The Radon Transform

Carsten Høilund

Aalborg University, VGIS, 07gr721

November 12, 2007

Contents

Chapter 1

Introduction

This is a report about the Radon transform for the course Image Processing held by Thomas Moeslund at Aalborg University in the autumn of 2007.

The Radon transform is named after the Austrian mathematician Johann Karl August Radon (December 16, 1887 – May 25, 1956) [4]. The main application of the Radon transform is CAT scans, Figure 1.1, where the inverse Radon transform is applied.

The Radon transform can also be used for line detection, which will be the focus of this report.

Chapter 2 on the next page explains the theory, mostly using pictures.

Chapter 3 on page 9 details the implementation using pictures. The MATLAB source code is also listed.

Chapter 4 on page 16 lists the results obtained with the implementation detailed in this report using two different input images and compares this with the result obtained using the Radon transform found in MATLAB.

Figure 1.1: A CAT scanner [1].

Chapter 2

Theory

This section is based on [2, 3].

Applying the Radon transform on an image $f(x,y)$ for a given set of angles can be thought of as computing the projection of the image along the given angles. The resulting projection is the sum of the intensities of the pixels in each direction, i.e. a line integral. The result is a new image $R(\rho,\theta)$. This is depicted in Figure 2.1 on the facing page.

This can be written mathematically by defining

$$\rho = x\cos\theta + y\sin\theta \tag{2.1}$$

after which the Radon transform can be written as

$$R(\rho,\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\delta(\rho - x\cos\theta - y\sin\theta)dx\,dy \tag{2.2}$$

where $\delta(\cdot)$ is the Dirac delta function.

There are two distinct Radon transforms. The source can either be a single point (not shown) or it can be a array of sources (as shown in Figure 2.1 on the next page). The method discussed in this report uses an array of sources. The theory is explained in Figures 2.1 on the facing page to 2.7 on page 8.

The Radon transform is a mapping from the Cartesian rectangular coordinates (x,y) to a distance and an angel (ρ,θ), also known as polar coordinates.

An example of the transform of an image for a specific angle is given in Figure 2.4 on page 6 and Figure 2.6 on page 7. The transform for a set of angels can be depicted in an image, as in Figure 2.7 on page 8.

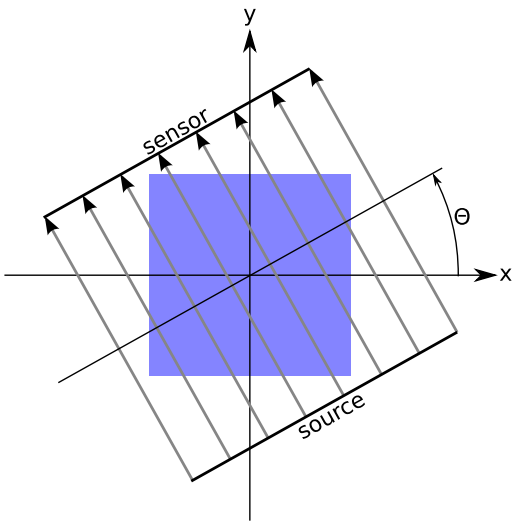

Figure 2.1: The source and sensor contrapment is rotated about the center of the object. For each angle θ the density of the matter the rays from the source passes through is accumulated at the sensor. This is repeated for a given set of angels, usually from $\theta \in [0;180)$. The angel 180 is not included since the result would be identical to the angel 0.

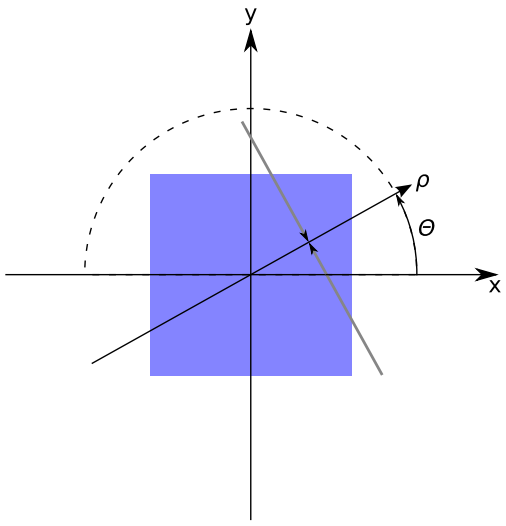

Figure 2.2: For each angle θ and each distance ρ the intensity of the matter a ray perpendicular to the ρ axis crosses are summed up at $R(\rho,\theta)$.

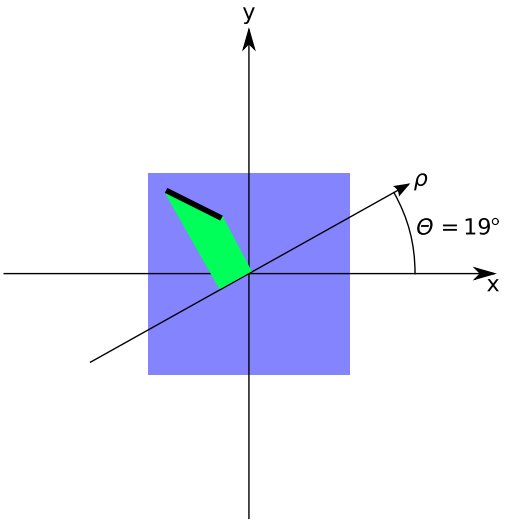

Figure 2.3: As an example, the line in this image will at $\theta = 19^o$ be distributed over a larger interval.

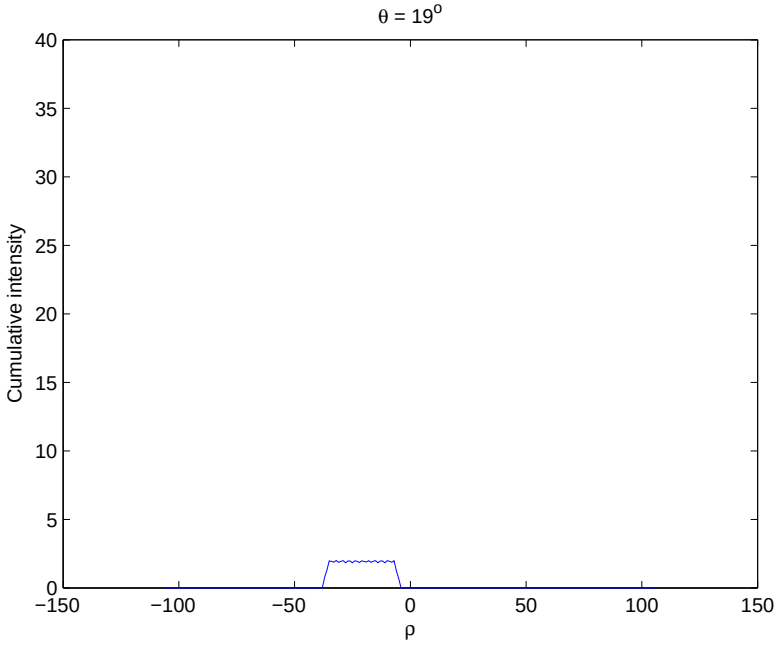

Figure 2.4: The result of a Radon transform with $\theta = 19^o$ which there is no definite peak.

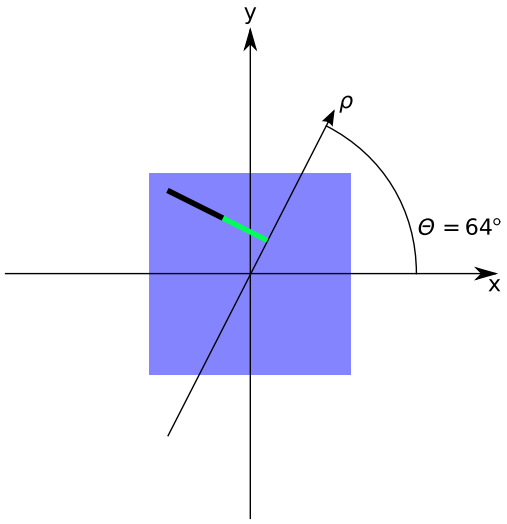

Figure 2.5: When $\theta = 64^o$ the line will be distributed over a very small interval.

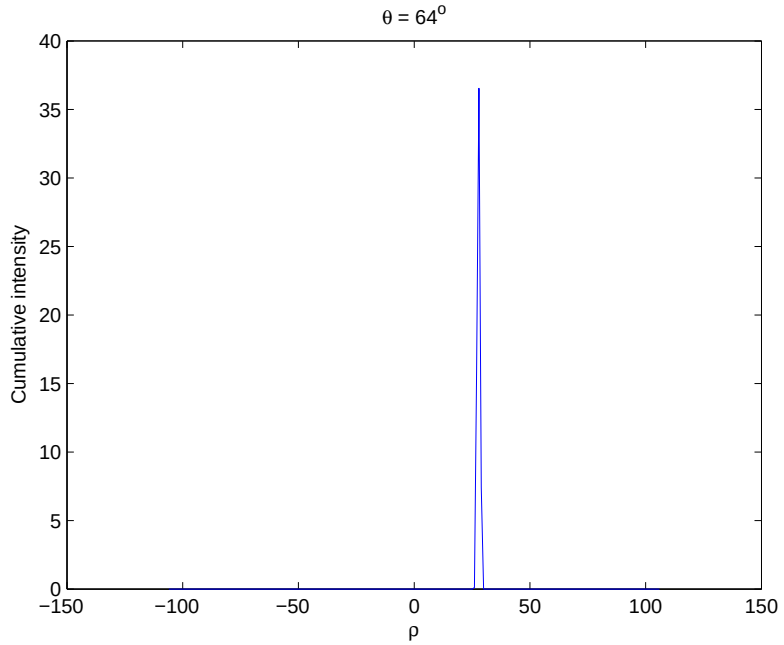

Figure 2.6: The result of a Radon transform with $\theta = 64^o$, perpendicular to the line in the image. This results in a peak, which makes it possible to read the line parameters.

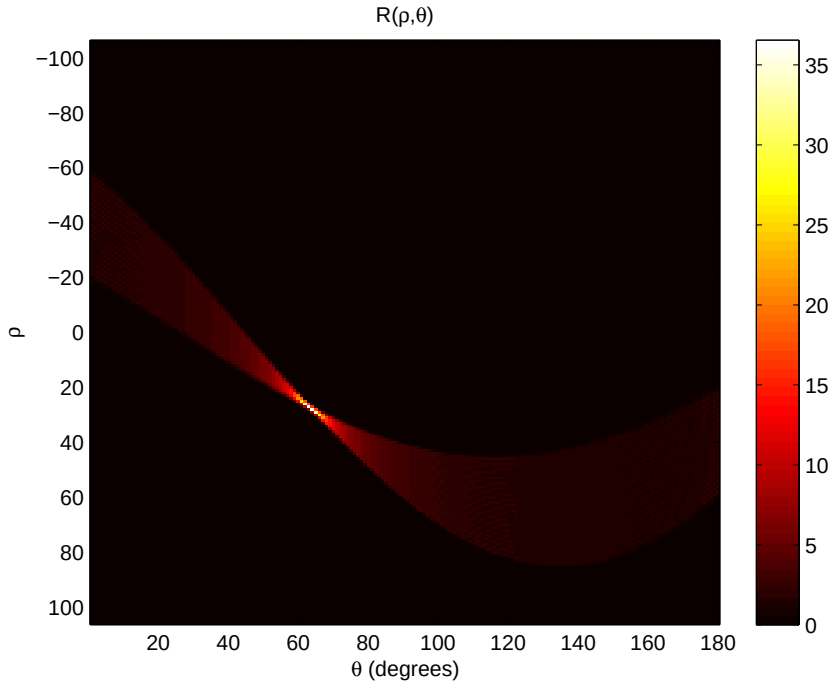

Figure 2.7: The complete Radon transform of the image. The white spot is the distance from the center and the angel at which the sum of intensities in the image peaks. It is thus the slope of the line along with the position.

Chapter 3

Implementation

The method with an array of sources is chosen, since this is more straight forward. Also, there are two general ways to implement the chosen Radon transform. It can either be a function of θ and ρ for which all matching pixels are calculated, or it could be implemented as a function of the image pixels. The latter is the easiest but is, however, the Hough transform.

It is implemented in MATLAB. The source is shown in Listing 3.1 on page 13. The function takes a grayscale image as input and displays the Radon transform as described herein.

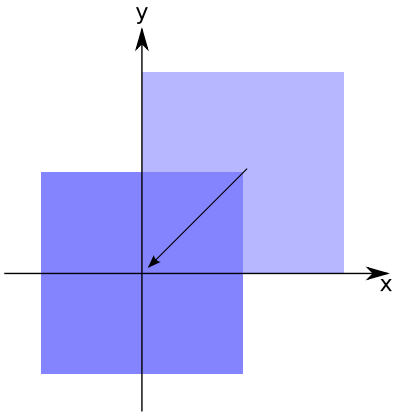

Figure 3.1: The pixel coordinates in an image is usually only positive. The first step is therefore to center the image which can be accomplished by subtracting half the width from each x coordinate and likewise half the height from each y coordinate.

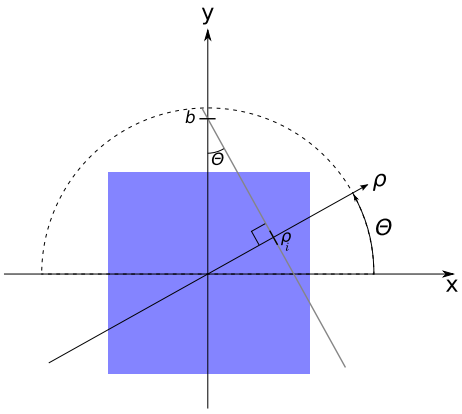

Figure 3.2: The equation of the summation line is given as $y = ax + b$. As can be seen by using trigonometry, the inclination is $a = -\frac{cos(\theta)}{sin(\theta)}$ and the intersection with the y axis is $b = \frac{\rho}{sin(\theta)}$. This fits with Eq. (2.1) on page 4. These parameters are determined for each combination of θ and ρ. The maximum ρ is set equal to the length of the diagonal of the image. The ρ coordinates are, like x and y, also centered.

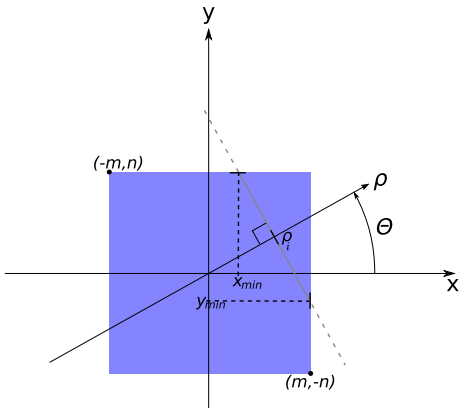

Figure 3.3: In order to reduce the number of calculations necessary the maximum and minimum of either x or y are determined.

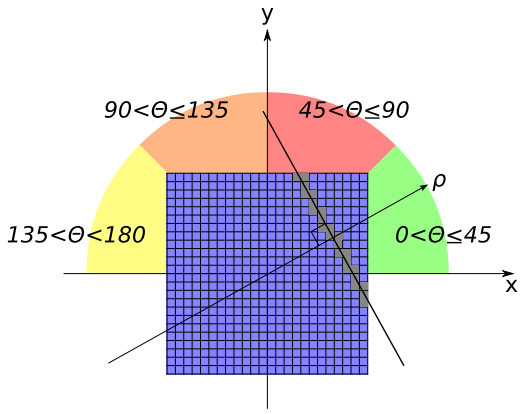

Figure 3.4: Whether x or y is used as the variable and how the minimum and maximum of said variable is calculated depends on in which of the four areas depicted θ is in since using e.g. x as the variably when the summation line has an absolute inclination of more than 1 will cause some pixels to be skipped. Neither $\theta = 0$ nor $\theta = 180$ is included since the line perpendicular to this would have infinite inclination.

$$
\begin{aligned}
0 < \theta \leq 45: \quad x &= \frac{y-b}{a} \\
y_{min} &= max(-n, am+b) \\
y_{max} &= min(n, -am+b)
\end{aligned}
$$

$$
\begin{aligned}
45 < \theta \leq 90: \quad y &= ax+b \\
y_{min} &= max(-m, \frac{n-b}{a}) \\
y_{max} &= min(m, \frac{-n-b}{a})
\end{aligned}
$$

$$
\begin{aligned}
90 < \theta \leq 135: \quad y &= ax+b \\
y_{min} &= max(-m, \frac{-n-b}{a}) \\
y_{max} &= min(m, \frac{n-b}{a})
\end{aligned}
$$

$$
\begin{aligned}
135 < \theta < 180: \quad x &= \frac{y-b}{a} \\
y_{min} &= max(-n, -am+b) \\
y_{max} &= min(n, am+b)
\end{aligned}
$$

$$
\begin{aligned}
\theta = 180: \quad \rho &= x + \lfloor \frac{\rho_{max} - 2m}{2} \rceil \\
y &= [-m, m]
\end{aligned}
$$

Figure 3.5: The formulas used to calculate the coordinates and the minimum and maximum of the variable depending on the angle θ, where m is half the width of the image and n is half the height of the image, a is the inclination, b is the intersection with the y axis, and ρ_{max} is the size of the diagonal of the image, i.e. $\rho_{max} = \lceil \sqrt{(2m)^2 + (2n)^2} \rceil$. In e.g. the first equation the x coordinate depends on the chosen y coordinate in order to make sure all pixels along this line is taken into account when calculating the transform, as noted in Figure 3.4 on the preceding page.

Listing 3.1: Implementation

```
function [ res ] = myradon(f)

[N M] = size(f);

% Center of the image
m = round(M/2);
n = round(N/2);

% The total number of rho's is the number of pixels on the diagonal, since
% this is the largest straight line on the image when rotating
rhomax = ceil(sqrt(M^2 + N^2));
rc = round(rhomax/2);
mt = max(theta);

% Preallocate the matrix used to store the result
% add 1 to be sure, could also be subtracted when checking bounds
res = cast(zeros(rhomax+1,mt),'double');

tic
for t = 1:45 % below 45 degrees, use y as variable
    costheta = cos(t*pi/180);
    sintheta = sin(t*pi/180);
    a = -costheta/sintheta; % y = ax + b
    for r = 1:rhomax
        rho = r - rc;
        b = rho/sintheta; % y = ax + b
        ymax = min(round(-a*m+b),n-1);
        ymin = max(round(a*m+b),-n);
        for y = ymin:ymax
            x = (y-b)/a;
            xfloor = floor(x); % The integer part of x
            xup = x - xfloor; % The decimals of x
            xlow = 1 - xup; % What it says
            x = xfloor;
            x = max(x,-m);
            x = min(x,m-2);
            res(rhomax - r + 1,mt-t) = res(rhomax - r + 1,mt-t) + xlow*f(y+n+1,x+m+1);
            res(rhomax - r + 1,mt-t) = res(rhomax - r + 1,mt-t) + xup*f(y+n+1,x+m+2);
        end
    end
end
for t = 46:90
    costheta = cos(t*pi/180);
    sintheta = sin(t*pi/180);
    a = -costheta/sintheta; % y = ax + b
    for r = 1:rhomax
```

```
47          rho = r - rc;
48          b = rho/sintheta; % y = ax + b
49          xmax = min(round((-n-b)/a),m-1);
50          xmin = max(round((n-b)/a),-m);
51          for x = xmin:xmax
52              y = a*x+b;
53              yfloor = floor(y);
54              yup = y - yfloor;
55              ylow = 1 - yup;
56              y = yfloor;
57              y = max(y,-n);
58              y = min(y,n-2);
59              res(rhomax - r + 1,mt-t) = res(rhomax - r + 1,mt-t) + ylow*f(y+n+1,x+m↘
                    +1);
60              res(rhomax - r + 1,mt-t) = res(rhomax - r + 1,mt-t) + yup*f(y+n+2,x+m+1)↘
                    ;
61          end
62      end
63  end
64  for t = 91:135
65      costheta = cos(t*pi/180);
66      sintheta = sin(t*pi/180);
67      a = -costheta/sintheta; % y = ax + b
68      for r = 1:rhomax
69          rho = r - rc;
70          b = rho/sintheta; % y = ax + b
71          xmax = min(round((n-b)/a),m-1);
72          xmin = max(round((-n-b)/a),-m);
73          for x = xmin:xmax
74              y = a*x+b;
75              yfloor = floor(y);
76              yup = y - yfloor;
77              ylow = 1 - yup;
78              y = yfloor;
79              y = max(y,-n);
80              y = min(y,n-2);
81              res(rhomax - r + 1,mt-t) = res(rhomax - r + 1,mt-t) + ylow*f(y+n+1,x+m↘
                    +1);
82              res(rhomax - r + 1,mt-t) = res(rhomax - r + 1,mt-t) + yup*f(y+n+2,x+m+1)↘
                    ;
83          end
84      end
85  end
86  for t = 136:179 % above 135 degrees, use y as variable
87      costheta = cos(t*pi/180);
88      sintheta = sin(t*pi/180);
89      a = -costheta/sintheta; % y = ax + b
90      for r = 1:rhomax
91          rho = r - rc;
```

```
92          b = rho/sintheta; % y = ax + b
93          ymax = min(round(a*m+b),n-1);
94          ymin = max(round(-a*m+b),-n);
95          for y = ymin:ymax
96              x = (y-b)/a;
97              xfloor = floor(x);
98              xup = x - xfloor;
99              xlow = 1 - xup;
100             x = xfloor;
101             x = max(x,-m);
102             x = min(x,m-2);
103             res(rhomax - r + 1,mt-t) = res(rhomax - r + 1,mt-t) + xlow*f(y+n+1,x+m
                    +1);
104             res(rhomax - r + 1,mt-t) = res(rhomax - r + 1,mt-t) + xup*f(y+n+1,x+m+2)
                    ;
105         end
106     end
107 end
108 for t = 180 % the sum-line is vertical
109     rhooffset = round((rhomax - M)/2);
110     for x = 1:M % cannot use r as x in both res and f since they are not the same
            size
111         r = x+rhooffset;
112         r = rhomax - r + 1;
113         for y = 1:N
114             res(r,t) = res(r,t) + f(y,x);
115         end
116     end
117 end
118 toc
119 rhoaxis = (1:rhomax+1) - rc;
120 figure
121 imagesc(1:180,rhoaxis,res);
122 colormap(hot), colorbar
```

As can be seen, linear interpolation is used as given in Listing 3.2, where ylow gives the amount that falls into the y bin and yup gives the amount that falls into the $y+1$ bin.

Listing 3.2: Linear Interpolation

```
1 yfloor = floor(y);
2 yup = y - yfloor;
3 ylow = 1 - yup;
4 y = yfloor;
```

This concludes the implementation. The results obtained with this implementation is given in the next chapter.

Chapter 4

Results

The results obtained with the algorithm as described here differs from the built-in MATLAB Radon function. Using Figure 4.1 as the input image Figure 4.2 on the facing page shows the result of the algorithm as described in this report and Figure 4.3 on the next page depicts the result from the built-in MATLAB Radon transform.

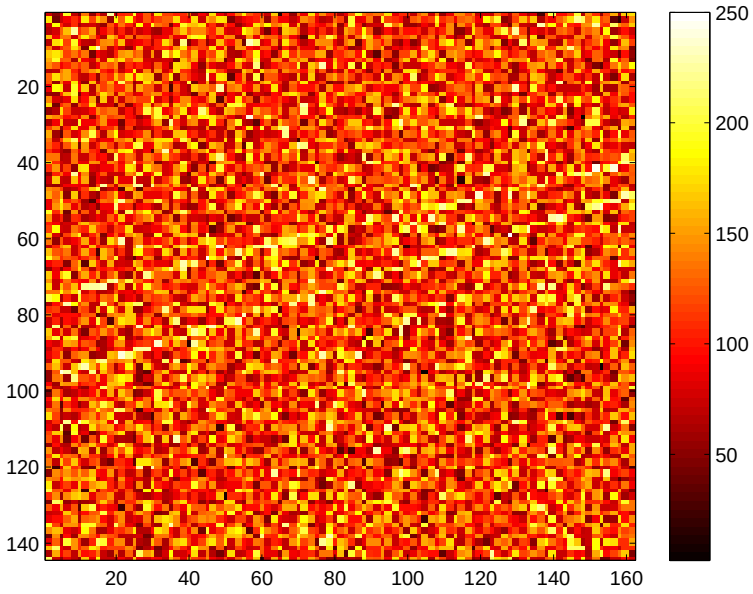

Figure 4.1: The input image. Notice the two lines faintly visible.

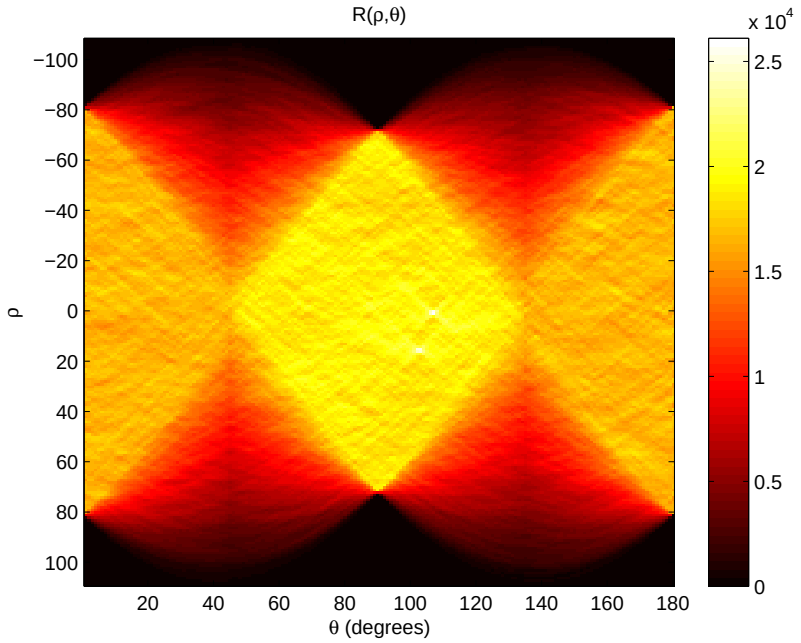

Figure 4.2: The result obtained with the implementation given herein. The two small, bright spots are the lines visible in the input image.

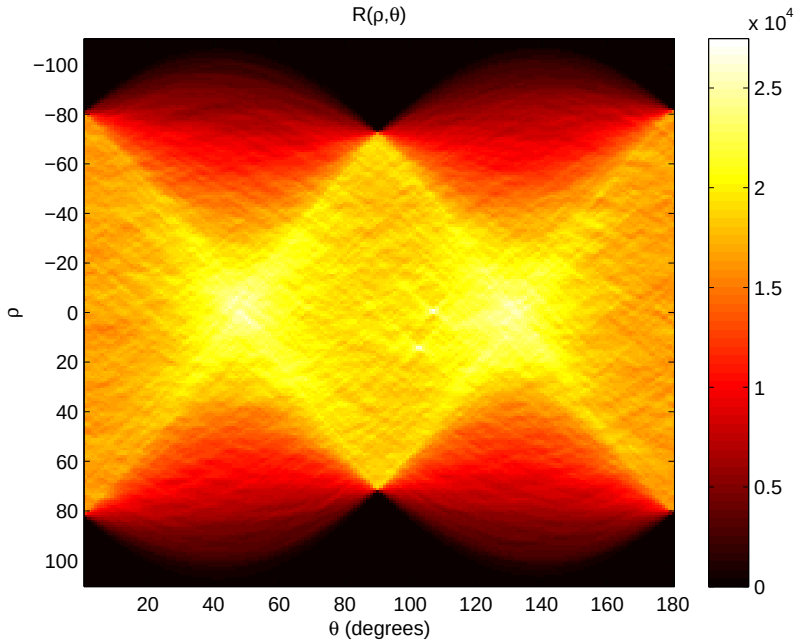

Figure 4.3: The result obtained using the built-in MATLAB Radon transform. The two lines are also visible here, but competes with the general accumulation of pixel intensities at $\theta = 45^{o}$ and $\theta = 135^{o}$.

This does not, however, imply that this implementation is useless. It has some advantages. Compared to Figure 4.3 on the preceding page the image in Figure 4.2 on the previous page can more easily be thresholded leaving only the pixels of importance.

The results are not always this different as can be seen when using Figure 4.4 as input which gives Figure 4.5 and Figure 4.6 on the next page, respectively.

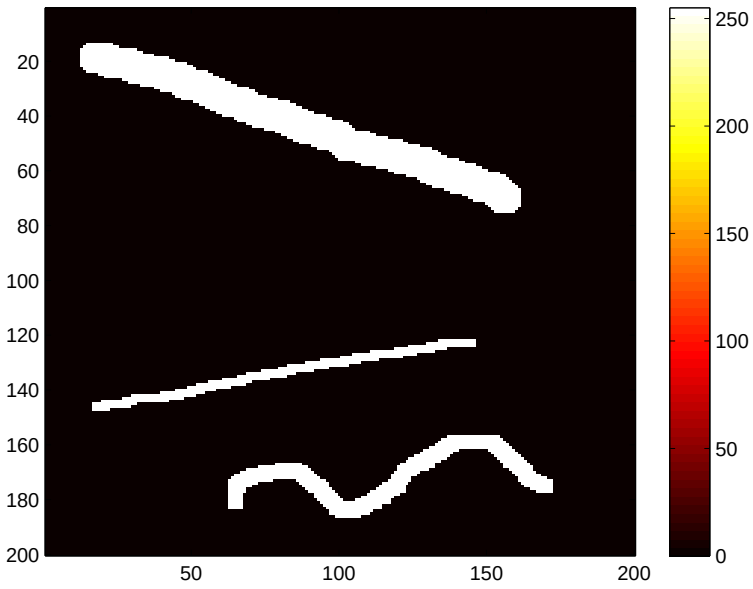

Figure 4.4: Another image used as input.

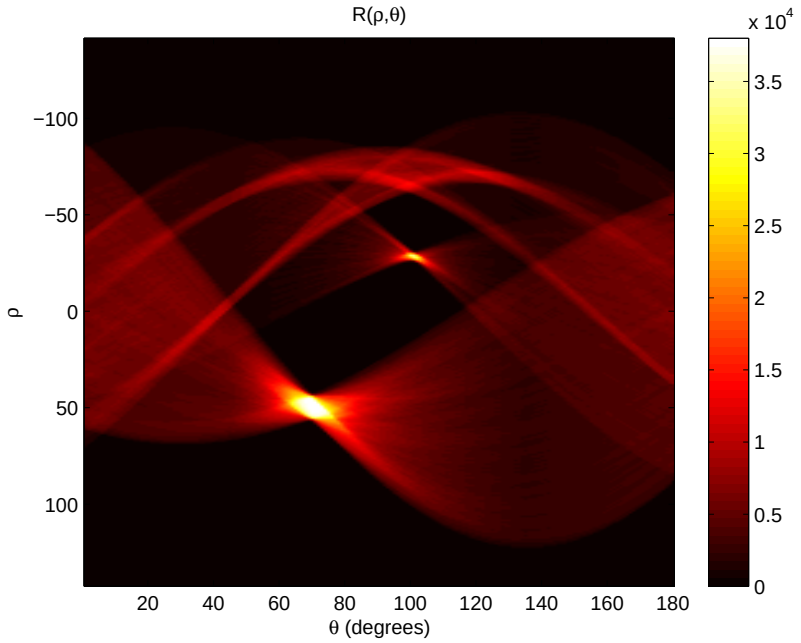

Figure 4.5: Result obtained with the implementation from this report. The fat line is the largest, bright spot and the thin line is the small, bright spot. The curves not meeting to form a bright spot is the wavy line.

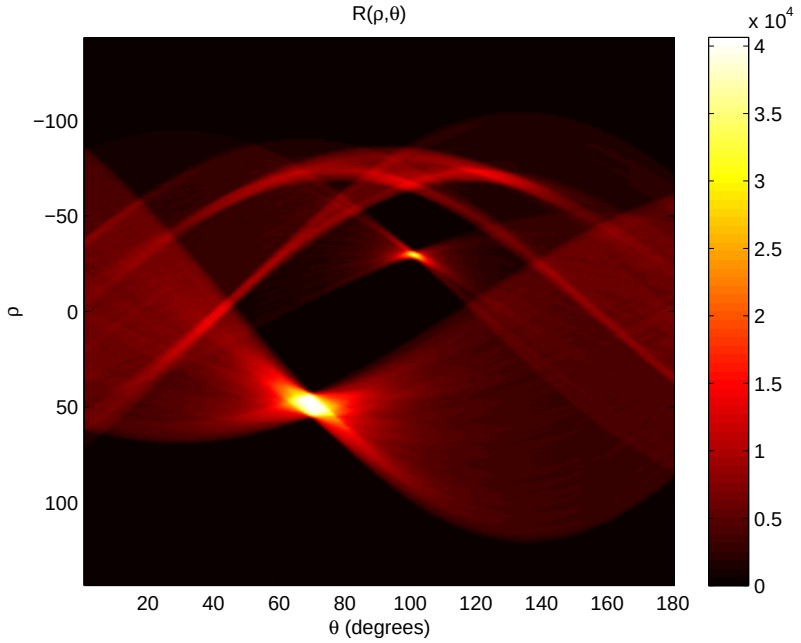

Figure 4.6: Result obtained with the implementation from MATLAB. This is nearly identical to Figure 4.5. The difference is presumably due to using a different interpolation.

Bibliography

[1] Muffet. CAT scanner for simulation. http://flickr.com/photos/calliope/357130113/.

[2] The MathWorks. Radon Transform . http://www.mathworks.com/access/helpdesk_r13/help/toolbox/images/transfo9.html.

[3] Peter Toft. The Radon Transform . http://eivind.imm.dtu.dk/staff/ptoft/Radon/Radon.html.

[4] Wikipedia. Johann Radon. http://en.wikipedia.org/wiki/Johann_Radon.