EE 368 Image processing ...
Aerial image processing, finding the darn roads

Logi Vidarsson, SU # 4799011

May 30, 2001
## Contents

1 Overview .................................................. 2

2 Refining the image .......................................... 2
   2.1 Thresholding ........................................... 2
   2.2 Noise reduction ........................................ 4

3 Finding the crossroads ...................................... 5
   3.1 Taking the Hough transform ............................. 5
   3.2 Finding the crossroads ................................. 6

4 Detecting paths .............................................. 7
   4.1 A virtual car ............................................ 7
   4.2 Constructing the graph .................................. 8

5 Conclusion and future work .................................. 9
1 Overview

The objective of our project is to write a program that can extract information on the roads on the image. As with any information extraction system \(^1\) we want to “exaggerate the positives, and hide the negatives”. That is somehow bring out the roads in the image before processing.

After refining the input image, we use the Hough transform to extract the roads into a mathematical form. With the roads in a mathematical form its easy to seek the intersections, those form the nodes of the graph we intent to form, describing the road structure on the image.

Using virtual cars to drive from these nodes we can easily determine the paths of the graph, and the distance between the nodes. Hence we can construct a weighted graph that can easily be used to determine the best path between any two points, much like Mapquest. We summarize our system in a block diagram.

![Block diagram](image)

Figure 1: Block diagram

2 Refining the image

As a little introduction to my country Iceland, we grab a set of test images showing the city of Reykjavik \([1]\) on a beautiful fall day. Since these images are generally somewhat large and generally to complex for demonstration purposes it was decided to use only a portion of the image, shown here in a black dashed box on figure 2.

In the refining phase, we want to change this picture of down town Reykjavik into a binary image, having 1’s for roads and 0’s otherwise.

2.1 Thresholding

Asphalt is generally gray, which for at least the city of Reykjavik is a rare color, so Thresholding is a natural choice. But we have a color image so we have three channels we can use for filtering

\(^1\)This holds for the fashion industry as well
Figure 2: A picture of my bellowed home city, Reykjavik Iceland [1]

purposes, but before we start to set thresholds we need to answer one important question. What is gray?

To help answer that question we take a look at a slice of our test image that shows both asphalt (the thang we are trying to extract) and other things found in cities, such as houses, gardens and so fourth. We plot the color intensities for the 47Th row.

![Color intensity values around the 47Th row](image)

Figure 4: Color intensity values around the 47Th row

Showing the slice that we take from the test image again to help us focus on certain areas.

![A slice of the test image](image)

Figure 5: A slice of the test image
Figure 3: A zoom in, used as a test image [1]

It looks like there is a road around the 30Th column and a slightly narrower road around the 90Th column. We note that the color intensity values are quite stable around the road pixels, suggesting that the color threshold method will give informative results. The blue channel however is more “steppy” that is has more extreme values, so we chose to set the higher threshold for blue slightly larger than for the other colors that is at 130 (out of 256), but we note that the green and red channels are somewhat lower so we set the higher threshold at 120. The lower limits seem more reliable and after some experimentation a value of 90 looks nice. The binary image we get after Thresholding can be seen on figure 6.

Figure 6: Threshold-ed test image

2.2 Noise reduction

However, as suspected the color gray does indeed appear on something else than asphalt even in my beloved city Reykjavik, so some noise and mis-detection is going to appear. Also these images were grabbed in JPEG format, which has the tendency to smear out colors, so Thresholding alone is not going to be sufficient.

We are at this stage most concerned with the missing road pixels, since that might distract our
line detection algorithm we will run later. So we perform the dilate operation to seal up any holes in the “asphalt” after some experimentation it turns out to be good to use a 3 by 3 matrix for the dilate operation. However now we have smeared out our roads, but that might make life hard for our line detection algorithm since it likes its roads narrow.

The erosion operation comes in to save the day, we use the same 3 by 3 erosion matrix as before. In fact we have performed the close operation using a 3 by 3 matrix, this is the procedure used generally to seal holes of zeros in images. [3]

![Figure 7: Refined test image](image)

We see that there still is some noise present in the image, but the main roads are clearly visible and narrow. As we will see the next phase is very tolerant to salt and pepper noise, since salt and pepper noise rarely forms up into lines.

3 Finding the crossroads

With the refined image at hand we can move on to finding a mathematical form for the roads. It comes as no surprise that roads usually are straight, at least according to my skydiving instructor when he referred to unsuitable landing sites.

Using the Hough transform we can easily extract the roads as formulas for lines, finding the intersection of all lines that occur inside the image, then gives us a list of potential crossroads.

3.1 Taking the Hough transform

The Hough transform provides a convenient way to find lines in the image, and extract their mathematical representation. Following the lecture notes in EE368 [2], we parameterize the lines by using the equation.

\[ \rho = x \cos \Theta + y \sin \Theta \]

For each road point in the refined image, we find the set of parameters \( \Theta \) and \( \rho \) for all lines passing through that point. We then keep track of how often each parameter is requested in a big table (sometimes called the Hough map).

After processing all the road pixels we collect those parameters that have occurred often enough (Experience showed that keeping all parameters that occur more often than 85% of the most
popular one, gave very pleasing results). We however want quantize the Hough map to make sure that we do not represent the same line twice.

![Hough transform on the rectified image](image)

Figure 8: Results from taking the Hough transform, the Hough map

Using Thresholding on the Hough map we keep only the line parameters that are common enough.

### 3.2 Finding the crossroads

With the roads as a simple mathematical formula, we can easily calculate the crossroads with the aid of a little linear algebra. For all pairs of line parameters lets call them \( \rho_1 \) and \( \Theta_1 \) for line 1 and for line 2 \( \rho_2 \) and \( \Theta_2 \), the intersection of these two lines is given by.

\[
P = \begin{bmatrix} \cos(\Theta_1) & \sin(\Theta_1) \\ \cos(\Theta_2) & \sin(\Theta_2) \end{bmatrix}^{-1} \begin{bmatrix} \rho_1 \\ r \rho_2 \end{bmatrix}
\]

If the determinant of the matrix is not zero (or close to that) we calculate the intersection and if its inside the image we save the intersection as a potential intersection point.

Although we have been very careful in making the Hough map, its clear that we will always have lines that are slightly angled versions of other lines, after all roads are not infinitely thin, so angled versions will always creep in.

So some filtering is needed to correct for this. We start by sorting the nodes in descending order, by how many neighbors they have on a 2 pixel radius. Then simply going down the list and killing all the neighbors of each potential crossroad leaves us only with a handful of potential crossroads, and as we can see in the picture, usually a well positioned one since the one we keep have the most neighbors of all the previous ones.
Figure 9: Potential crossroads overlay-ed with the test image

We note that we have slightly more crossroads than actually are in the image, but they are well positioned usually in the center of the intersection.

4 Detecting paths

Virtual cars are the weapon of choice when finding the paths between individual nodes in the graph, now that we already have the nodes at hand, all we need to do is to send virtual cars from each node in all directions and see where they go, but first lets look more closely at the virtual car it self.

4.1 A virtual car

A virtual car is sadly without a V-8 233 HP engine, but under the hood so to say we store its position in pixels, and where its heading at any given step. In each step we evaluate the next potential pixel, by finding out if its close enough to the color of the starting position. If that’s the case we continue in that direction. ²

However if the color value is not close enough we first try tilting to the right and try then the pixel straight ahead, if that does not work we try rotating left, and then slightly more to the right and so on. Here is a short block diagram that shows how this works.

Figure 10: Block diagram for the virtual car

²Colors are compared like one would compare two vectors, since after all in each pixel we store three numbers representing the color in a color vector C. We compute the norm of the difference between the color at the starting position C_s to the color of a potential destination C_d and compare that number to a threshold, or mathematically we are close enough if ||C_s - C_d|| < 35 the threshold value of 35 was found as always via experimentation
4.2 Constructing the graph

We now drive virtual cars in all directions from all nodes to find out where they go, stopping if they come close enough to another node (within 2 pixels), they wander out of the image or get stuck backtracking to often over the way the have already driven (Currently they are allowed 3 “backings”).

If they stop at another node, we store the distance the car has traveled as the distance between the starting node and the node it wound up at. For simplicity we store the graph as a matrix, since we found 8 nodes during our test runs, this becomes an 8 by 8 matrix where the $a_{ij}$ element describes the distance in traveling from node $i$ to node $j$.

For our test image, the resulting matrix becomes.

$$G = \begin{bmatrix}
0 & 0 & 12 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 4 & 5 & 7 & 0 & 0 \\
18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 16 & 0 \\
0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\
15 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 & 0 & 30 & 0 & 0 \\
\end{bmatrix}$$

With the following nodes.

<table>
<thead>
<tr>
<th>Node Number</th>
<th>vertical coords</th>
<th>horizontal coords</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>33</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>26</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td>24</td>
</tr>
</tbody>
</table>

Since this is not very humanly readable, we draw the graph, overlaying the test image.
5 Conclusion and future work

As always with image processing, it is very time consuming to come up with the optimum setting for all the different filters and routines one runs. It would be nice to automate this process for this case, since the author spend considerable amounts of time, trying to tweak the parameters in the refining process to give the correct results, and also often these parameters are not the same for all images, say for example on a rainy day asphalt usually becomes much darker than on a dry day. Also various aspects of the system need work if a robust system is to be developed. Currently I've concentrated all my settings to work on the given test image of downtown Reykjavik.

Despite its shortcomings the system has demonstrated that the Hough transform is extremely useful in image processing as a very powerful tool to detect lines and get them into a mathematical form that can easily be used to extract information from the test image, like the graph of a road structure.

As simple as they are, virtual cars are quite powerful, for example note how the left road on the test image looks wider than the right road. Virtual cars catch this quite nicely since it takes more effort to squeeze a virtual car through a narrow and a crooked street that a straight wide road (Backing up and turning the car guarantee that).

The views expressed here are only those of its author and do not necessarily represent the views and opinions of others.

References


9