

Femur Head and Neck Angle Measurement

A. OBJECTIVE

The objective of the project is to create a program that reads in a binary image file of a femur, fits circles through the femur head and femur neck, and automatically measures the angle between the line connecting the center of the femur head circle and the center of the femur neck circle, and the line that is vertical through the center of the femur head circle when the bone lies horizontally with the head to the right, like in Figure 1.

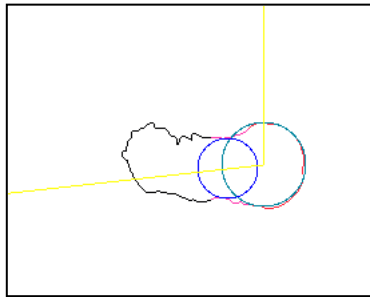


Figure 1: Angle Representation - Horizontal

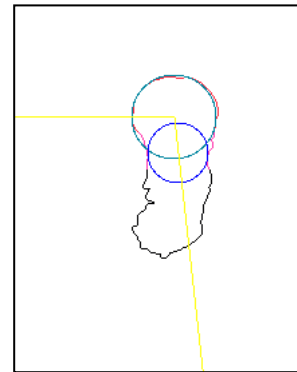


Figure 2: Angle Representation - Vertical

This angle is equivalent to the angle between the center of the femur head circle and the center of the femur neck circle, and the horizontal line through the center of the femur head circle, when the bone lies vertically with its head upwards, like in Figure 2. The angle we are calculating is angle PAB in Figure 3.

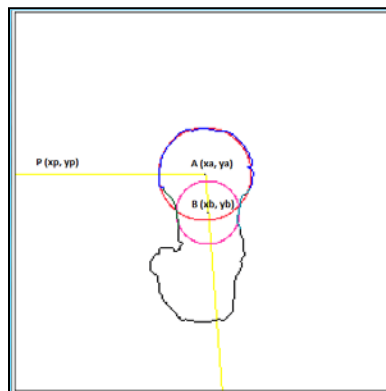


Figure 3: The program calculates angle PAB

The most important requirement for the program, apart from fitting circles accurately, is that the circle fits and angle measurements should be consistent across independent executions of the program.

B. Graphical User Interface, Input and Output

The input for the program is a binary image, containing the white bone on a black background. The image is in Analyze 7.5 format (though the program supports a variety of formats).

The output of the program is a modified image, with only the contours of the bone, two circles and lines to represent the angle that is calculated, as well as the size of the angle in both degrees and radians.

The program is implemented in Java, as a plug-in for ImageJ. The GUI for the program makes use of the Action Bar plug-in (Mutterer). The GUI consists of a menu bar that has 7 buttons, separated into 3 groups (Figure 4).



Figure 4: The Graphical User Interface

The last group contains a single button that turns the ImageJ interface on and off and provides a more stand-alone look and feel to the program.

The penultimate group has 4 buttons: zoom in and zoom out, and rotate left and rotate right (90^0). The zoom buttons are provided since the image tends to look pretty small even at full resolution. The rotate buttons are provided because the program assumes the images are originally vertically aligned with the femur head on top, and the user has to make sure that is the case before running the program. It is my understanding that images come out of the machine orientated this way in the first place, so this step is actually not necessary in normal cases.

The first group contains two buttons that are essential to our program. The first button of the group opens the binary image and converts it to an RGB one. It also sets the color map such that the brightness/contrast levels are adjusted appropriately in order to see the white bone (opening the image normally, using File>Open, opens a black image), as well as sets the size of

the pixels to 1 (not necessary, but it makes the program consistent). We need this image to be converted to RGB since we will draw colored circles and lines on it. Using this button the user can read in a variety of image formats, per ImageJ's capabilities (Figure 5). It is nevertheless up to the user to make sure that the image he reads in is binary, and to rotate the image appropriately before pressing the run button.

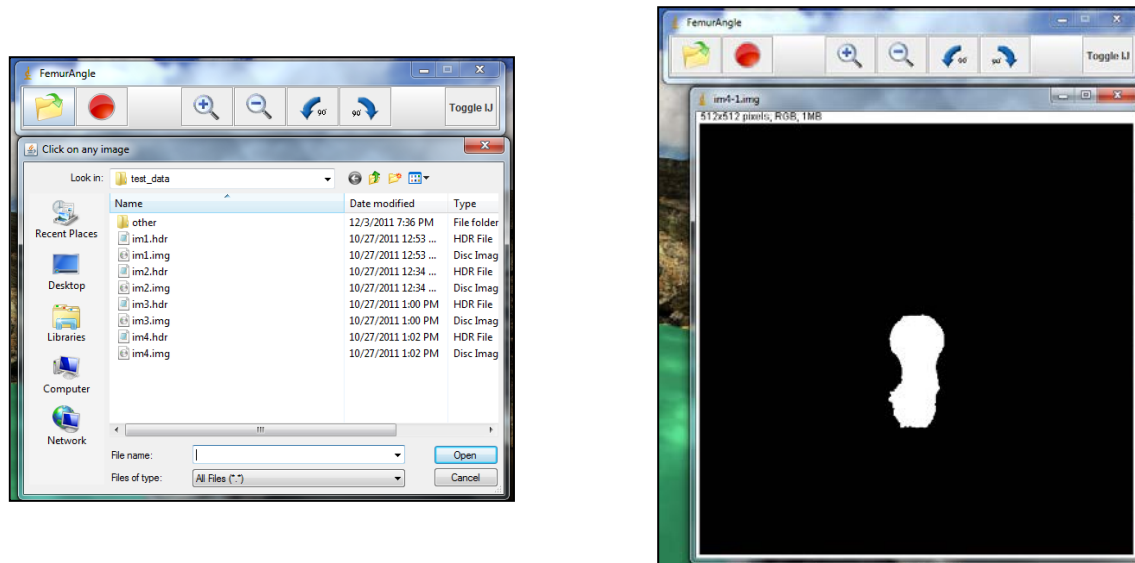


Figure 5: Open Files Button

The second button in this group is the button that executes the code (the “run” button). It finds the boundary points of the bone, keeps the first third of them as the points to do circle fitting for the femur head, separates the first half of the second third into two parallel sets to fit the second circle, applies the Kasa circle fitting algorithm on the first set to find the femur head circle and computes the closest points on the other separated set to find the femur neck circle. It then calculates the angle and draws all the borders, circles and angle lines. It also pops-up a dialog with the angle measurements in degrees and radians (up to 4 decimals), in a format that is easy to copy (Figure 6).

The mathematics behind the Kasa algorithm is described in detail in section C. It is worth mentioning here, though, that, by the algebraic nature of the solution, the program is consistent in placing the circles when run independently on the same data. The circle fit is also visually (and algebraically) correct given the nature of our data (i.e. the fact that the points on the femur head describe an arc that covers more than 180° and that they lie almost on a circle).

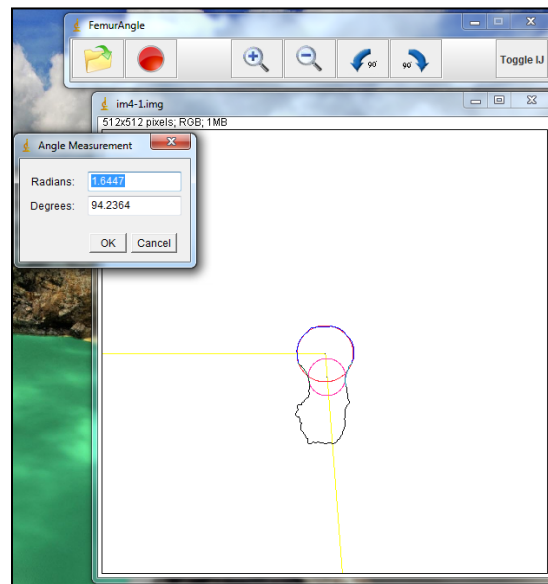


Figure 6: Program Processing and Output

This is of course true under the assumptions that the femur head and neck are accurately represented by the points that the fitting is done through, and a completely disproportionate femur would pose problems because of the point selection assumption: the algorithm would still be consistent, but not correct in such cases (see section F for problems and section G for possible solutions and future work).

Figure 7 gives the program overview.

- ⦿ Read in a binary image file in Analyze 7.5 format, representing the femur
- ⦿ User makes sure the bone lies vertically with femur head up
- ⦿ Fit circle through femur head with center A
- ⦿ Fit circle through femur neck with center B
- ⦿ Find angle between the line AB and the horizontal line through A

Figure 7: Program Interaction Overview

To install the program, the user has to install ImageJ and then copy the ActionBar folder provided into the plugins folder. To run the program, the user opens ImageJ, goes to the Plugins menu item, then to ActionBar > FemurAngle. Restart ImageJ if ActionBar doesn't originally show up. I have tested the program and it works on both Mac OS X and Windows 7, provided the appropriate version of ImageJ is installed.

C. Theory Preliminaries

1. Circle Equation

The equation of a circle of center (x_c, y_c) and radius R is, for all points (\bar{x}_i, \bar{y}_i) on the circle:

$$(\bar{x}_i - x_c)^2 + (\bar{y}_i - y_c)^2 = R^2. \quad [1]$$

2. Least Square Fitting For Circles

Least square fitting is a mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the square distance from the points on the fitting curve to the given data points. In other words, given n points (x_i, y_i) where $1 \leq i \leq n$, we need to minimize the cost function:

$$F = \sum_{i=1}^n d_i^2, \quad [2.1]$$

where d_i is the Euclidean distance from a point (x_i, y_i) in our data to the circle.

That is,

$$d_i = \sqrt{(x_i - \bar{x}_i)^2 + (y_i - \bar{y}_i)^2}, \quad [2.2]$$

where (x_i, y_i) are points in our data set and (\bar{x}_i, \bar{y}_i) are the points on the circle. This distance in [2.2] is equivalent to the distance from (x_i, y_i) to the center of the circle (x_c, y_c) minus the distance from the points on the circle (\bar{x}_i, \bar{y}_i) to the center of the circle (that is, the radius R).

So:

$$d_i = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} - R, \quad [2.3]$$

The minimization of the cost function [2.2] under the definition of the distance in [2.3] is a non-linear problem that has no closed-form solution. The possible solutions are either iterative and costly or approximate by nature.

The iterative solutions to the minimization of F problem are called geometric fits. Various methods have been developed to accomplish geometric fitting, including the Levenberg-Marquardt method (Levenberg, 1944) (Marquardt, 1963), the Landau algorithm (Landau, 1987) and the Spath algorithm (Spath, 1996). N. Chernov and C. Lesort also proposed an alternative to

the Levenberg-Marquardt method that is more stable across points covering small arcs (N. & Lesort, 2005). Although they theoretically produce a better visual fit, the geometric fitting methods are very expensive. On our data, the approximate methods tend to work quite as well.

The non-iterative solutions are called algebraic fits. Algebraic fitting provides a fast and non-iterative approximation to the LSF problem. There are many approaches to doing this fit, for example the methods by Pratt (Pratt, 1987), Taubin (Taubin, 1991) and Kasa (Kasa, 1976). Although there are problems with the Kasa fit for random sets of data, the problem we have renders itself to a good Kasa solution.

3. Algebraic Circle Fitting using the Kasa Method

The Kasa circle fitting method (Kasa, 1976) minimizes the sum of the squares of the algebraic distances between the points on the circle and the given data points. The cost function for the minimization problem is:

$$F_k(x_c, y_c, R) = \sum_{i=1}^n [(x_i - x_c)^2 + (y_i - y_c)^2 - R^2]^2 \quad [3.1]$$

This turns out to be a good approximation for samples of points that lie approximately on a circle (outliers tend to affect the fit gravely) and that cover large enough arcs of the circle (small arcs lead to a too small R). In the case of our femur head, we sample points that are on the boundaries of the femur head which means that they are approximately on a circle and that they cover an arc bigger than 180° .

The equation in [3.1] is equivalent to:

$$F_k(B, C, D) = \sum_{i=1}^n (z_i + B \cdot x_i + C \cdot y_i + D)^2, \quad [3.2]$$

where:

$$\begin{aligned} z_i &= x_i^2 + y_i^2 \\ B &= -2x_c \\ C &= -2y_c \\ D &= x_c^2 + y_c^2 - R^2 \end{aligned} \quad [3.3]$$

To minimize the cost function one needs to set the gradient to 0. If we partially differentiate $F_k(B, C, D)$ in [3.2] with respect to B , C and D respectively, we obtain the gradient of the function.

$$\begin{aligned}\nabla F_k(B, C, D) &= \left(\frac{\partial F_k}{\partial B}, \frac{\partial F_k}{\partial C}, \frac{\partial F_k}{\partial D} \right), \text{ where} \\ \frac{\partial F_k}{\partial B} &= 2 \left(\sum_i x_i z_i + B \sum_i x_i^2 + C \sum_i x_i y_i + D \sum_i x_i \right) \\ \frac{\partial F_k}{\partial C} &= 2 \left(\sum_i y_i z_i + B \sum_i x_i y_i + C \sum_i y_i^2 + D \sum_i y_i \right) \\ \frac{\partial F_k}{\partial D} &= 2 \left(\sum_i z_i + B \sum_i x_i + C \sum_i y_i + D \cdot n \right)\end{aligned}\tag{3.4}$$

By setting these to 0, we get

$$\begin{aligned}B \sum_i x_i^2 + C \sum_i x_i y_i + D \sum_i x_i &= - \sum_i x_i (x_i^2 + y_i^2) \\ B \sum_i x_i y_i + C \sum_i y_i^2 + D \sum_i y_i &= - \sum_i y_i (x_i^2 + y_i^2) \\ B \sum_i x_i + C \sum_i y_i + D \cdot n &= - \sum_i (x_i^2 + y_i^2)\end{aligned}\tag{3.5}$$

Define P_1 , P_2 and P , as:

$$P_1 = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{pmatrix}, \quad P_2 = \begin{pmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ \vdots \\ x_n^2 + y_n^2 \end{pmatrix} \quad \text{and} \quad P = \begin{pmatrix} B \\ C \\ D \end{pmatrix}\tag{3.6}$$

such that [3.5] is equivalent to:

$$P_1 \cdot P = P_2 \rightarrow P = P_1^{-1} \cdot P_2.\tag{3.7}$$

Solving [3.7] for P we find B , C and D which give us x_c , y_c and R according to [3.3].

D. Algorithms and Code

The program is implemented as a Java class, named FemurAngle, that implements the ImageJ PlugInFilter class. This means that the program runs on an ImagePlus object that contains an ImageProcessor. The ImageProcessor contains the pixel data (8-bit, 16-bit, float or RGB) of the 2D image and some basic methods to manipulate it. The FemurAngle program assumes the image is RGB (since opening it from the FemurAngle interface automatically converts it to RGB). In order for the program to run, an image has to be first loaded into ImageJ, or otherwise the user receives an error.

There are 15 methods in this class: two methods that have to exist in all ImageJ plug-ins, setup (constructor) and run (main method), as well as 13 methods that I wrote. Three of these

methods are used for border detection and selection (getting the border for the entire image and selecting the points on the border through which to fit the first circle and the second circle respectively). Three other methods are used to draw the contours, the circles and the lines for the angle. One method implements the Kasa circle fit described in C.3, another method implements the closest-points function for the diameter of the second circle, and another method implements the angle finder (as the slope between the horizontal and the line connecting the two circle centers). The rest of the methods are helper methods, for sorting arrays, removing duplicates in arrays, finding the centroid and calculating the distance between two points.

a. The Femur Head Circle: Kasa Circle Fit

The *fitCircle* method has as parameter an n-by-2 array, *points*, containing the x and y coordinates of the n points that form the contour of the femur head (1/3 of the number of border points). It then creates the n-by-3 array *p1*, whose first two columns are the same as the columns of *points* and whose last column contains all 1s, as well as the array *p2* which only has one column, $x_i^2 + y_i^2$. Then, *p1* and *p2* are converted into JAMA Matrix form (compatibility with the package is the reason why *p2* was declared n-by-1 in the first place).

Equation [\[3.8\]](#) is solved by a simple call in JAMA, $P = P1.inverse().times(P2)$. The radius and center of the circle is then found from the elements of *P* by solving equations [\[3.3\]](#).

The method returns an array of 3 elements, $centAndRad = \{x_c, y_c, R\}$. (Chernov)

b. The Femur Neck Circle: Closest Points in the Neck Region

The parameter of the *closestPoints* method is an m-by-2 array containing the x,y coordinates of the m points in the neck of the femur (1/6 of the number of border points). It then splits this array into two arrays, the array of the points situated on the left of the centroid of these points and the array of the points on the right. It then finds the minimum distance between the points on the left and the points on the right and saves the positions where this distance is minimized. These points describe the diameter of the circle, the center of the circle is the center of this point and the radius is the distance from any of the two points to the center.

The method returns an array of 3 elements, $centAndRad = \{x_c, y_c, R\}$.

c. Finding the Angle

The angle is the tangent of the slope. The *getAngle* method returns this angle, given the centers of the circles.

E. Results

On the examples that I was given, the program fits both circles very well (Figure 8). The method is also consistent across independent runs on the same data.

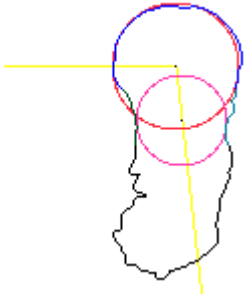
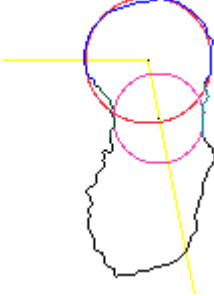
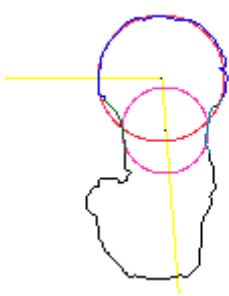
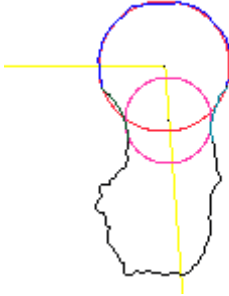
			
Radians: 1.6815 Degrees: 96.3402	Radians: 1.6476 Degrees: 94.3987	Radians: 1.7415 Degrees: 99.7824	Radians: 1.6447 Degrees: 94.2364

Figure 8: Visual Results

F. Problems

The circle fitting method (for the femur head) doesn't work well if the points through which we need to fit the head circle cover an arc that is smaller than 10 degrees. In such cases the radius can be gravely underestimated, leading to very small circles. If the data has pronounced outliers, the center point would be moved toward the outlier location. In general, this is not a problem for our data, given that femur heads don't usually have severe outliers, and the points through which we are trying to fit the data cover an arc bigger than 60 degrees.

Finding the closest points on both sides of the femur neck poses a problem if there are outliers that are closer than the points that would be picked by a biomedical specialist.

Both problems can be solved (partially) by a series of opening and/or closing operations, but since these operations can be applied a different number of times for best results, the results would not be consistent between different independent runs for the program.

Another problem is that the program only works if the image is binary, with a black background and a white bone, and if the femur is vertical and the femur head is up. It thus requires user input in order to make sure that these conditions are true. A better program would allow the user to binarize the image as well as automatically detect the orientation of the femur head.

One of the biggest problems of the program is the way it picks the points to fit the circles through. It makes an assumption that the first third of the points are representative of the femur head and the next sixth of the points are representative of the neck (coming left-right, top-bottom). A better heuristics for picking the points would be better for nonstandard bones.

G. Future Work

There are a few improvements that can be made to the program, giving the user more control. The only problem with giving the user control is that this fares for irreproducible results. One way of making the results reproducible would be to have the program write log files with the settings and allow the program to read and/or modify the log files.

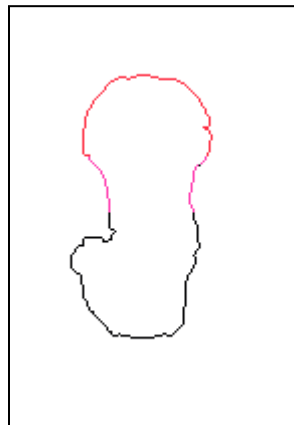


Figure 9: Head (red) and Neck (magenta) Regions

One of the improvements would allow the user to (visually) modify the fraction of points through which the fitting is done (red and magenta in Figure 9). This would ensure that the fitting is done through the appropriate points, but since a different user can choose a different number of points, the program loses its consistency.

Another improvement would be to allow the user to do any number of subsequent opening and closing operations. Opening and closing are already implemented in ImageJ so this

would just be a matter of adding buttons for that. Again, the biggest problem with this is the inconsistency in the results that would ensue. A button for binarizing the image can also be added. The user could also be asked for input with regard to the orientation of the image.

In terms of code improvements, a way to detect the femur head and neck would be ideal, as well as a better heuristics for deciding upon the points that comprise the head and neck.

References

- Kasa, I. (1976). A curve fitting procedure and its error analysis. *IEEE Trans. Inst. Meas.* 25 , 8-14.
- Landau, U. (1987). Estimation of a circular arc center and its radius. *Computer Vision, Graphics and Image Processing* 38 , 317-326.
- Levenberg, K. (1944). A Method for the Solution of Certain Non-linear Problems in Least Squares. *Quart. Appl. Math.* 2 , 164-168.
- Marquardt, D. (1963). An Algorithm for Least Squares Estimation of Nonlinear Parameters. *SIAM J. Appl. Math.* 11 , 431-441.
- Mutterer, J. (n.d.). *Action Bar Plugin*. Retrieved from http://imagejdocu.tudor.lu/doku.php?id=plugin:utilities:action_bar:start
- N., C., & Lesort, C. (2005). Least Squares Fitting of Circles. *Journal of Mathematical Imaging and Vision* 23 , 239-252.
- Pratt, V. (1987). Direct least-squares fitting of algebraic surfaces. *Computer Graphics* 21 , 145-152.
- Spath, H. (1996). Least-Squares Fitting By Circles. *Computing* 57 , 179-185.
- Taubin, G. (1991). Estimation Of Planar Curves, Surfaces And Nonplanar Space Curves Defined By Implicit Equations, With Applications To Edge And Range Image Segmentation. *EEE Transactions on Pattern Analysis and Machine Intelligence* 13 , 1115-1138.