MVA'94 IAPR Workshop on Machine Vision Applications Dec. 13-15, 1994, Kawasaki

ON THE EXTRACTION OF VARIOUS REGIONS IN VECTOR MAPS

Guofang Jiao, Eihachiro Nakamae, Katsumi Tadamura
Hiroshima Prefectural University, 562 Nanatsuka, Shobara 727, Japan

Hiroyuki Inuyama
Sanei Giken Co., LTD, Maple Bldg 4F, Hiroshima 730, Japan

ABSTRACT

On a topographical map of civil engineering, there are various enclosed areas, for instance, rice fields, wild fields, buildings, roads, walls, regional boundaries and so on. Before a software system such as road planer is run, it is necessary to extract these various regions and features, and to transform them to 3D data. The automatic extraction and classification of all of them on a screen are difficult and very time-consuming. It is better to combine the automatic recognition with interactive operation. It is obvious that interaction is easily done if vector data of maps is applied. On the other hand, the vector data is much less than the raster data of the same map. This paper proposes a practical solution for understanding of vector maps, including two major methods; ditching (DIrected Track stretCHING) method for open regions (e.g., roads, slopes and walls etc.) and inward tracing method for bounded regions. (e.g., various fields)

INTRODUCTION

There are many feature recognition techniques on the understanding of engineering drawings and topographical maps[5-6], especially, state transit[1], morphology and hough transformation[2], and MAP (multi-angle parallelism) [3] etc.. However, they are mostly based on a map's raster instead of vector data. We think that it is important to extract various regions in vector maps automatically because, in most cases, the extractions of various features have to be executed after the vector data is transformed from related raster data by applying a universal program. It is reasonable to search for such enclosed areas as rice fields, vegetable fields, meadows, buildings, roads, walls and other surrounding regions step by step since such a large and time-consuming task as understanding of a map had better to be divided into small subtasks in accordance with their different features. That is easily done by interaction if based on the vector data. After the vectorization of raster data, of course, the omission of some significant information and the addition of some noises increase the difficulty in the extracting of the features. However, the recognition on the raster data takes much more memory space than that on the vector data does, and the former is much slower. On the other hand, it is more easy to do interactive operation such as picking a line on the basis of vector data because the interaction is very important in a practical software of the understanding of topographical maps [4]. So the recognition on the vector data is also very useful.

In general, as shown in fig.1(a) which is a vector map with Japanese legends (the same as the following), a map consists of many parts with certain areas and specific meanings. Every part here is defined as a **enclosed area**, even though the boundary is not closed, for example, rice fields, meadows, buildings, roads, rivers, slopes, mountains, towns, vegetation and regional boundaries etc.. According to respective geometric features, they can be divided into four classes:

- **Open symbol-like regions**, consisting of relevant symbols that form long and almost parallel areas, for instance, slopes, various walls etc..
- **Open bounded regions**, such as roads, whose boundaries are not closed and almost parallel.
- **Bounded regions**, containing zero, one or more significant symbols. Such areas as fields, buildings, and the other surrounding regions are specified by their boundaries. We assume that there should be one or more symbols inside a bounded region to be searched. When there is no symbol, for instance, in built-up areas, in regional boundaries and in some fields where the symbols are omitted, a related symbol must be added to the region by interaction. That is because specifying the location and the type of a region is the main purpose of the paper.
- **Non-bounded regions**, for example, mountains etc. consisting of contour lines.

Our methods for the understanding of maps combine automatic extraction with interactive operations. For example, contour lines, character strings, and so on are indicated by interaction. The automatic recognition of open regions is derived by interactive indication of the initial conditions. And, in order to reduce interference and speed up the search for the bounded regions, a **search range** (e.g., a rectangle or a polygon region) surrounding all symbols and polylines contained in the bounded regions to be searched, is indicated on the screen by interaction. Thus any lines outside the indicated range are ignored.

The procedure for the understanding of a vector map is divided into the following five phases.

- Interactive indications of contour lines and character strings.
- Extraction of open regions.
- Interactive insertion / indication of some symbols in the case of no symbols inside the bounded regions and/or of the symbols inside the bounded regions not being able to be recognized automatically.
- The search for the bounded regions that invokes the recognition of things like solid lines, dotted lines, dashed lines, and the extraction of symbols.
- The modification of the results if necessary and the assignment of such attributes as altitudes by interactive tools.

This paper emphasizes two major methods, ditching (DIrected Track stretCHING) method for open regions (e.g., roads, slopes and walls etc.) and inward tracing method for bounded regions. (e.g., various fields and regional boundaries) resp..

DITCHING FOR OPEN REGIONS

The open regions such as roads have the following characteristics which become the basis of and starting point for the method proposed in this paper.

. Their border lines are almost parallel and/or not closed.
. The local curvatures of the border lines are not so large.

The ditching (DIrected Track stretCHING) method consists of two actions; interactive initialization and automatic recognition. The interactive initialization includes the following items:

. indication of two segments whose purpose is to inform the automatic recognition from where it should be started.
. input of zero, one or two lines called cut-lines whose purpose is to inform the automatic recognition where it has to stop (e.g., **t1** and **t2** in fig. 2)..

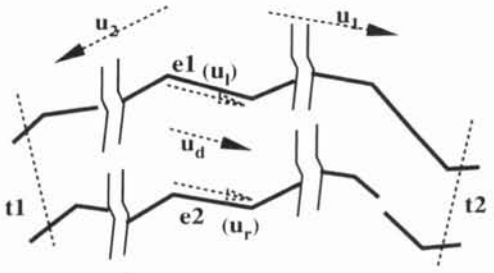


Fig. 2 Ditching Method

If the user gives initial conditions by arbitrarily and interactively indicating two segments (e.g., **e1** and **e2**) belonging to the two border lines as illustrated in fig.2, then the automatic procedure of the ditching method starts from these segments (called initial tracks, left side-track and right side-track resp,) and follows in order and advances step by step firstly in the direction u_1, and after that, in the opposite direction u_2. Thus the tracks are continously stretched until the terminal condition is met.

Each step of the ditching method includes the following operations:

. **SelBehind** operation plays the role of the selection of the track which lags behind. We call the track a behind track. The work is to check which of two side-tracks lags behind in the direction of u_d.
. **ExtBehind** operation is to find a candidate segment in the side of the behind track in accordance with certain evaluation rules and to connect the behind track with the candidate. The detail of the operation is explained in the first following subsection.
. **ChkTermin** operation is to test whether the terminal conditions are met or not. The terminal conditions may be the following cases:
  . no candidate segment.
  . the track intersects a line segment which is not candidate for stretching the track, such as contour line and cut-lines input by interaction at first.
. **RevDatrack** operation is to revise the datum-track which is a bisector line of a trapezoid constructed by two side tracks.

After an open region is extracted, there is an operation called **ExpBranch,** the function of which is to explore a potential branch of the sought open region based on a couple of points at a fork. And then, the above steps are invoked to search for the branch. As a result, a lot of open regions may be sought at the same time, so this operation is very convenient for users.

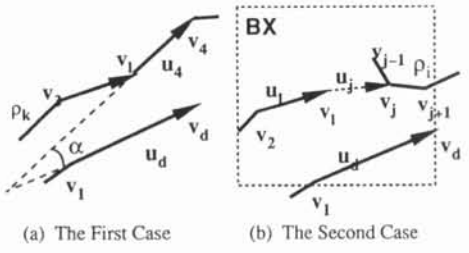
ExtBehind Operation

The following two cases exist for this operation. Let's assume that the left side-track is the track which lags behind.

Case 1. If the point v_1 on the track is not an end vertex of a polyline ρ_k in the direction of track stretching (see fig.3(a)), then the next segment u_4 from v_1 to v_4 is chosen for the extension of the lagging track if the following conditions: is satisfied.

b1. The absolute value of a directed angle from u_4 to datum-track (u_d) is not larger than a constant **Centd** (e.g., $\pi/10$) given by the system.

b2. u_4 does not intersect the extension of u_d.



(a) The First Case (b) The Second Case

Fig. 3 **ExtBehind** Operation

Case 2. Otherwise, as shown in fig. 3(b), for every polyline ρ_i inside and across **BX** which is rectangle defining the search range dependent on the scale of the map, if ρ_i has never been chosen for the extension of a side-track and the following conditions are satisfied, then vector u_i from v_1 to v_j is selected for the extension of the lagging track. Here $\mathbf{Cd}(u_j)$ is a directed angle from u_j to u_d

c1. Vertex v_j is inside **BX**;

c2. v_j and v_1 are in one side (e.g., the left side) of u_d;

c3. $-\mathbf{Centd} < \mathbf{Cd}(u_j) < \mathbf{Centd}$;

c4. $\mathbf{Cd}(u_j)$ is the largest (for reference, in case of **open symbol-like region**s, the absolute value of $\mathbf{Cd}(u_j)$ is the smallest).

The lagging track (e.g., the left side-track) is extended to v_j , and the two points v_1 and v_j are stored as a fork of a potential branch of the open region which is being sought.

In the above two cases, if the extension of a side-track fails, the automatic procedure is terminated in current direction of track-seeking, and then follows in the opposite direction. If the extension of a side-track is successful, the check for the terminal conditions must be executed.

RevDatrack Operation

After the **ExtBehind** operation, the datum-track should be also extended and revised for subsequent track stretching. This operation consists of two suboperations, as demonstrated in fig.4; here the (left) side-track is extended to v_j and the right side-track (u_r) is from v_1 to v_r.

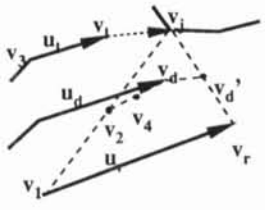


Fig. 4 **RevDatrack** Operation

. The datum-track is extended to $\mathbf{v_d}'$, where $\mathbf{v_d}'$ is the midpoint of $\mathbf{v_j}$ and $\mathbf{v_r}$;

. $\mathbf{v_d}$ is moved to $\mathbf{v_4}$, where $\mathbf{v_4}$ is the midpoint of $\mathbf{v_d}$ and $\mathbf{v_2}$, and $\mathbf{v_2}$ is the midpoint of $\mathbf{v_j}$ and $\mathbf{v_1}$. The suboperation is designated to balance the influences of the two side-tracks.

ExpBranch Operation

In the **ExtBehind** operation, all pairs of points at forks in open regions such as roads are stored. These points are in order so that it is possible to determine in which side the branch is. In order to explain the following methods clearly, we call the open region that has already been extracted the sought region and its corresponding tracks the sought tracks. In the meantime, we term both the open region which is being extracted the current region and its corresponding tracks the current tracks.

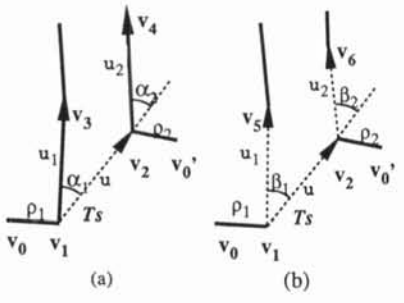


Fig. 5 **ExpBranch** Operation

Let's assume that the pair of points is $\mathbf{v_1}$ and $\mathbf{v_2}$, $\mathbf{u}$ is a vector from $\mathbf{v_1}$ to $\mathbf{v_2}$, and a part of the sought side track ***Ts*** on which both $\mathbf{v_1}$ and $\mathbf{v_2}$ are is a polyline from $\mathbf{v_0}$ to $\mathbf{v_0}'$(see fig.5). The exploration of potential branch is done in the following steps.

. pick one polyline ρ_1 on which $\mathbf{v_1}$ is located.

. pick the other polyline ρ_2 on which $\mathbf{v_2}$ is located.

. for ρ_1 and ρ_2, follow the same procedure as below.

- in the left side of the track ***Ts*** on which $\mathbf{u}$ is, find a vertex $\mathbf{v_3}$ ($\mathbf{v_4}$) next to $\mathbf{v_1}(\mathbf{v_2})$ on $\rho_1(\rho_2)$ under the condition: $\pi/6 < \alpha_1(\alpha_2) < 5\pi/6$ (see fig.5(a))
- If $\mathbf{v_3}(\mathbf{v_4})$ exists, let $\mathbf{u_1}(\mathbf{u_2})$ be a vector from $\mathbf{v_1}(\mathbf{v_2})$ to $\mathbf{v_3}(\mathbf{v_4})$; otherwise, in the left side of the track ***Ts***, find such a vertex $\mathbf{v_5}(\mathbf{v_6})$ on one of other polylines around $\mathbf{v_1}(\mathbf{v_2})$ under the conditions: $\pi/6 < \beta_1(\beta_2) < 5\pi/6$ and $|\beta_1(\beta_2) - \pi/2|$ is the smallest (see fig.5(b))
- If $\mathbf{v_5}(\mathbf{v_6})$ exists, let $\mathbf{u_1}(\mathbf{u_2})$ be a vector from $\mathbf{v_1}(\mathbf{v_2})$ to $\mathbf{v_5}(\mathbf{v_6})$.

. If both $\mathbf{u_1}$ and $\mathbf{u_2}$ exist and the following conditions are satisfied, $\mathbf{u_1}$ and $\mathbf{u_2}$ are the two initial side tracks of the branch.

$\pi/4 <$ the angle between $\mathbf{u_1}$ and $\mathbf{u_2} < 3\pi/4$;

$\mathbf{u_1}$ doesn't intersect $\mathbf{u_2}$.

After the initial side tracks of the branch are obtained, the automatic procedure of ditching method is executed to search for the branch. After the branch finishes, the next potential branch is explored. The procedure is repeated until all pairs of points stored in **ExtBehind** operation have been explored. As a result, a lot of branches may be extracted automatically.

INWARD TRACING METHOD

We assume that there is at least one symbol inside every bounded region. We define the center, **CO**, of the symbol as the **origin** of the corresponding region. The first polyline that a ray from the origin meets is named as the **closest polyline**(e.g., ρ in fig.6). It is undoubted that each polyline as a part of the boundary of the corresponding region is the closest one. The closest polyline as a part of the boundary is called a **valid candidate**. Each valid candidate has two vertices, defined as **connectors**, one is called the **clockwise connector** and the other the **counter-clockwise** one. For example, in fig.6, $\mathbf{v_c}$ is the clockwise one on ρ, $\mathbf{v_2}$ is the counter-clockwise one on ρ'. The clockwise connector (e.g., $\mathbf{v_c}$) of one valid candidate (e.g., ρ) connects with the counter-clockwise connector (e.g., $\mathbf{v_2}$) of the other (e.g., ρ') when the two valid candidates are connected.

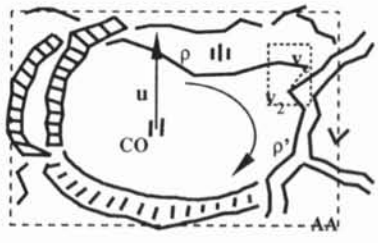


Fig. 6 Inward Tracing Method

Before the search for bounded regions, a search range (**AA**) such as a rectangle or a polygon should be indicated by interaction in order to reduce the interferences. And then, for every symbol inside the range, it is possible to search for the corresponding boundary. As illustrated in fig.6, the method is started from an initial valid candidate, ρ, which is a polyline that a ray from the origin of the symbol meet firstly, and goes in a clockwise and then counter-clockwise direction until the boundary of the bounded region is closed or the search fails.

Here the word "Inward" has two meanings: every valid candidate is closest polyline; the sum of two weighted angles (e.g., α_{i1} and α_{i2} in fig.9) is smallest when next valid candidate is traced from the current one. An initial valid candidate (ρ) is selected as the current valid candidate at first. Then, for current one, the following recursive procedure is executed in clockwise and counter-clockwise direction so that a series of valid candidates forming the boundary of the region are traced out.

```
If the clockwise (counter-clockwise) connector of the current valid
    candidate has not been traced out {
    trace the clockwise (counter-clockwise) connector;
    check whether there is a closest polyline relative to the ray from the
      origin to the connector.
    If true let the current valid candidate be the closest polyline;
    else {
        search for the next valid candidate under the meanings of "Inward";
        if next valid candidate exists {
                connect the current valid candidate and the next one;
                let the current one be the next one;
        }
    }
}
```

Finally, it is necessary to test whether a series of valid candidates construct a closed boundary or not, and whether the boundary surrounds its related symbol. If so, it means that the search is successful; otherwise, the search is a failure.

In the algorithm described above, there are two key operations: tracing a connector and searching for the next valid candidate, explained in following subsections.

Tracing a Connector

In order to describe the tracing method of a connector clearly, we introduce another concept called a **tangent point**. The tangent point, $\mathbf{v}_t$, is a vertex on a valid candidate, ρ, satisfying the following conditions (see fig.7).

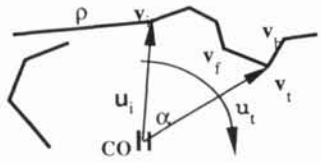
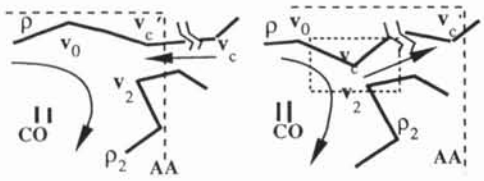


Fig.7 Tangent Point



(a) Outside Case (b) Inside Case

Fig. 8 Tracing a Connector

. Both $\mathbf{v}_f$ and $\mathbf{v}_b$ neighboring to $\mathbf{v}_t$ locate in one side of $\mathbf{u}_t$,

. Every vertex $\mathbf{v}_i$ on ρ with acute angle α is located on the same side as $\mathbf{v}_f$ of $\mathbf{u}_t$. Here α is the angle between $\mathbf{u}_t$ and $\mathbf{u}_i$, which is a vector from **CO** to $\mathbf{v}_i$

As shown in fig.8, if ρ is the current valid candidate in clockwise search, the clockwise connector $\mathbf{v}_c$ of ρ is determined by the following conditions and steps. Let's take $\mathbf{v}_0$ as a starting point on ρ.

. $\mathbf{v}_c$ is a tangent point or an end vertex of ρ if there is no tangent point . Here, $\mathbf{v}_c$ is clockwise (or counter–clockwise if in counter– clockwise search) from $\mathbf{v}_0$ relative to **CO**.

. If $\mathbf{v}_c$ is outside **AA**, let $\mathbf{v}_c$ be the first vertex $\mathbf{v}_c$' on ρ going entry to **AA** (see fig. 8(a)).

. Otherwise, if $\mathbf{v}_c$ is inside **AA** and not an end vertex of ρ, test if there is a next valid candidate, ρ_2, around $\mathbf{v}_c$. If not, let $\mathbf{v}_c$ be the next vertex going far from $\mathbf{v}_0$, this step is repeated until $\mathbf{v}_c$ is an end vertex of ρ or last vertex $\mathbf{v}_c$' inside **AA** (see fig.8(b)).

Search for Next Valid Candidate

We suppose that ρ is the current valid candidate, as shown in fig.9. If there is the closest polyline related to the ray from the origin to $\mathbf{v}_c$, then the closest polyline is selected as next valid candidate. Otherwise, every polyline (e.g., ρ_i) inside or going through a certain search range **BX** given by the system is checked on the following conditions and steps:

Find a point $\mathbf{v}_i$ on ρ_i nearest to $\mathbf{v}_c$ under which $\mathbf{v}_i$ is a vertex inside **BX**, or a perpendicular foot point $\mathbf{v}_j$ from $\mathbf{v}_c$ to segment **SE** of ρ_i if **SE** goes through **BX**.

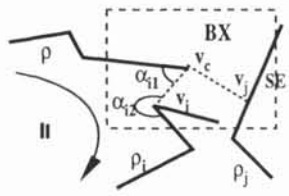


Fig.9 Next Valid Candidate

. Calculate angle $\alpha_i = 0.7 * \alpha_{i1} + 0.3 * \alpha_{i2}$.

Of all polylines inside or going through **BX**, if α_i of ρ_i is the smallest (or the largest if in a counter–clockwise search), ρ_i is the next valid candidate. If the counter–clockwise (or clockwise if in a counter–clockwise search) connector of ρ_i has been determined, the search stops.

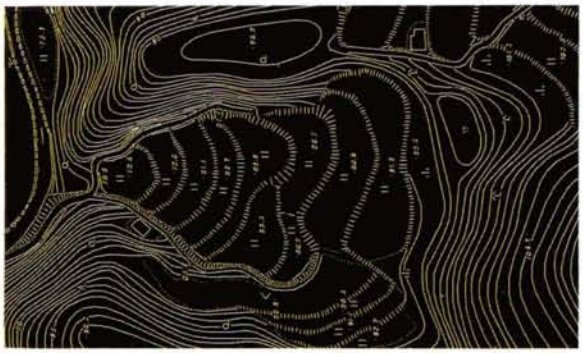
CONCLUSIONS

We proposed a ditching method for the extraction of open regions and inward tracing method for bounded regions. All of them are based on the vector data of a map. Many experiments reveal excellent efficiency, even though it is impossible for our algorithms to fit every case. Fig.1(b) is the final results of the extraction of various regions in fig.1(a). Fig.10 is an example of urban roads.

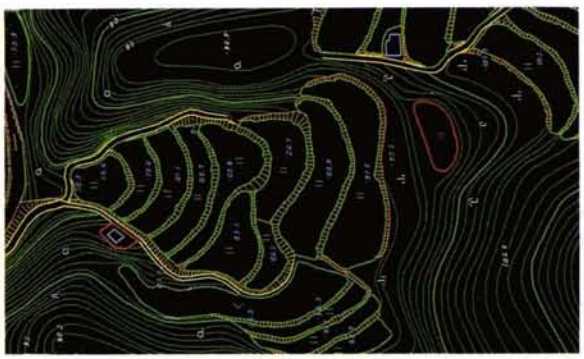
As a summary, the soundness for roads is up to 95 %. For other open regions such as slopes and various walls, the validity is only up to 75% and 65% resp.. This is because great interferences exist around the slopes and walls. For bounded regions, cases up to 85% are automatically solved. When some deviations exist in the results, they can be easily revised by using the interactive tool.

REFERENCES

1. Shin'ichi Satoh, Yutaka Ohsawa and Masao Sakauchi, A Proposal of Drawing Image Understanding System Applicable to Various Target Drawings, J. of IPS, Japan, 1992, Vol.33, No.9, pp1092–1102, (in Japanese).
2. Hiromitsu Yamada, Kazuhiko Yamamoto, and Katsumi Hosokawa, Directional Mathematical Morphology and Reformalized Hough Transformation for the Analysis of Topographic Maps, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.15, No.4, April, 1993, pp.380–387.
3. Hiromitsu Yamada, Kazuhiko Yamamoto, Taiichi Saito, and Shinji Matsui, MAP: Multi–Angled Parallelism for Feature Extraction from Topographical Maps, Pattern Recognition, Vol. 24, No. 6, 1991, pp. 479–488.
4. Yutaka Ohsawa, Yasuhiro Takishima, and Masao Sakauchi, An Efficient Map Data Conversion System Using Deviced Combination of Full Automatic and Human–assisted Recognition Phases, J. of IPS, Vol. J.72–D–II, No. 4, 1989, pp545–554, (in Japanese).
5. Yutaka Ohsawa, Shuzo Yamakawa, Yasumitsu Oda, and Tsuguo Shinhama, Recognition and Understanding Techniques for Graphical Database Capture, (Shokoda), 1989, (in Japanese).
6. Masao Sakauchi, On the Automated Data Conversion Techniques –– How can maps and engineer drawings be understood by computer, UDC 681.3:165:912.43, Vol.41, No.4, 1989, pp236–243, (in Japanese).

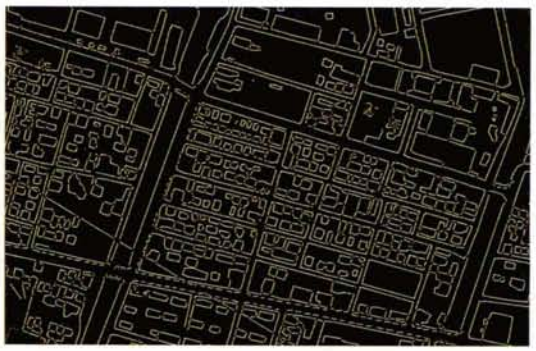


(a) before recognition

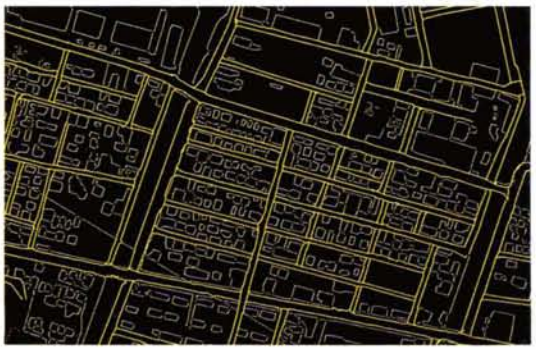


(b) final results with various regions

Fig.1 A Typical Map



(a) before recognition



(b) urban roads extracted

Fig. 10 An Example of Ditching Method