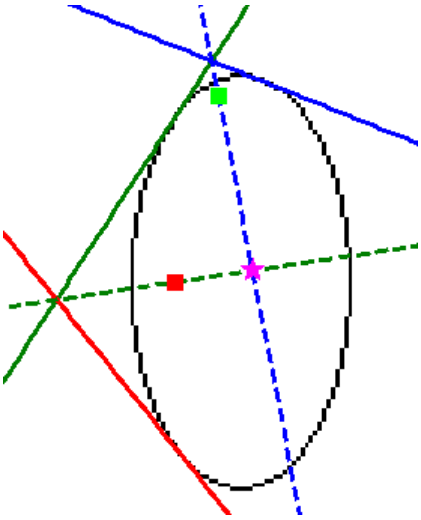
Ellipse Detection Using Randomized Hough Transform

Samuel Inverso

May 20, 2002



Abstract

This paper discusses the Randomized Hough Transform used to find ellipses in images. The equations to find ellipses and their implementation in the algorithm are explained. The algorithm performed well, finding ellipses of orientation 0^o and $9O^o$ from the x-axis in various images. Ellipses were also found in a real-world image after preprocessing.

Contents

1 Introduction

The Hough transform (HT) is a standard technique for detecting curves. The HT consists of three steps: 1. a pixel in the image is transformed into a parameterized curve, 2. valid curve's parameters are binned into an accumulator where the number of curves in a bin equals its score, and 3. a curve with a maximum score is selected from the accumulator to represent a curve in the image [5] [4].

This basic Hough Transform suffers from many difficulties stemming from binning the curves. The accumulator's bin sizes are determined by "windowing and sampling the parameter space in a heuristic way" [5]. To detect curves in a variety of images the window size must be large, and to detect curves with a high accuracy there must be a high parameter resolution. These two properties require a large accumulator and much processing time. Xu et. al. [5] identified possible problems that may occur if the accumulator is not properly defined. These are:

1. failure to detect some specific curves.
2. difficulties in finding local maxima
3. low accuracy
4. large storage
5. low speed

To reduce these problems Xu proposed a Randomized Hough transform (RHT). RHT randomly selects n pixels from an image and fits them to a parameterized curve. If the pixels fit within a tolerance they are added to an accumulator with a score. Once a specified number of pixel sets are selected the curves with the best score are selected from the accumulator and its parameters are used to represent a curve in the image. Because only a small random subset of pixels, n, are selected this method reduces the storage requirements and computational time needed to detect curves in an image. In RHT if a curve in the accumulator is similar to the curves being tested the parameters of the curves are averaged together and the new average curve replaces the curve in the accumulator. This reduces the difficulty of finding the local maxima in the hough space because only one point in the hough space represents a curve, instead of a clump of near points with a local maxima.

While Xu [5] convincingly argues the benefits of the Randomized Hough Transform, he does not give hard results to backup the theory. There are only two examples in his paper, one detecting lines and the other detecting circles.

As expected, both perform much better than the standard Hough transform. In addition, Xu states that RHT cannot be used for ”curves expressed by equations which are nonlinear with respect to parameters,” which includes ellipses. Robert McLaughlin [2] experimented with RHT and compared it against the standard HT and Probabilistic Hough Transform (PHT is similar to HT but only a small portion α of the pixels in the image, where $2\% < \alpha < 15\%$), are transformed.

McLaughlin achieved good results with RHT vs PHT and HT. He found RHT had ”higher accuracy that both HT and PHT, in noise free images with multiple ellipses” . Also, RHT was ”less subject to false alarms in both noise-free and noisy images. Moreover, RHT proved to be faster than either SHT or PHT ... [and required] substantially less memory” [2]. These results help verify Xu’s statements on RHT. However, contrary to Xu, McLaughlin performed all experiments with ellipses, which Xu states cannot be found by RHT. McLaughlin does this by deriving a linear equation for ellipses.

The author’s goal was to implement the Randomized Hough Transform described and implemented by McLaughlin [2]. In addition, Andrew Schuler’s implementation[3] of McLaughlin’s work served as a reference for this project. Both papers contain high level information, and a few equations, on parameterizing an ellipse and find it with a RHT, however, they leave many of the implementation details to the reader. This caused difficulty to the author in implementing the RHT because much research was required to determine the correct ellipse equations and their application to this problem. These equations and implementation are described below.

2 Algorithm

2.1 High Level Detail

The overall Randomized Hough Transform algorithm implemented is described below:

```
 1 while( we find ellipses OR not reached the maximum epoch ) {
 2    for( a fixed number of iterations  ) {
 3        Find a potential ellipse.
 4
 5        If( the ellipse is similar to an ellipse in the
 6        accumulator) average the two ellipses and replace the
 7        one in the accumulator. Add 1 to the score.
 8
 9        Else insert the ellipse into an empty position in the
10        accumulator with a score of 1.
11    }
12    Select the ellipse with the best score and save it in a
13        best ellipse table.
14    Remove the best ellipse's pixels from the image.
15    Clear the accumulator.
16 }
```

The algorithm executes for a number of epochs, where an epoch is the processing that occurs to find ellipses through accumulation. The algorithm completes when the maximum number of epochs is reached or it does not find ellipses for a specified number of epochs. This allows the user to specify a large epoch maximum and still not waste computing time if the algorithm stops finding ellipses.

The main body of processing occurs in the `for` loop starting on line 2. During the loop, ellipses found are accumulated and given scores. The larger the number of iterations the more likely multiple similar ellipses will be accumulated into a single ellipse and given a higher score. At the end of the `for` loop the accumulator is searched for ellipses with high scores, which are placed in a best ellipse table. To reduce redundant work, the best ellipses found are removed from the image. Because these best ellipses should no longer exist the accumulator is cleared. In this way, previously found ellipses will not generate high scores in the accumulator overshadowing ellipses not found.

2.2 Ellipse Dissection

As can be seen from the High Level Algorithm Details previously described the algorithm it self is fairly straight forward. The difficulty arises in actually parameterizing the ellipse such that it can be accumulated. This section describes in detail what is necessary to accomplish this.

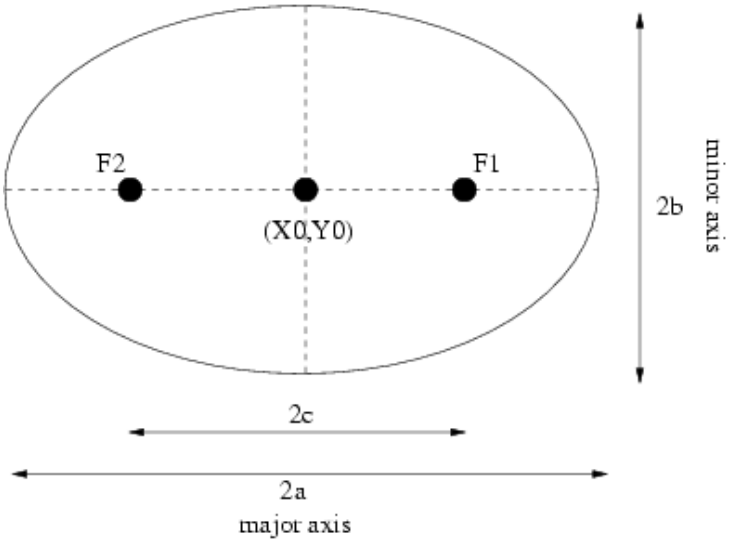


Figure 1: Ellipse Anatomy

Ellipse Equation: $a(x-p)^2+2b(x-p)(y-q)^2+c(y-q)^2=1$
With restriction: $ac-b^2>0$

An ellipse can be described in two ways, either by its center coordinate, semi-major axis length, semiminor axis length, and orientation (p,q,a,b,θ), or by its center coordinate, radius out from the foci 1, radius out from foci 2, and orientation (p,q,r_1,r_2,θ). The quin-tuple definition (p,q,a,b) was used in this implementation because it followed from the equations derived to find the ellipse in an image. The orientation, θ, was not used because no suitable equations could be found to derive it with the information in the image. This is also the reason the second form, i.e. using the radii, was not used.

It is worthing noting, McLaughlin [2] and Schuler [3] use the second form because it seemed to uniformally distribute the ellipse parameters across the hough space. The equations to produce the second form were not forth coming from those papers, and the author could not find or derive them elsewhere. Because the

orientation was not saved, this implementation only detects ellipses with major axis 0^o and 90^o from the x axis. However, this limitation was considered minor in respect to the overall problem.

Ellipse 4-tuple definition (p,q,a,b)

- `p` = x coordinate of ellipse center
- `q` = y coordinate of ellipse center
- `a` = semimajor axis length
- `b` = semiminor axis length

2.2.1 Determining Ellipse Center

There are five steps to determine an ellipse's parameters from an image starting from finding the center coordinates of the ellipse to determining the semimajor axis' length (a), semiminor axis' length (b), and half the distance between the foci (c).

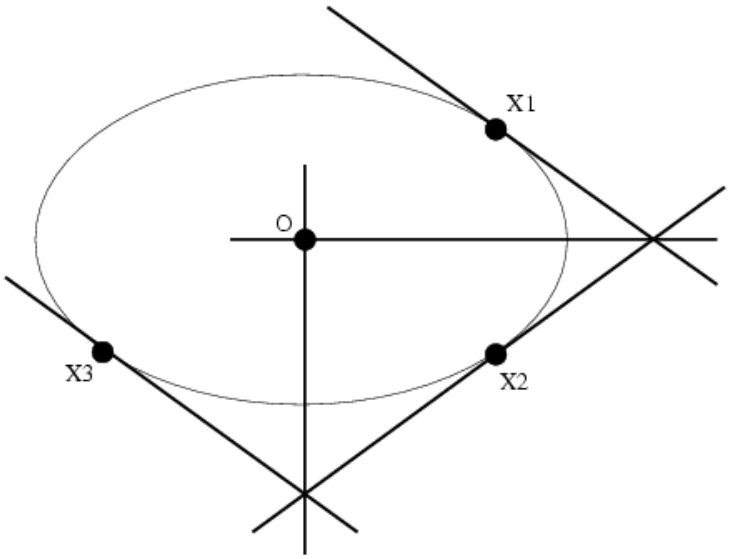


Figure 2: Determining an ellipse center, which is located at the intersection of the bisections of the three tangents to the ellipse.

1. Select three points, X_1, X_2, and X_3

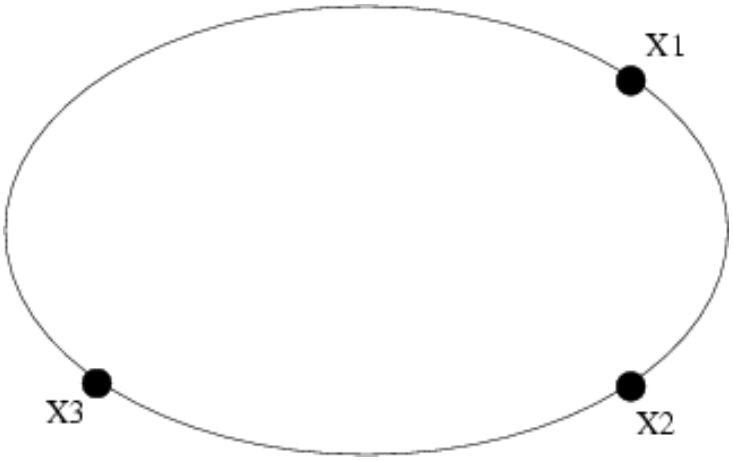


Figure 3: Selecting three points to check for an ellipse.

Three points are randomly selected from the image such that each point has an equal opportunity to be chosen. Three times the number of iterations random numbers were generated from 1 to the length of the image in subindicies to form sets of three points for each iteration. A subindex is the number of a cell in a matrix and ranges from 1 to the number of cells in the matrix. This is an alternative form for specifying a matrix cell from the normal row, column form.

Only unique random numbers generated for subindicies were kept to better cover the image, because each iteration requires three random points. If, after throwing away duplicate points, there were not enough points for all iterations specified, random numbers were generated until there were enough. All numbers were kept from this second generation, even if they duplicated the first sets.

2. Determine the equation of the line for each point where the line's slope is the gradient at the point: $y = mx + b$. This is done by checking the pixels around the point and performing a least squares line fit to them.

Determining the point's line equation is easy with MATLAB. `Roipoly` was used to select points in a seven by seven region around the point of interest. From the coordinates of these points we use the `polyfit` to find the slope m_1 and y-intercept b_1 for the point of interest.

3. Determine the intersection of the tangents passing through point pairs (X_1, X_2) and (X_2, X_3)

The tangent intersection points t_{12} and t_{23} are found by solving these systems of linear equations for the x and y coordinates:

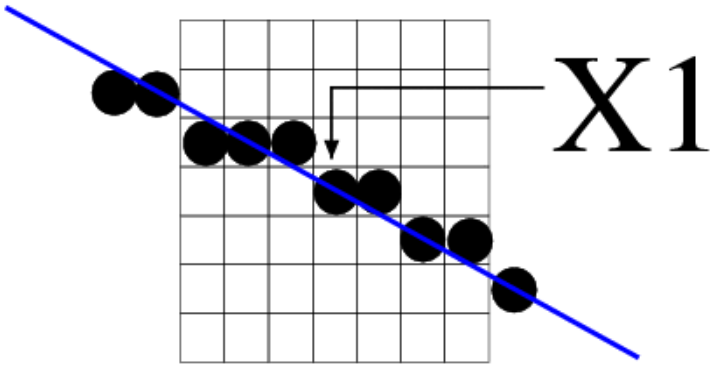


Figure 4: Determine the slope and y-intercept of the line passing through the selected point based on its neighbor pixels represented by the gridded pixels.

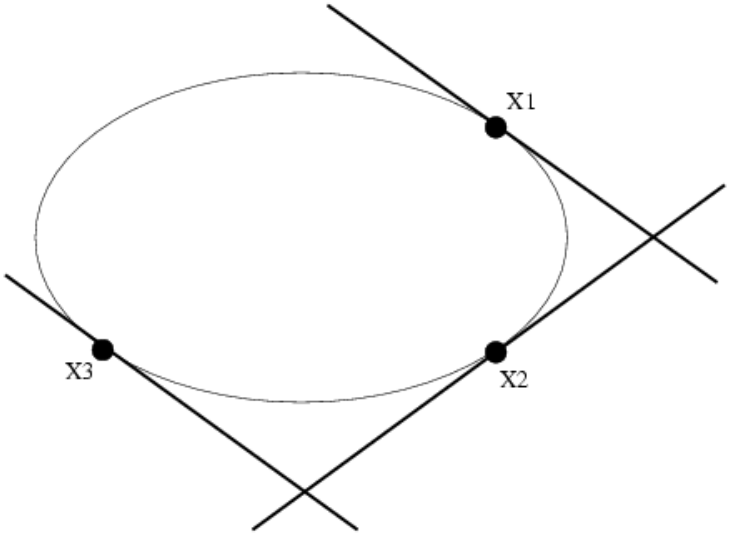


Figure 5: Tangents to the ellipse at points X_1,X_2, and X_3. The ellipse's center is located where the bisectors of the tangent intersections cross.

Tangents X_1 and X_2 for t_{12}:

$$\begin{bmatrix} m_1x & + & b_1 & - & y & = 0 \\ m_2x & + & b_2 & - & y & = 0 \end{bmatrix}$$

Tangents X_2 and X_3 for t_{23}:

$$\begin{bmatrix} m_2x & + & b_2 & - & y & = 0 \\ m_3x & + & b_3 & - & y & = 0 \end{bmatrix}$$

4. Calculate the bisector of the tangent intersection points. This is a line from the tangent's intersection, t, to the midpoint of the two points, m.

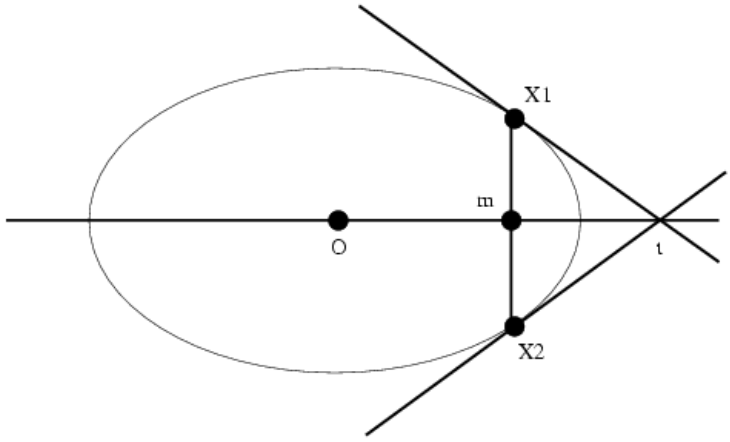


Figure 6: The line bisecting a tangent is found using the point slope line equation, the midpoint between the two points of interest, m, and the intersection of the points of interest's tangents, t.

The midpoint coordinate m_{12} equals half the distance from X_1 to X_2. The midpoint coordinate and bisection coordinate t_{12} are used to get the bisection line equation. This is found by solving the following equation to find the slope:

$$slope = \frac{m_y - t_y}{m_x - t_y}$$

and using the slope in the line equation to find the y-intercept:

$$b = slope * x - y = slope * t_x - t_y$$

the bisection line is then: $y = slope * x - b$

5. Find the bisectors intersection to give the ellipse's center, O

The ellipse's center is located at the intersection of the bisectors. The intersection coordinates are found using the bisectors line equations determined in step 4 in the following system of linear equations.

Ellipse center located at (x,y) derived from:

$$\begin{bmatrix} m_1 x & + & b_1 & - & y & = 0 \\ m_2 x & + & b_2 & - & y & = 0 \end{bmatrix}$$

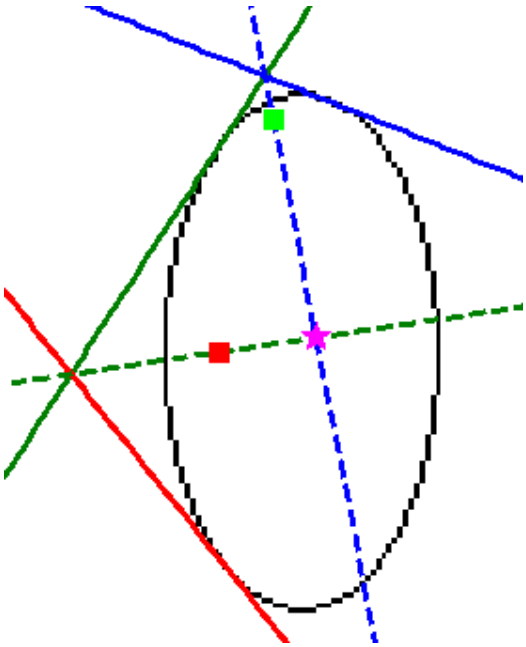


Figure 7: The tangents, bisectors, and center of ellipse found by the implemented algorithm.

2.2.2 Determining semimajor (a) and semiminor axis' (b) lengths and half the distance between foci (c)

Now that the ellipse's center (p,q) has been determined (in the previous section) the remaining ellipse parameters:

- `a` - semimajor axis length
- `b` - semiminor axis length
- `c` - half the distance between foci.

can be found from the ellipse equation: $a(x-p)^2+2b(x-p)(y-q)^2+c(y-q)^2=1$ using the three points randomly selected to create three linear equations with respect to a, b, and c. First, the ellipse is translated to the origin to reduce the ellipse equation to: $ax^2+2bxy+cy^2=1$. This is done by subtracting p from x and q from y for the three points selected in the beginning X_1, X_2, and X_3. Once the ellipse is translated to the origin, the following system of linear equations is solved to find the ellipse parameters a, b, and c:

$$\begin{bmatrix} ax_1^2 & + & 2bx_1y_1 & + & cy_1^2 & = 1 \\ ax_2^2 & + & 2bx_2y_2 & + & cy_2^2 & = 1 \\ ax_3^2 & + & 2bx_3y_3 & + & cy_3^2 & = 1 \end{bmatrix}$$

2.2.3 Verifying the Ellipse Exists in the Image

Even though at this point the ellipse parameters (p,q,a,b,c) were found it is possible the ellipse does not exist in the image. Two checks occur to verify the ellipse exists. First, because the ellipse is defined by the general equation for a conic section:

$ax^2+bxy+cy^2+dx+ey+f=0$

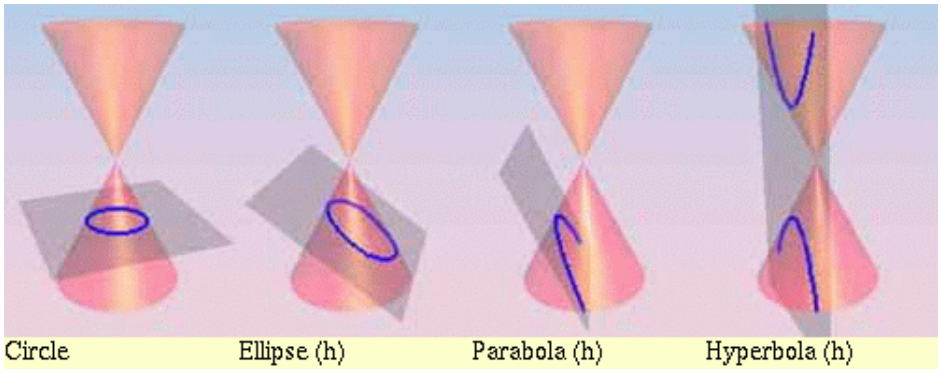


Figure 8: Example of different two-dimensional shapes derived from passing a plane through a conic section [1].

The sign of $4ac-b^2$ determines the type of conic section [1]:

$$\begin{array}{cc} >0 & \text{Ellipse or Circle} \\ =0 & \text{Parabola} \\ <0 & \text{Hyperbola} \end{array}$$

If the sign is positive then it is an ellipse. Even though the ellipse equation is satisfied as we see from Figure 9 it is possible the ellipse does not have enough pixels in the image.

To determine if the ellipse exists in the image the equation of the ellipse is used to generate points in the image on the perimeter to the ellipse. The number of points generated is equal to the circumference of the ellipse, which is found

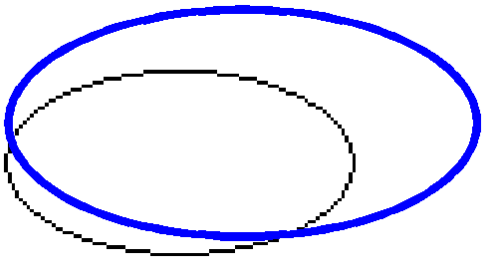


Figure 9: The black ellipse is in the image. The blue ellipse is verified by the ellipse equation, however, it does not actually exist in the image because its ratio of pixels to circumference is to low.

p	q	a	b	score
98.8937	99.5075	25.9742	47.2827	3.0000
100.5589	99.2206	25.5182	37.1271	2.0000
105.9815	82.0521	28.4240	56.8710	1.0000
115.0860	86.8599	38.8704	58.1393	1.0000
109.0266	102.1437	45.7310	43.1455	1.0000
103.0161	95.9755	20.9283	51.6607	1.0000
89.7838	122.2568	11.4692	19.6179	1.0000

Figure 10: Example accumulator.

with the equation: $\pi * semimajor\_axis * semiminor\_axis$. These points are used to generate a mask of the ellipse, which is 'anded' with the image. The number of pixels in the new image are counted and divided by the circumference of the ellipse. This yields a ratio of pixels to circumference. If the ratio is greater than a threshold specified by the user the ellipse exists in the image.

2.3 Accumulating

At this stage the ellipse's parameters were found and it was verified to exist in the image. Now the ellipse is added to the accumulator.

The accumulator stores the $(p,q,a,b,score)$ of an ellipse. The half distance between the foci, c, is not stored because it is not needed to generate ellipse points. Ellipse points are generated by solving the following equations for $\phi = 0\ to\ 2*\pi$:

$$x = a * cos(\phi)$$
$$y = b * sin(\phi)$$

The number of points generated are equal to the number of values used between $[0$ and $2*\pi]$, in this algorithm the number of values generated is equal to the circumference of the ellipse.

Below is an example accumulator. The best ellipse has a score of 3.0, is centered at (98.9,99.5), has semimajor axis of length 25.97 and semiminor axis length 47.3. Figure 11 shows the best ellipse found over the original image.

The following three steps occur to accumulate a new ellipse's center coordi-

- If the distance between the new ellipse center is within a threshold. $\sqrt{(p_i - p)^2 + (q_i - q)^2} >$ distance_threshold
- $|a_i - a| >$ semimajor_axis_threshold.
- $|b_i - b| >$ semiminor_axis_threshold.

2. For any ellipse in the accumulator where the above conditions hold, perform a weighted average between each of the ellipse parameters (use the score as the weight) and replace the ellipse in the accumulator with the new weighted ellipse, then increase the score for this ellipse by one.

   Example weighted average of semimajor axis length:

$$\frac{a_i * score + a}{score + 1}$$

3. If there are no ellipses in the accumulator that satisfy this condition place the new ellipse in the accumulator with a score of 1.

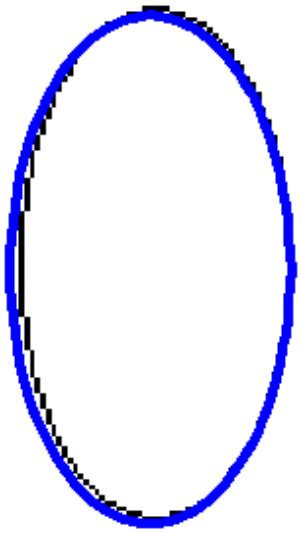


Figure 11: Best ellipse with score 3.0 from the example accumulator shown in blue over the black ellipse image.

2.4 Storing Best Ellipses and Repeating

After the `for` loop to accumulate ellipses completes the algorithm finds the best ellipses in the accumulator and stores them in a matrix of the same form as the accumulator $(p, q, a, b, score)$. Ellipses are added to the best ellipses matrix the same way they are stored in the accumulator described in the previous section.

Each ellipse is compared to the new ellipse and if they are similar they are weight averaged together based on their scores. This prevents duplicate ellipses from occurring in the best ellipse table if they are found during different epochs.

When an ellipse is placed into the best ellipse matrix it is removed from the image to increase the likelihood other ellipses in the image will be found. Figure 18 shows an example of the found ellipses removed from the image.

Once all the best ellipses are added to the matrix and removed from the image the accumulator is cleared for the next epoch and the process repeats.

3 Results

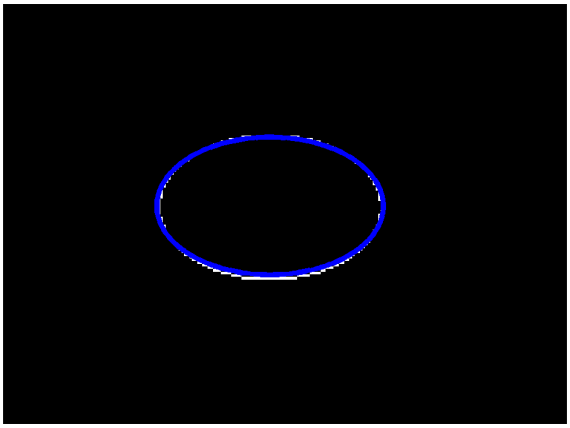


Figure 12: Ellipse found in blue with score seven after 5 epochs 10 iterations per epoch.

Figure 12 illustrates the result of running the program on an ellipse with orientation 0^o to the x-axis. The program ran for 5 epochs at 10 iterations per epoch.

The best ellipse found scored seven. The ellipse does fit perfectly on the ellipse in the image, the non-overlapping area is a product of stretching the image in MATLAB.

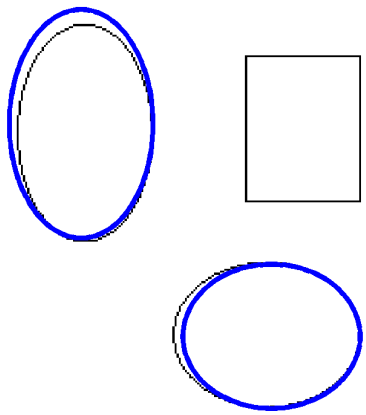


Figure 13: Non-ellipse object example. Both ellipse found after 5 epochs 30 iterations per epoch.

To determine if the ellipse program discriminates against non-ellipse objects the ellipse program was run with non-ellipse object images. Figure 13 exemplifies this. Both ellipses were found after 5 epochs at 30 iterations each. The program did ignore the square and found both ellipses. The ellipses found by the program were closer to original ellipses than they appear in the image because MATLAB had difficulty redrawing the found ellipses over the original image when the image was resized for input into the paper.

In the description of the algorithm it was mentioned only ellipses with orientation 0^o and 90^o to the x-axis are accumulated correctly as equations to find the orientation information could not be found or derived. Figure 14 demonstrates the programs behavior given an ellipse with orientation 45^o from the x-axis. As expected, the program does not find the ellipse. The center of the ellipse was found, but during the check if the ellipse exists the ellipse is drawn without an orientation and therefore did not contain enough pixels to be considered an ellipse. If the equations for the orientation were found, it would be trivial to add this property to the ellipse program, as the sub-function to generate ellipse points to test for

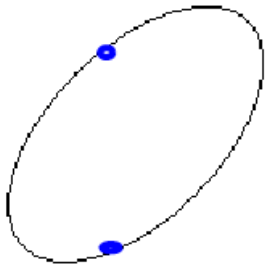


Figure 14: Demonstration of findellipse on ellipse with orientation 45^o from x-axis.

ellipse existence does have a parameter for orientation. To make the image more interesting than it otherwise would normally be, the score threshold was set to one to show all ellipses found. If the score threshold was set to some real value, such as four, the two blue ellipses would not occur on the image. The reason those tiny ellipses are given a score of one is discussed in the explanation of the noisy ellipse, Figure 15.

The ellipse program was tested against some noisy ellipse images. Figure 15 illustrates the result of adding 9% salt & pepper noise, with MATLAB's `imnoise`, to an ellipse image. The program ran for 20 epochs at 1000 iterations for 76 minutes. Of the 20,721 unique pixels in the image approximately 6,000 pixels were tested. Given the random nature of RHT it is unlikely these were 6,000 unique pixels. The only ellipse found is shown in blue, it had a score of 1. Given a longer running time and higher ellipse scoreThreshold the program should find the ellipse. It would also help if points were chosen with more intelligence than the pure randomness of RHT.

After running through many tests, it was determined when two points are selected about 75% of the semiminor axis apart and the third point was found far away from the first two, there was a higher possibility an ellipse would be correctly found. This could be achieved by adding parameters to the ellipse program that search for ellipses with a certain range of semimajor and semiminor axis

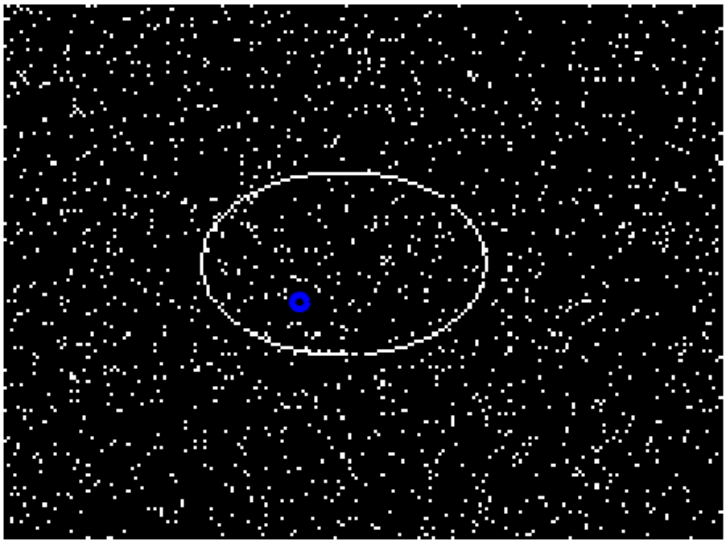


Figure 15: Ellipse image with 9% salt & pepper noise added. Blue 'ellipse' found had a score of 1.

lengths, which would cut down on the wasted computation for ellipses that could not possibly exist but must be tested because of the noise in the image.

Even though the findellipse program had difficulty finding ellipses in this noisy image, a small amount of preprocessing to remove the salt & pepper noise would greatly increase the possibility of finding the ellipses, as shown with the quarter experiment,

Figures 16 and 17 illustrate the program behavior for overlapping ellipses. The program did find all ellipses over different runs. Figure 16 is the result of a 29 minute run over 10 epochs and 100 iterations per epoch. Figure 17 is the result of running the program for 68 minutes over 10 epochs at 200 iterations per epoch. Figure 18 shows the image with all ellipses found subtracted. The ellipses found were not perfect because much of the image ellipses remain. This explains why, after such a long run, the all ellipses were not found in the overlapping image because the program kept using points from ellipses already found. One solution to this problem is to allow the user to specify a thickness of an ellipse to remove from the image. With a sufficiently thick ellipse all points for a found ellipse will be removed, reducing the computation time to randomly find other ellipses in the image.

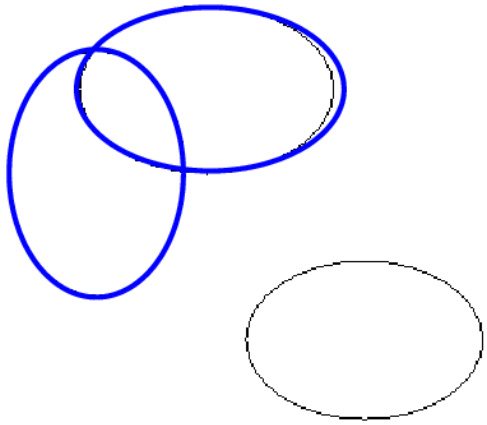


Figure 16: Result of running findellipse on overlapping ellipses for 10 epochs with 100 iterations per epoch.

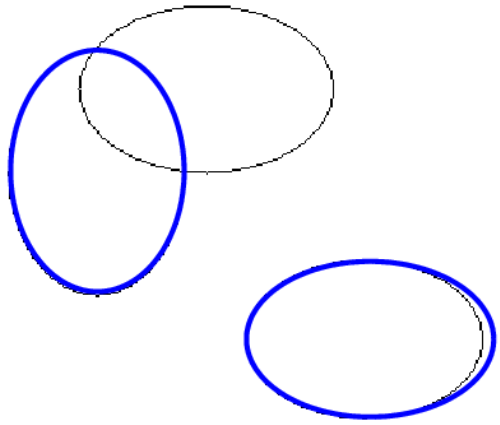


Figure 17: Result of running findellispe on overlapping ellipses for 10 epochs at 200 iterations per epoch.

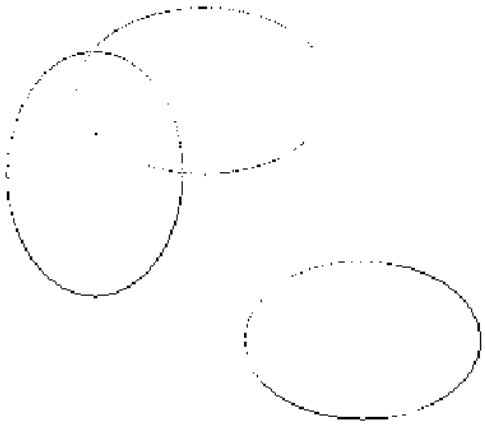


Figure 18: Result of subtracting all ellipses seen in Figure 17 from overlapping ellipse image. Notice the ellipses found were not completely removed from the image.

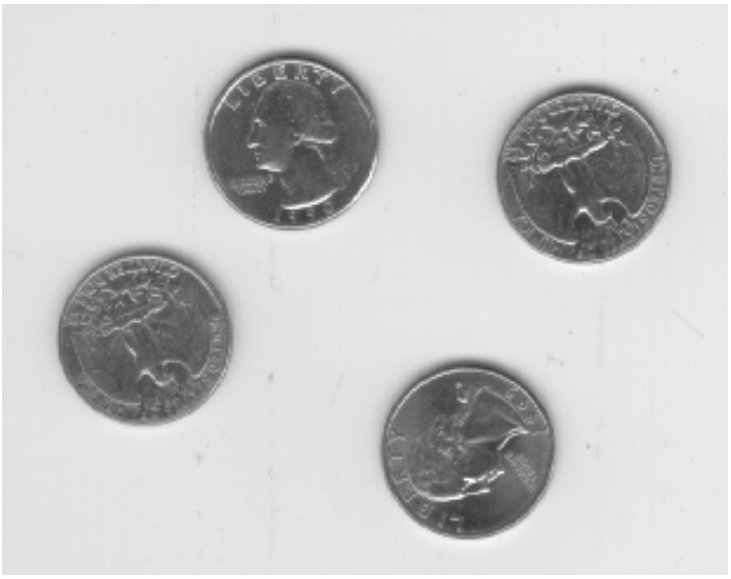


Figure 19: Example real-world image of four quarters.

Figure 19 shows a real-world image the program ellipse detected. This is image eight.tif from MATLAB, consisting of four quarters. A small amount of image preprocessing was done before ellipse detection because the findellipse program works best with ellipse edges. Figure 20 illustrates a bad way to detect edges on the image. This image was produced using the original image directly and MATLAB's `edge` function. Findellipse had the same difficulty with this image as the noisy ellipse image.

To correct this, the original image was median filtered using an eight by eight matrix with MATLAB's `medfilt2` function. Figure 21 shows the filtered image. The visage of George Washington and Eagle were nicely flattened by the median filter. Figure 22 shows the edge image of the median filtered image. This image was much easier for ellipse detection as seen in Figure 23 where all of the quarters were found. This demonstrates that a small amount intelligent preprocessing goes along way in object detection.

The RHT ran for two hours on the quarter image and processed 10,800 non-unique pixels. As a comparison, the normal Hough Transform would need to transform all pixels in the 242x308 image, equaling 74,546 pixels. Given the HT and RHT perform approximately the same pixel computations it would have taken fourteen hours for the HT to find the quarters. In addition, the HT would have needed to store information on all 74,546 pixels while the RHT only stores

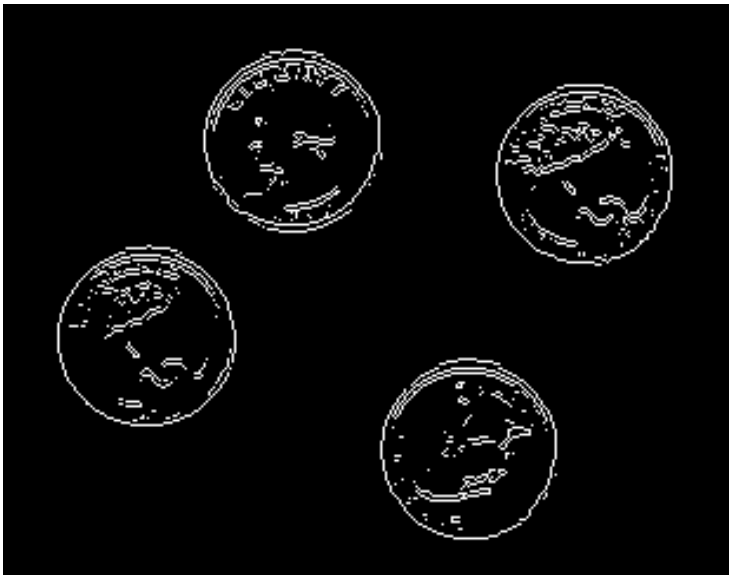


Figure 20: Edge image directly from the original quarters image. The findellipse program had the same difficulty with this image as the noisy image.

information on the current pixel three set and the ellipses found. This succinctly illustrates the benefits in reduced computation time and storage required by the Randomized Hough Transform over the Hough Transform

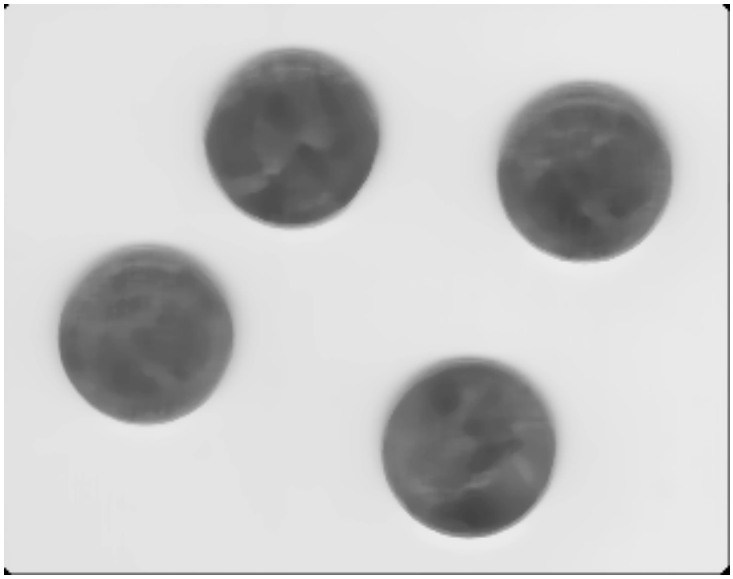


Figure 21: Simple preprocessing of quarter image before running findellipse. This is the result of median filtering with and eight by eight matrix on the original image. Nicely flattens the quarters.

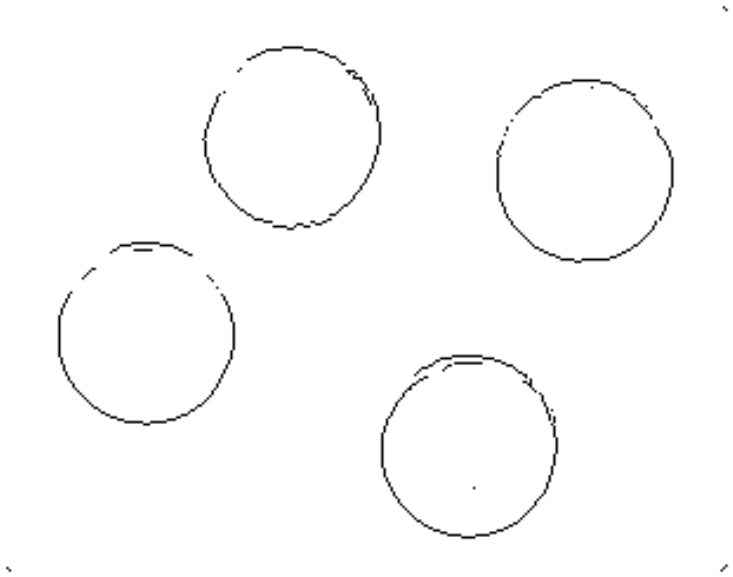


Figure 22: Edge image of the median filtered image quarter image.

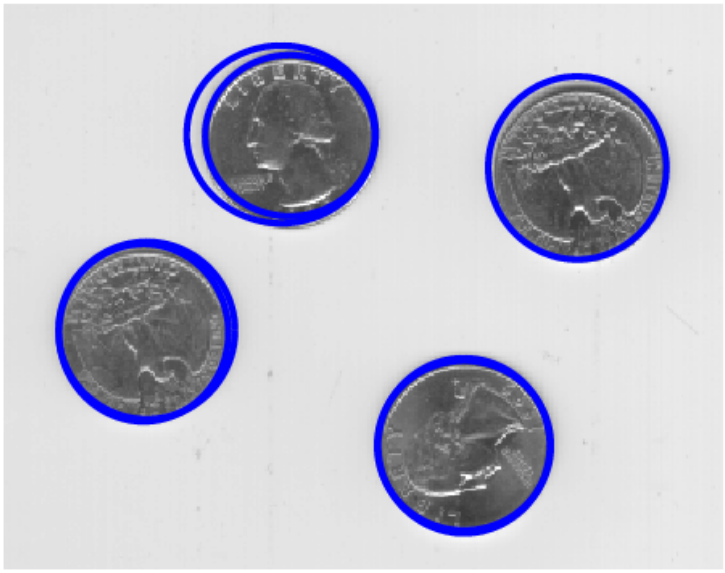


Figure 23: Ellipses found in blue on the edge image of the median filtered image overlaying the original image.

4 Conclusions

The Randomized Hough Transform performed well in the experiments. RHT accurately discriminated between ellipses and non-ellipse objects. In noisy images the RHT had difficulty finding the ellipses.Using image preprocessing to reduce the noise, and the ability to limit the search space to ellipses in a range of semi-major and semiminor axis lengths, these problems can be compensated for.

The majority of time spent developing this project consisted of finding and deriving the ellipse equations. The Randomized Hough Transform algorithm was simple once the equations were collected. Given the expense of finding the equations would also have occurred if the normal Hough Transform was implemented, the RHT did significantly reduce HT computation time and storage requirements needed to find ellipses.

5 Program Manual

The Randomized Hough Transform was implemented in the MATLAB function `findellipse`. The program requires the `ellipsep.m` file to generate (x,y) points on the ellipse. In addition, because the program is highly configurable all parameters are set inside the `findellipseParams.m` file. This is more convenient than a function that takes fifteen arguments.

The program is run in MATALB with `findellipse(image)` where `image` is a binary matrix with 1's as the pixels to look at for ellipses. The `image` matrix is not preprocessed in findellipse and should be an edge matrix when passed to the program.

`Findellipse` returns a matrix with the best ellipses found, and image matrix with all found ellipses removed. The best ellipses matrix returned is of the form:

p	q	a	b	score
98.8937	99.5075	25.9742	47.2827	3.0000
100.5589	99.2206	25.5182	37.1271	2.0000
103.0161	95.9755	20.9283	51.6607	1.0000
89.7838	122.2568	11.4692	19.6179	1.0000

where:

- `p` = x center coordinate
- `q` = y center coordinate
- `a` = semimajor axis length
- `b` = semiminor axis length
- `score` = the ellipses score

Note, the best ellipses matrix does not contain the ellipse orientation, therefore only ellipses 0^o and 90^0 to the x-axis are accurately detected.

The following is a description of all the parameters in `findellipseParams.m`. The parameters should be directly changed in this file.

- **scoreThreshHold** If an ellipse's score in the accumulator is $>$ scoreThreshold it is considered an ellipse in the image.
- **iterations** Numbers of times to accumulate potential ellipses.

- **maxEpochs** The total number of epochs to execute. An epoch is the time the algorithm takes to find an ellipse. Each epoch runs a loop for the number of 'iterations' set above. Only the best ellipses in each epoch are saved.

  Note: the algorithm may stop prematurely if maxEpochsWithoutFindingElipse is set to a low number.

- **maxEpochsWithoutFindingEllipse** The maximum number of consecutive epochs the program will execute before prematurely terminating the algorithm.

- **ratioPixelsFoundToCircumferenceThresh** The check to determine if an ellipse is really in the image A number of pixels equal to the circumference of the ellipse that would be on the perimeter of the ellipse are extracted from the image. If the number of pixels divided by the circumference of the circle is greater than the ratioPixelsFoundToCircumferenceThresh it is considered to exist

- **distSimThresh**, **majorAxisSimThresh**, and **minorAxisSimThresh** If the distance between an accumulated ellipse and the ellipse being added to the accumulator is $<$ distSimThresh and the absolute difference between the major axises is $<$ majorAxisSimThresh and the absolute difference between the minor axises is $<$ minorAxisSimThresh then the ellipse in the accumulator is close enough to the ellipse being tested so they are averaged together. The accumulated ellipse is weighted by its score in the accumulator.

- **showError** Any error that occurs during accumulation is ignored because it is most likely the result of the ellipse not being in the image. All error messages are suppressed unless showError = 1. Note: if set to 1 the program still ignores the error, but it will tell you it ignores it.

- **showBest** 1 to show the best ellipses on the image at the end. 0 don't do it.

- **debugOn** 1 to print debugging info, 0 to not

- **showit** 1 to show the accumulated ellipse, 0 to not

References

[1] David Manura. Dave's math tables: Conic sections.

[2] McLaughlin and Robert. Randomized hough transform: Improved ellipse detection with comparison. Technical Report JP98-01, 1998.

[3] Andrew Schuler. The randomized hough transform used for ellipse detection.

[4] Stockman Shapiro. *Computer Vision*. Prentice Hall, 2001.

[5] Lei XU, Erkki OJA, and Pekka Kultanena. A new curve detection method: Randomized hough transform (rht). *Pattern Recognition Letters*, (11):331–338, 1990.