the image and I use the center point and radius to determine the total angle that is cropped by the edge of the image. As a result, I even found circles that are at the side of the image.

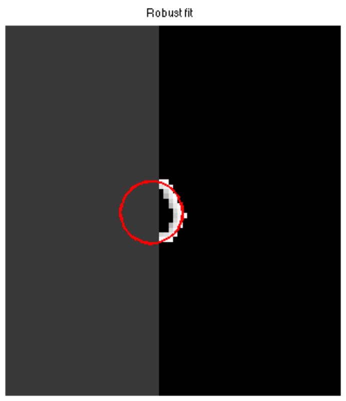

Figure 6.2: Fitting on significantly cropped circles

2. **Prevent intersecting circles:** if nFound > 0, we will check whether the circle intersect with previously found circles. If two circles are intersecting, the distance between two circles

is smaller than the total radius of the two circles. We do allow some intersecting as some cell and budging out from another we should count them as two cells instead of one. After some testing, I set the the intersecting angle to be within π as I would like to capture the small buds on some cells. If multiple circle are intersecting I just add the total intersecting angle together.

If circles totally overlapping on each other, we cannot calculate the intersecting angle. To avoid that we simply make sure the distance between the two circles is larger than the largest radius of the two circles.

To calculate the intersecting angle I used the following formulars:

$$x = \frac{d^2 - r^2 + R^2}{2d} \tag{6.1}$$

$$intersectingAngle = 2 * arcsin(\frac{x}{r}) \tag{6.2}$$

Note: see figure 6.3

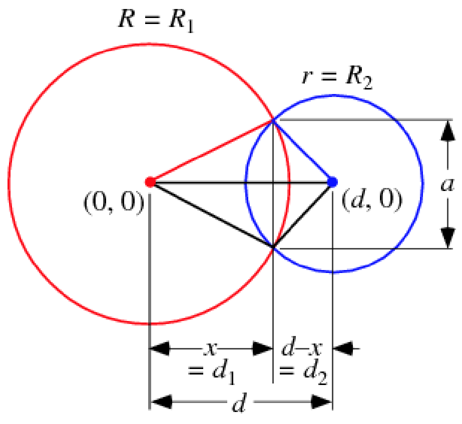

Figure 6.3: Circle intersection figure (reference from [2])

7 Brief evaluation

Below are my results for running all 4 images

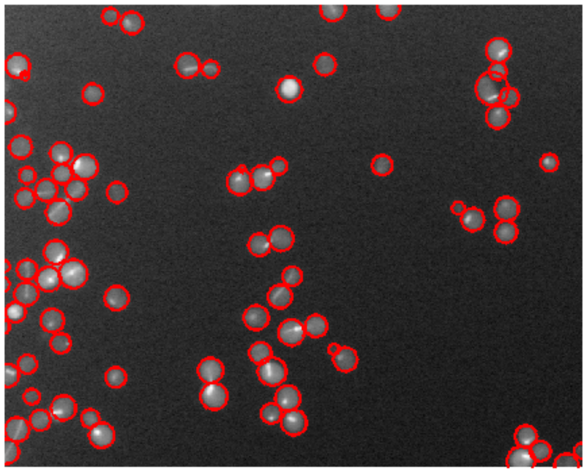

Figure 7.1: Results set cellGFP2a_00

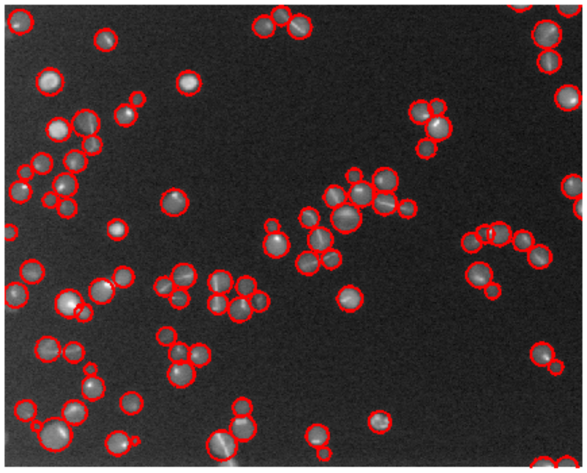

Figure 7.2: Results set cellGFP2a_10

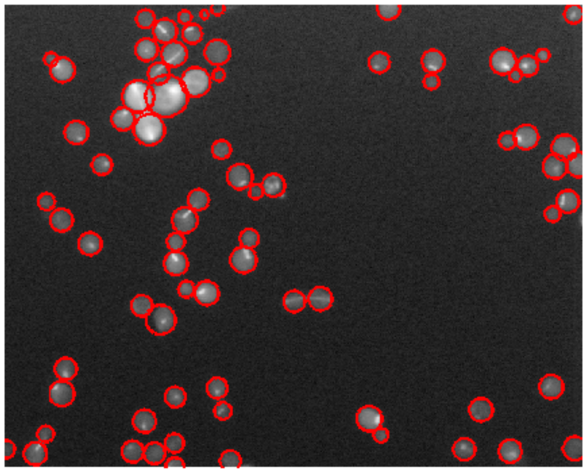

Figure 7.3: Results set cellGFP2a_01

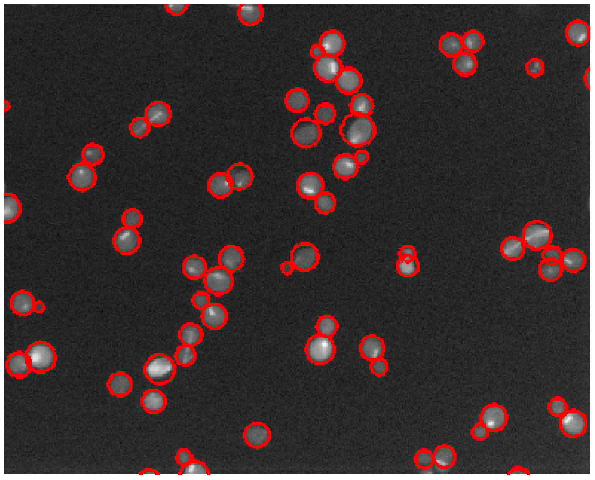

Figure 7.4: Results set cellGFP2a_11

My algorithm did a pretty good job finding all the cells (with small twist in sigma and bin size for each results). However, there are still some cell at the edge of the image and some buds are unable to be detected. Besides these two "failure modes" I notice that the most obvious error is associated with elliptical shape.

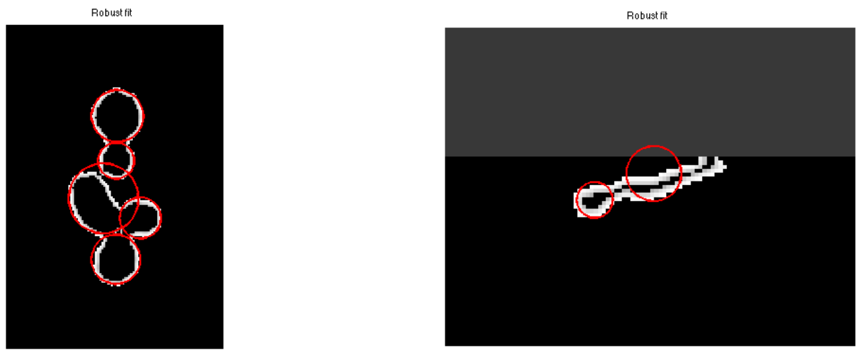

Figure 7.5: Elliptical shape fit

As we can see from figure 7.5, the elliptical shape have a huge error (the circle tends to over estimate. To fix this, I can change the circle model to elliptical shape model which we will have more parameters to find. Hough transform (or use RANSAC) should still be fine to do the job but it probably require more computation time.

References

[1] http://en.wikipedia.org/wiki/Noncentral_chi-squared_$r$distribution

[2] http://mathworld.wolfram.com/Circle-CircleIntersection.html