Digitize City Roads Structure using Image Process Technology

ShanghaiTech University, SIST
Wang Yihan 46710811
Zhuang Ziwen 14872434
Wang Letong 55343321

***Abstract*—By applying image processing technology, we successfully digitized the graph information of the road in a city. In specific, we generated a graph structure to represent the roads in the city from the given information. Also, this method used only the position information, as well as lower the noise of the useless points, which lower the cost of constructing a road digital information in a region from the distribution of the traffic data.**

I. Introduction

In this project, we use the one-day order data of Didi Company for road reconstruction. In this process, we try to build the roads based on the track of the orders, show them in a plot and find the necessary roads.

Before seeing the details of the project, let's have a look of assumptions we made to promise our ideas work. First, we assume that the ground is flat, although the earth is spherical. Second, we consider the data points as continuous, which means we can use the continuous data can build the continuous road. Third, we assume that the data points are evenly distributed.

In this work, the data we have is limited, and the most efficient information is position information, followed by the time information. If we want to do some work with both position information and time information, it is difficult to deal with it in a limited time.

The position information is relatively concentrated, which is distributed in one city, representing the road cars passing by. Using the track, we can get all the cars' track and form a figure. As a result, we can verify the figure and deal with it to digitize the roads and intersections. After observing the figure simply, we think building the road in the city feasible, and begin our work.

II. Preparation

In order to find the image coordinates corresponding to points' latitude and longitude, we need to determine the latitude and longitude range of the given data of orders. So we traverse the data and find four points which represent the biggest and smallest latitude and longitude.

After knowing the range of the footprint, given the resolution of the map(x pixels * y pixels), we can construct a foot print matrix. Although the earth has a spherical surface, the city is very small compared to it. So we assume the city has a plat surface, and directly use the difference of latitude and longitude to represent the distance between two points.

By calculating, every geographic coordinate is corresponded to an index of the matrix. Every entry of the matrix represents how many footprints fall into this index.

III. Extracting the image

A. Plotting Dots

In order to reconstruct the city's roads into a digitalized graph, we added all the longitude and latitude points as footprints. By marking all the footprints onto a image (represented as a matrix), we got a general traffic map of the city.[1]

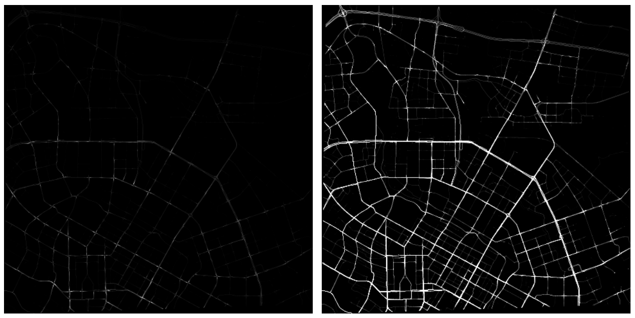

Fig. 1. The footprints

To meet the needs of constructing the digitalized graph of the city, we need to make the computer recognize the crossroads and the roads displayed on the image, which are nodes and edges in the graph. Initially, we assumed that the footprint density is uniformly distributed on the image, so that the value where the intersections at would be twice of that where the roads at. This would make the recognizing process easy enough to pursue.

In the first glance of the image, we found that the footprint density were much higher when they were at the waiting track[2] of the crossroad, but not the intersection of two roads. And ordinary footprints were quite lower than the values at the waiting track. To make the image more visible, we applied a binary method to display the image only in 0 or 1, which could make the image clearer for us to decided how to recognize roads and intersections.

B. Image Enhance

By analyzing this binary image, we discovered that the greatest obstacles which kept us from recognizing the roads

[1]The picture on the right is the initial image which is enlightened to make the marks more visible.

[2]Waiting track means the regions where most of the vehicles were waiting for the traffic lights to turn green, so that they could keep moving.

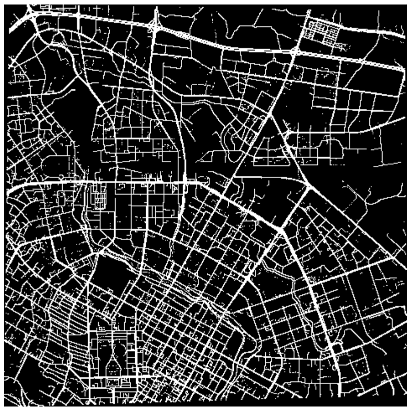

Fig. 2. The image plotted in only 0 or 1

and intersections are those noise, which seem like the roads on which too few cars went. Thus, the points are quite discontinuous, and they are supposed to be treated as noise. According to the works by Jinhua Yu, Yuanyuan Wang and Yuzhong Shen[1], there is a nonlinear diffusion method which helps smooth the noise in the image.

According to the Anisotropic Diffusion method, we define

$$\begin{cases} \frac{\partial I}{\partial t} = div\left[c(||\nabla I) \cdot \nabla I\right] \\ I(t=0) = I_0 \end{cases}$$

In the above formula, it applies gradient methods to smooth the image so that the noise will not be so distracting for the crossroad recognition. I denotes the pixel value of the image, If added subscript, I_p for instance, it means the exact value in the position p of the image. div is the divergence operator. c is the "edge-stopping" function or diffusion function. And I_0 in specific is the original image.

Based on the Anisotropic Diffusion process produces an image with less noise from a set of gradually smoothed image I^t where t is the index of the iteration of the image. By solving the above equation, the gradient of each iteration can be solved and be added to each iteration of the pixel value.

$$I_p^{t+1} = I_p^t + \frac{\lambda}{|\eta_p|} \sum_{q \in \eta_p} c\left(\nabla I_{q,p}^t\right) \nabla I_{q,p}^t$$

Since we still cannot developed a method that can determines whether the smooth process should be stopped. We tried different iteration types

After checking performance of different iterations, we selected the 60 iteration as the best performance.

IV. Construct and Digitalize the graph

A. First Attempt

Have gotten a complete plot, we tried to identify the nodes and edges in the image.

1.Hough Transform[2] Algorithms(used for identifying edges)

In a two-dimension plane, a straight line can be represented in this form: $\rho = xcos\theta + ysin\theta$. θ represents an angle which is formed by the vertical line and horizontal line measured in counter-clockwise.ρ is the distance from the origin point to the line. A (θ,ρ) pair is a point in Hough Space. Given a point in a plane, every edges going through the point is on a sine curve in Hough Space. So the problem of finding straight lines in a plane is transfered to finding curve intersections in Hough Space.

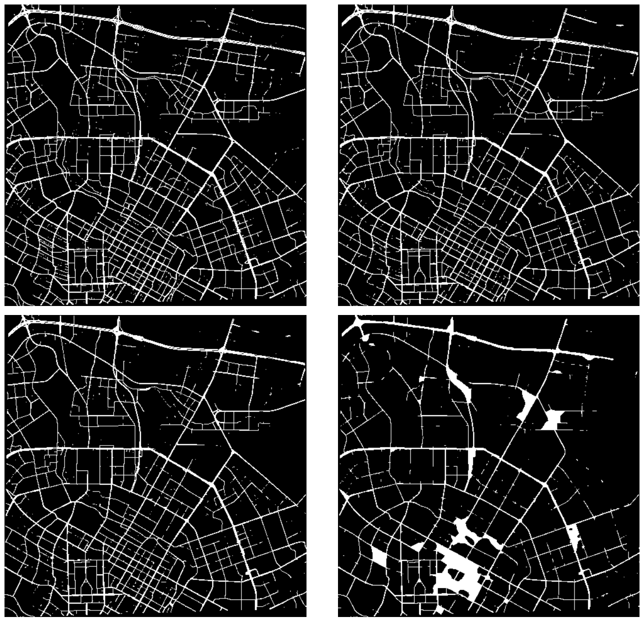

Fig. 3. Iterations with (20, 60, 100, 1000)

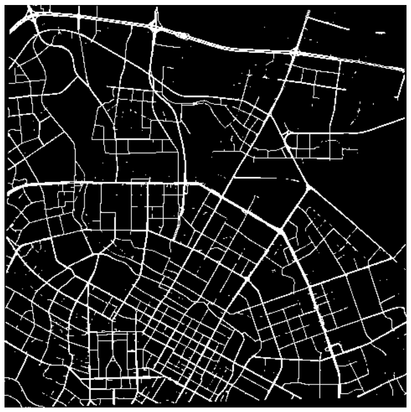

Fig. 4. Performance we chose (60 iteration)

We used the OpenCV in python to implement the Hough transform, but unfortunately this method didn't work. The functions in OpenCV python can not decide the start and end points of the lines and this function can only fit the straight lines but not the curved lines.

2.Harris: A Combined Corner and Edge Detector(used for finding nodes)

The method can find all the difference in intensity for a

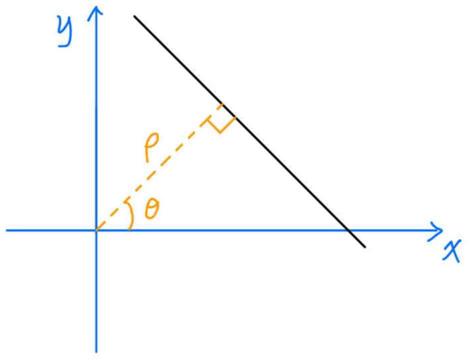

Fig. 5. polar representation of a line

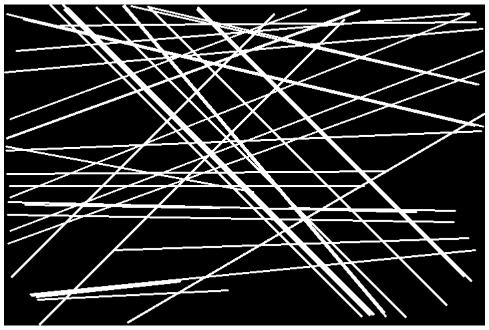

Fig. 6. A failed example

shift of (u,v) in all directions. This is expressed as below:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

This window function gives weights to pixels.

$$E(u,v) = \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

where

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

And the I_x, I_y represent the derivatives of x or y direction. In this way, we can get a equation to represent the ”scroe”, which can determine if there is a corner in the window.

$$R = det(M) - k(trace(M))^2 = \lambda_1 \lambda_2 - k(\lambda_1 - \lambda_2)^2$$

where λ_1 and λ_2 are eigenvalue of M.

if $|R|$ is small, it means, λ_1 and λ_2 are small, so the point is in a flat region;

if $R << 0$, it means $|\lambda_1 - \lambda_2|$ is great, so the point is on an edge;

if $R >> 0$, it means λ_1 and λ_2 is almost the same, so the point is on a corner. As a result, the Harris Corner Detection can give these scores to identify the corners.

We use OpenCv’s function $cornerHarris()$ to implement this method. The function has 3 parameters:

blockSize: the size of neighbourhood considered for

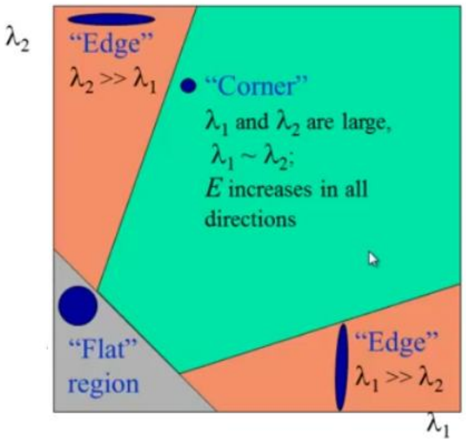

Fig. 7. Harris Corner Detector

corner detection

k: the k in the above equation calculating R(it is usually between 0.04 to 0.06)

ksize: Aperture parameter of Sobel derivative used

After trying different values for each parameter, we choose the best parametric combination.

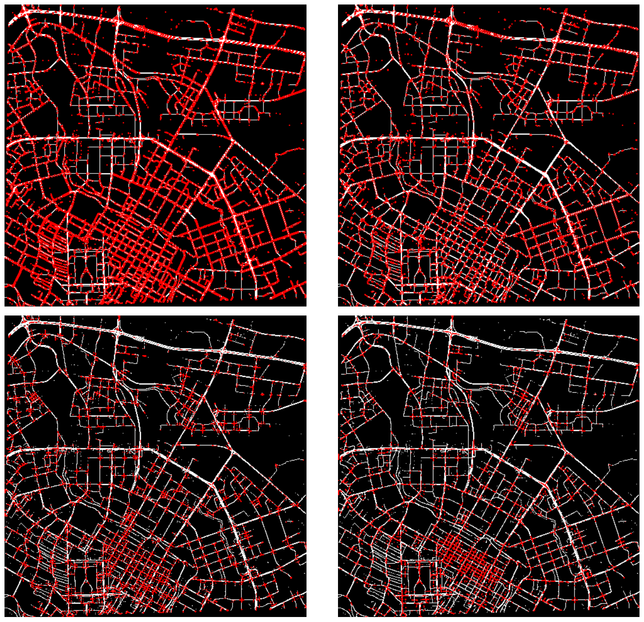

Fig. 8. ksize with (1, 3, 15, 29)

B. Clustering

After getting all the nodes, we found that there were several noise nodes off the roads and existed several points among one intersection. So we should deal with the noise and put the points in one intersection together. We calculate the Huffman distance to represent the distance between two nodes, and if the distance was less than a threshold value, we regard them as one node and put the coordinate points in a same set, where the sets are represented as a disjoint set format. After traversing the point list, we had known that which nodes in fact could be considered as one node in the intersection. Then we calculate the average values among the nodes in same sets to get a median to represent a intersection.

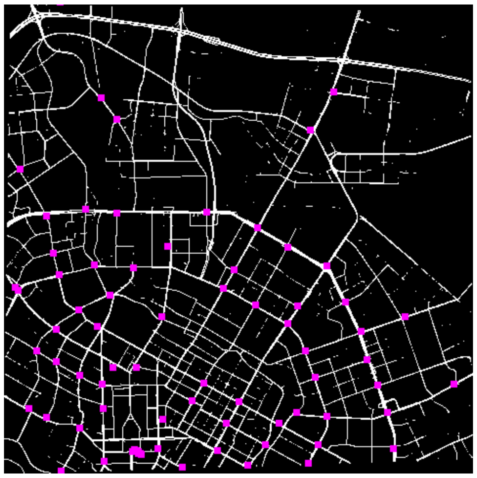
Fig. 9. Clustered nodes in the image

C. Edge Verification

After recognizing the nodes in the image, we verified the roads that connect each pair of intersections directly by moving through the image and if most[3] of the pixels that lied on the path are marked as footprints, where the path between two node is a straight line that connects two nodes directly. To measure whether there are a straight line connect the two nodes directly, we generated a straight line formula based on the two coordinates. Thus, we can acquires all the coordinates of pixels on the straight line. Then, by moved the testing patch along the straight line, we could test whether the patch contains enough highlighted pixels, which means there are footprints on the path. If almost all the patches get the result of being on the highlighted pixel, this two nodes are connected directly by a straight line.

D. Graph class

In this time, we had all the information about the nodes, edges and the relationship.To store the nodes and edges in the map, we created a graph class which can be used to store and do some calculation.
1.Create a node
We should put each node in the graph one by one, and the graph class, which was an undirected graph, was used for store them in an array.
2.Build the roads
After add all the nodes in the graph had been added, we began to build the relationship among them. Using the information provided by the "edge verification" part, we would put in a pair and connect them in the graph, which means putting each other node's index in the node's adjancent list.

In this part, we had a fuzzy operation. Owing to the little error occured when detected the nodes, the same may have different expression.
3. Judge if there is a road between two nodes.
This part is a function of our work. If the input were two node coordinates and need to inquiry whether there is a road between them directly or not, we should traverse the adjancent list until one node which is same as one of the input node was found. Then we should detect the neighbor nodes recorded in the adjancent list. If there was another node in it, it means that the two nodes were actually a pair of neighbors.
4. Sort the shortest path between two nodes.
This is an application of the graph data structure. Given two intersections, it could find the shortest path between them, using the A* search algorithm with linear distance as heuristic function.

V. Applications

In this work, we can tell the intersections and roads in the plot, which means they are represented as a digital form.

1) Digitize the road information at low cost: There are still many regions that have no digitized traffic information that can be processed by computer. Using this technology, companies can get how the roads are distributed on a rural area or those countries that have no public traffic services, considering mobile phone location can be also used as footprints of how the citizens moved in the region.

2) Show traffic volume: Different roads have different traffic volume, and the density of the orders can represent the traffic volume. Through the information, we can tell which roads are often passed by, which may be the more important roads. So we can widen the more important roads or add more public traffic routes.

3) indicate traffic jam and find the quickest rout between two points: Now, we only use the GPS data without time stamp data. Combined with time stamp, we could indicated the speed of motor vehicles, which could be added to the graph structure. With velocity information, the digital graph could also indicated traffic jam, and help to find the quickest rout between two points.

VI. Discussion

A. Improvement

During the process of finishing this work, we also have some ideas to improve them, but limited by time, we can't implement them. We write down these points for further improvement.
1. Not all the nodes are detected by our algorithm.
In the Harris edge detector, the noise point should be ignored so the threshold set in the code should increase. However, the higher threshold value would lead to some corners can't be detected.

[3]By most, there is a function to measure the amount of pixels that meet the requirements

The Harris edge detection algorithm is find the nodes through the corners, but some corners are too smooth to be regarded as a corner. As a result, the algorithm doesn't consider them as the corners but as some lines. Also, what need to be detected are actually the intersection between two lines. (possibly not straight lines) Thus, the detector might recognize some of the big crossroad as many corners, or many intersections.

2. Some noise can't be moved.

There are some order track off the roads, which we regard them as noise. We have tried several methods to remove them, and most of the noise can be removed, but not all the noise. These noise points could have a bad effect on our following work.

3. Some points may be clustered wrong.

The map is detailed, and some Huffman's distance between two close intersections is very small, even smaller than the distance between two points in a same big intersection. In this situation, owing to the limit of the clustering algorithm, we can't tell the nodes on the very close two intersections, and regard them as one node.

VII. Direction

A. Graph class member functions

1. create_node: This function is used to create a node storing in the graph. Input is the coordinate of the node.
2. create_road: Create the relationship between two nodes meaning that they are connected. Input is two nodes' coordinates.
3. equal:Judge if two nodes are in a same intersection.
4. find_neighbor: Find all the directly connected nodes to the input node. The input is the node's coordinate.
5. find_node:Given a node's coordinate, the function would find the node in the graph. If a node is found, the function would return the index in the graph's adjacent list. Otherwise, it would return -1.
6. path: input two pairs which represent the coordinates of star point and the end point, it will return an array containing the intersections the shortest path going through.

B. Matrix definition

By operating on the image which is represented on a matrix, we have to define our own matrix class and perform the image processing operation. In general, the most important information is how to match the row-column index to the x-y coordinates. Considering x-axis is always on the clockwise direction of the y-axis, so as the row index and the column index. we defined the matrix locating method as follows.

Under this definition, we can locate matrix value by indexing as x and y coordinates.

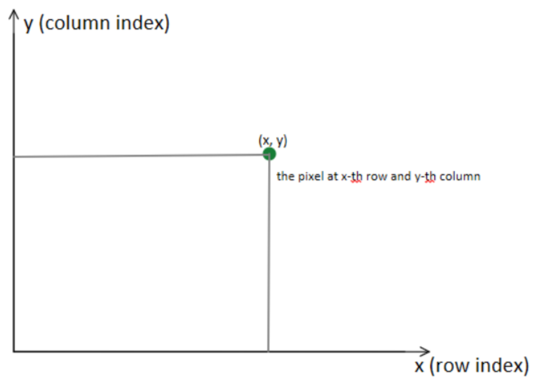

Fig. 10. Matrix axis definition

C. Image processing methods

All the Image processing methods that we have tried are in the 'imageProcess' file, which includes the edge detection kernel and the sharpen kernel to perform the convolution, KNN removal process to eliminate the noise points that could disturb the crossroad recognition and Anisotropic diffusion technology to finally smooth the image.

D. Image output library

According to the protocol of the Netpbm format, we implemented the method to write to file by ourself. So that the matrix can be displayed directly locally.

E. Perform the demo

1) Make the executable:

Since the data is too big, we save the footprints in a matrix file. But there are still some other operations that requires the data. Thus, you need to put two data file under the data folder. Please be aware of the file name has to be exactly 'gps_20161101' and 'order_20161101'.

2) Compile the file:

1. Linux environment

Simply move your directory to the code folder, then simply type make

2. windows environment

You need to have your visual code installed and be available to compile normal file. By open the project folder using visual code, press 'ctrl+shift+B', the editor should compile it self.

3. Use the demo

Open the terminal or power-shell and move to the code directory. There should be an executable available. Take Linux for example, type ./main.out

There would be a prompt displayed on the screen that tells basically what the program will do if you type the following thing as the first parameter in the command line.

By executing the demo, type ./main.out demo The process is displayed on the screen, and then it will shows how many nodes are recognized connected. And you should input which nodes you want the program find connected and the path. Then you can find three file under the /demo folder. And you can see the nodes that are in the path way and those nodes recognized by the program.

References

[1] Yu, J., Wang, Y., & Shen, Y. (2008). Noise reduction and edge detection via kernel anisotropic diffusion. Pattern Recognition Letters, 29(10), 1496-1503. doi:10.1016/j.patrec.2008.03.002

[2] Hough transform. (2018, June 19). Retrieved June 15, 2018, from https://en.wikipedia.org/wiki/Hough_transform

[3] OpenCV: Harris Corner Detection. (n.d.). Retrieved from https://docs.opencv.org/master/dc/d0d/tutorial_py_features_harris.html