HARVESTING GEOGRAPHIC FEATURES FROM HETEROGENEOUS RASTER MAPS

by

Yao-Yi Chiang

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2010

Dedication

To my father, Chiú^{n} kok-liâng, and my mother, Hâu lē-hôa,

in-ūi lín-ê ài kā chi-chhî, chiah-ū chit-pún phok-sū lūn-bûn.

Acknowledgments

I would like to first and foremost thank my advisor, Professor Craig A. Knoblock. Craig has made my graduate study a wonderful journey. He gave me freedom and support in doing my research. He gave me opportunities to present in conferences and showed me how to interact with people in the research community. He spent numerous hours with me discussing research ideas and reading my papers. He has made me a better researcher (and a better person) than I had ever hoped. The first time after I gave Craig my draft of a conference paper, he said to me, and I quote: "Yao-Yi, where are your paragraphs?" And now, I can finish a whole Ph.D. thesis with meaningful paragraphs. I would like to also thank Craig on a personal level. He was (and still is) always willing to help on things not limited to the research. He taught me how to play ping-pong, showed me tricks on bidding on hotels and ski cabins, explained English pronunciations, and much more. Most importantly, he is fun to talk with and is very knowledgeable on all sorts of interesting things.

I would like to thank my other dissertation committee members, Professors Cyrus Shahabi and John P. Wilson. Cyrus has been a great source of ideas and inspiration on solving research problems and a great person for discussions. John gave me valuable advise on doing practical computer science research from the view of a geographer. Both

John and Cyrus helped to make the thesis possible from the very beginning. I would like to thank Professors C.C. Jay Kuo and Gérard G. Medioni for serving on my guidance committee and advising me on the research direction. I would like to thank my undergrad advisor Professor Frank Y.-S. Lin for his encouragement and advice that inspired my to pursue my own Ph.D. I would like to thank Phyllis O'Neil for her excellent comments and editing.

I would like to thank everyone in our group. Thank you, José Luis Ambite, for those lengthy (and fun) discussions of world history (in particular, the history of Spain, Taiwan, and China). Thank you, Mark Carman, for being a caring friend and helping me on the research. Thank you, Matt Michelson and Martin Michalowski, for your humor that made the office a fun working place. Thank you, Anon Plangprasopchok, for proofreading my papers and discussing homework assignments and tough math problems. Thank you, Alma Nava, for helping me on those administration issues at ISI.

I cannot live such a great life in Los Angeles without my friends; thank you, Chih-Kuei Sung, Yen-Ming Lee, Jimmy Pan, Sean Lo, John Wu, and Hui-Ling Hsieh. Thank you, Jason Chen and Shou-de Lin, for those discussions and advise in my earlier research life at USC. Thank you, Ling Kao, for always being around sharing those stressful moments. Thank you Dan Goldberg for showing me how to surf. Thank you, Chia-Hsiang Yang, for those late night 5k and 10k runs and being a great companion for overnight studies. Thank you, Chih-Wei Chang, you know how much I want to play StarCraft II with you, and it was truly a motivation for finishing my dissertation early. Thank you, Hung Yuan Su, you are like a brother to me.

I would like to thank my family for their love and support. My parents were always there for me when I had the most frustrating moments. Thank you my sister, Weili, for helping me on those tedious experiments.

Last, thank you, my dearest girl friend, Han-Chun, for your patience, caring, and love.

This research is based upon work supported in part by the University of Southern California under the Viterbi School Doctoral Fellowship, in part by the National Science Foundation under Award No. IIS-0324955, in part by the Air Force Office of Scientific Research under grant number FA9550-04-1-0105, and in part by the United States Air Force under contract number FA9550-08-C-0010.

The U.S. Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

Table of Contents

List Of Tables

List Of Figures

Abstract

Raster maps offer a great deal of geospatial information and are easily accessible compared to other geospatial data. However, harvesting geographic features locked in heterogeneous raster maps to obtain the geospatial information is challenging. This is because of the varying image quality of raster maps (e.g., scanned maps with poor image quality and computer-generated maps with good image quality), the overlapping geographic features in maps, and the typical lack of metadata (e.g., map geocoordinates, map source, and original vector data).

Previous work on map processing is typically limited to a specific type of map and often relies on intensive manual work. In contrast, this thesis investigates a general approach that does not rely on any prior knowledge and requires minimal user effort to process heterogeneous raster maps. This approach includes automatic and supervised techniques to process raster maps for separating individual layers of geographic features from the maps and recognizing geographic features in the separated layers (i.e., detecting road intersections, generating and vectorizing road geometry, and recognizing text labels).

The automatic technique eliminates user intervention by exploiting common map properties of how road lines and text labels are drawn in raster maps. For example, the road lines are elongated linear objects and the characters are small connected-objects.

The supervised technique utilizes labels of road and text areas to handle complex raster maps, or maps with poor image quality, and can process a variety of raster maps with minimal user input.

The results show that the general approach can handle raster maps with varying map complexity, color usage, and image quality. By matching extracted road intersections to another geospatial dataset, we can identify the geocoordinates of a raster map and further align the raster map, separated feature layers from the map, and recognized features from the layers with the geospatial dataset. The road vectorization and text recognition results outperform state-of-art commercial products, and with considerably less user input. The approach in this thesis allows us to make use of the geospatial information of heterogeneous maps locked in raster format.

Chapter 1

Introduction

In this chapter, I present the motivation for this thesis. I explain the challenges, outline my contributions, and briefly describe my approach and previous work that my approach is based on.

1.1 Motivation and Problem Statement

Humans have a long history of using maps. In particular, paper maps have been widely used for documenting geospatial information and conveying the information to viewers. Because of the widespread use of Geographic Information Systems (GIS) and the availability of low cost and high-resolution scanners, we can now obtain a huge numbers of scanned maps in raster format from various sources. For instance, the United States Geological Survey (USGS)[1] has been mapping the United States since 1879. The USGS topographic maps cover the entire country with informative geographic features, such as contour lines, buildings, and road lines. The georeferenced USGS topographic maps in raster format (i.e., the digital raster graphic, DRG) are now accessible from the USGS

[1] http://www.usgs.gov/

and the TerraServer-USA[2] websites. Other map sources, such as online map repositories like the University of Texas Map Library,[3] also provide information-rich maps and historical maps for many countries. Websites such as OpenStreetMap[4] and MultiMap[5] provide high quality digital maps generated directly from vector data (i.e., computer-generated maps) with valuable geospatial information, such as business locations. Moreover, we can harvest images from image search engines (e.g., Yahoo Image Search) and identify raster maps among the images [Desai et al., 2005; Michelson et al., 2008].

Raster maps not only are readily accessible compared to other geospatial data (e.g., vector data and imagery), but are also an important source of geospatial information. First, raster maps provide the most complete set of data for certain geographic features, such as the USGS topographic maps, which contain the contour lines of the entire United States. By fusing the publicly available USGS topographic maps with imagery, we can provide terrain information for any location in the U.S. inexpensively and fast. In addition, we can align parcel maps to obtain the georeferenced parcel data for building an accurate geocoder [Goldberg et al., 2009] for the areas where the parcel data are otherwise not easily accessible. Second, raster maps contain particular information that is difficult to find elsewhere. For example, the tourist map found using an image search engine on the Internet shown in Figure 1.1(a) contains location information such as gas stations,

[2]http://terraserver-usa.com/

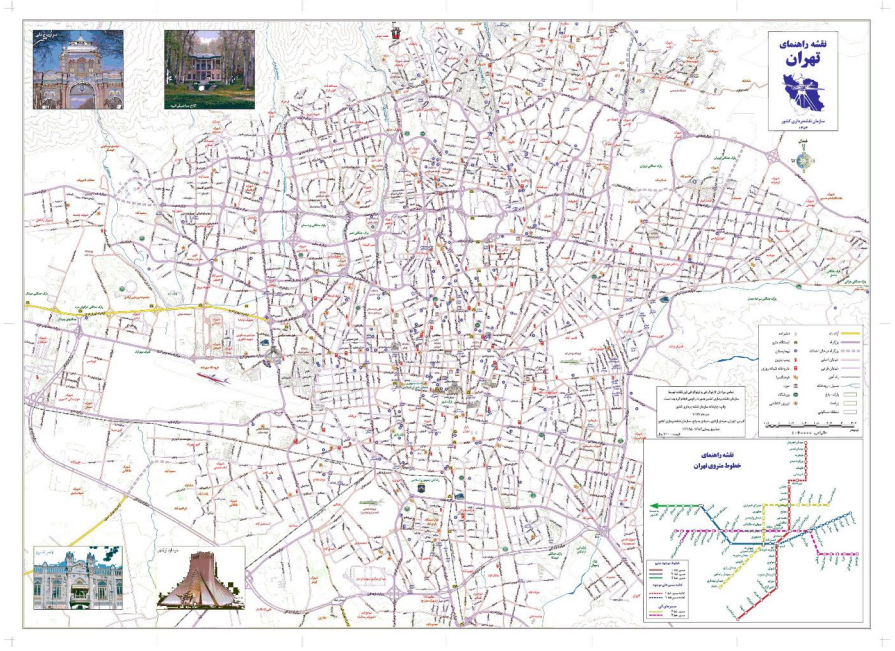
[3]http://www.lib.utexas.edu/maps/

[4]http://www.openstreetmap.org/

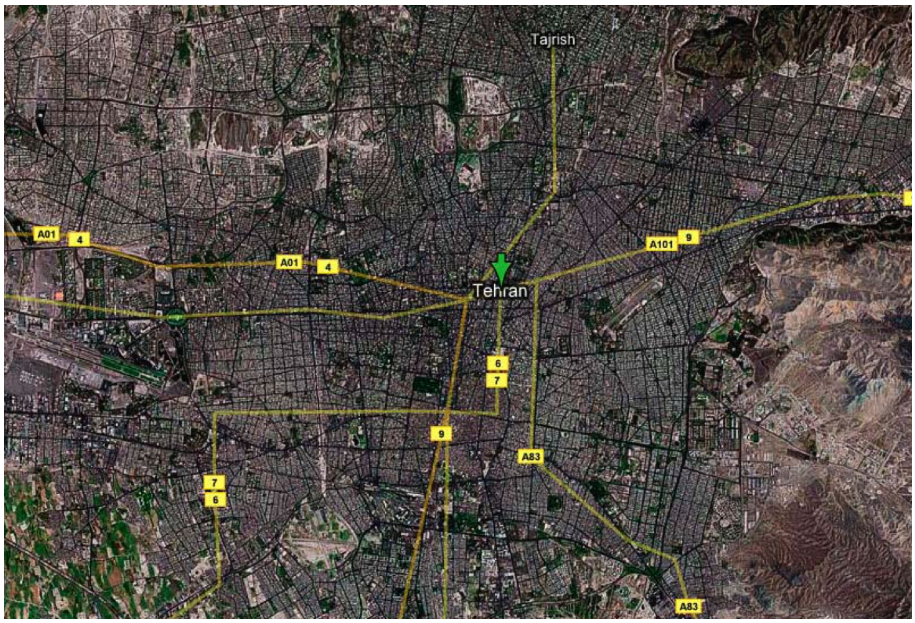
[5]http://www.multimap.com/

hotels, and road names of Tehran, Iran, while the hybrid view from Google Maps shown in Figure 1.1(b) shows only the major roads and their labels.

We can exploit the information in raster maps in several ways: first, we can fuse the raster map with other geospatial data as the example shown in Figure 1.2(a). By aligning the tourist map to the imagery, we can create an enhanced integrated representation of the two geospatial datasets and provide additional information, which cannot be viewed by the imagery alone. Second, we can extract image layers of individual geographic features (i.e., feature layers) from raster maps and fuse the extracted layers to other geospatial data. For example, we can extract the text layer from the raster map and align the text layer to imagery to annotate the imagery as shown in Figure 1.2(b). Third, we can recognize the features in the extracted feature layers to generate map context and improve understanding. For example, we can extract the road-intersection templates (i.e., the positions of road intersections, the number of roads intersecting at each intersection, and the orientations of the intersecting roads) and road vector data from the road layers of raster maps. The extracted road-intersection templates then can be used to extract roads from imagery [Koutaki and Uchimura, 2004]. Figure 1.2(c) shows that the aligned road-intersection templates can be used as seed templates to extract the roads from the imagery of Tehran, Iran, where we have limited access to the vector data. The extracted road topology and road vector data can be used as a matching feature to align the raster map, separated feature layers, and recognized features to other geospatial data that contain roads [Chen et al., 2008; Wu et al., 2007]. Similarly, we can recognize the text labels in the text layer to generate context for the imagery and produce map metadata for the indexing and retrieval of the raster map and imagery.



(a) A tourist map on the Internet

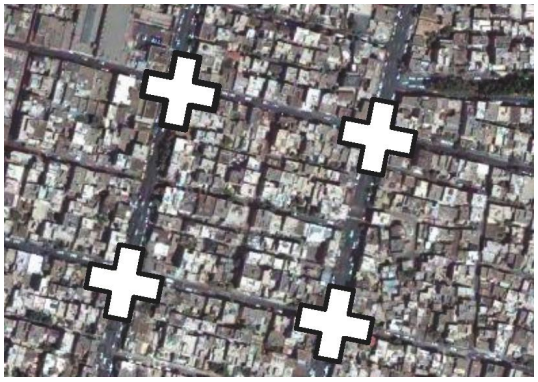


(b) The hybrid view of Tehran from Google Maps

Figure 1.1: The tourist map contains rich information that is difficult to find elsewhere for the city of Tehran, Iran

(a) Fusing a tourist map with imagery

(b) Labeling roads in imagery with a text layer separated from a map



(c) Extracting roads from imagery using road-intersection templates extracted from a map

Figure 1.2: Exploiting geospatial information in raster maps

Harvesting geographic features from raster maps is challenging for a number of reasons: first, maps are complex and contain overlapping layers of geographic features, such as roads, contour lines, labels, etc. Second, the image quality of raster maps is sometimes poor due to the scanning and/or image compression processes for preparing the maps in raster format. Third, the metadata of raster maps is often not available, such as the vector data used to produce the raster maps, map geocoordinates, legend information, etc. To overcome these difficulties, this thesis presents a general approach for harvesting geographic features from raster maps. This approach can process raster maps with varying map complexity and image quality, and does not rely on any prior knowledge of the input map.

1.2 Thesis Statement

We develop a general approach to exploit the information in heterogeneous raster maps. This approach decomposes the maps into raster layers of geographic features, recognizes features in the layers, and aligns the raster maps, extracted layers, and recognized features to other geospatial data.

1.3 Approach

The overall approach to harvesting geographic features from heterogeneous raster maps is shown in Figure 1.3. Figure 1.4 shows examples of input raster maps. The input is either a computer-generated map, such as the TIGER/Line map shown in Figure 1.4(a), or a scanned map, such as the USGS topographic map and Thomas-Brothers map shown

in Figure 1.4(b) and Figure 1.4(c). Since the geographic features in raster maps generally overlap, the first step is to separate the raster map into individual layers of geographic features, which is called the map decomposition step. To process raster maps with varying map complexity (i.e., overlapping features) and image quality, the map decomposition step includes a fully automatic approach and a supervised approach with user training. For raster maps with good image quality (e.g., computer-generated raster maps), I present a fully automatic technique that exploits the distinctive geometry of the desired geographic features (e.g., road lines) to decompose raster maps into feature layers, namely the road layer and the text layer [Chiang et al., 2005, 2008]. For complex raster maps, such as the maps that contain non-road linear features, or maps with poor image quality, I present supervised techniques including user labeling that require minimal user input to separate the road and text layers from raster maps [Chiang and Knoblock, 2009c]. The supervised techniques can be used on scanned and compressed maps that are otherwise difficult to process automatically and tedious to process manually.

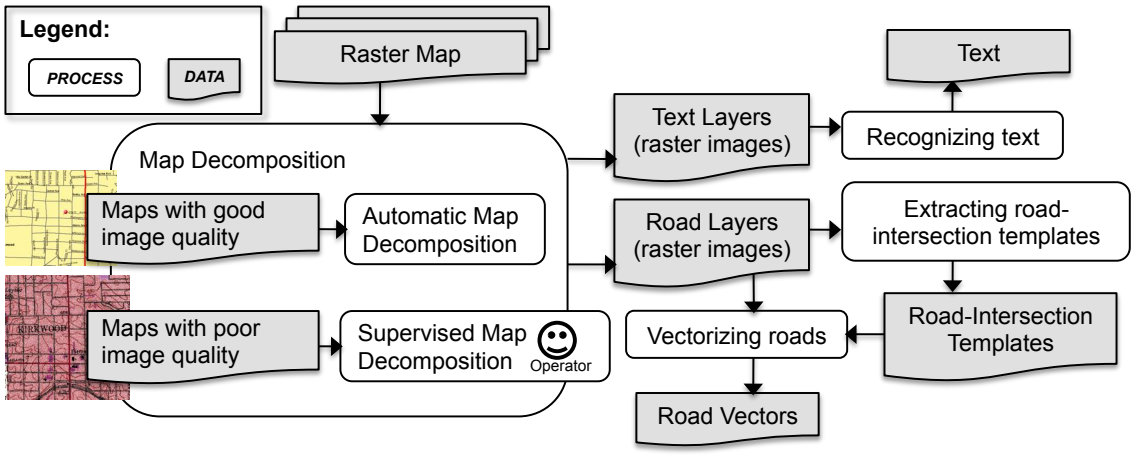


Figure 1.3: The overall approach to harvesting geographic features from heterogeneous raster maps

(a) An example of TIGER/Line Maps (b) An example of USGS topographic maps (c) An example of scanned Thomas-Brothers Maps

Figure 1.4: Example raster maps

With the separated feature layers from the map decomposition step, the next step is to recognize geographic features in the feature layers. For the road layer, I present an automatic technique to extract the road-intersection templates from the raster layer [Chiang and Knoblock, 2008; Chiang et al., 2005, 2008]. Since the road is a common geographic feature across many geospatial data sources, the set of road-intersection templates of a raster map can serve as a reference feature in a map conflation system [Chen et al., 2006a, 2008] to compute a transformation matrix for georeferencing the map and aligning the map with other geospatial data, such as imagery. Further, I present an automatic technique to extract the road vector data from the road layer based on the accurately extracted road-intersection templates [Chiang and Knoblock, 2009a,b]. Therefore, we can produce road vector data from raster maps for areas where we have a limited access to such data.

For the text layer, I present an automatic technique that first identifies individual text labels that are multi-oriented and in various font types and sizes in the text layer. Then, the automatic technique rotates each identified text labels to the horizontal direction and

employs an optical character recognition (OCR) component to translate the labels into machine-editable text.

1.4 Previous Work on Map Imagery Conflation

Chen et al. [2004b] and Wu et al. [2007] present techniques to align road vectors to imagery. Chen et al. [2004b] utilize road-intersection templates extracted from road vector data to search a local area for identifying corresponding road intersections in imagery and then align the road vector data to the imagery using the identified intersection correspondences. Wu et al. [2007] use the road lines from road vector data to match the road areas in the imagery, which allows their approach to handle suburban areas where only a few road intersections exist.

In our previous work [Chen et al., 2004a], we present an approach to automatically align raster maps with orthoimagery. We first exploit our previous work [Chen et al., 2004b] to identify the locations of road intersections in imagery, and we detect the road-intersection templates in raster maps by using the technique described in this thesis (Chapter 2). Finally, we compute the alignment between the two point sets and use the conflation technique [Saalfeld, 1993] to align the map with the imagery.

This thesis presents a general technique that takes a raster map as input, decomposes the map into layers of geographic features, and recognizes the geographic features (i.e., the road-intersection templates, road vector data, and text labels) in the separated layers. By exploiting the previous work [Chen et al., 2004a,b; Wu et al., 2007] with the extracted road-intersection templates or road vector data, we can georeference the raster map, the

extracted feature layers, and the recognized features and align them to other geospatial data.

1.5 Contributions of the Research

The key contribution of this thesis is a *method for separating and recognizing geographic features from raster maps and aligning the raster maps, separated layers, and recognized features to other geospatial data.* This thesis offers four major contributions:

- An automatic approach that separates road and text layers from raster maps (Chapter 2)
- An automatic approach that extracts road-intersection templates from road layers for the alignment of raster maps and other geospatial data (Chapter 2)
- A general approach for extracting and vectorizing road geometry from raster maps (Chapter 3), which includes:
  - A supervised approach that handles complex raster maps or maps with poor image quality for separating road layers from the maps
  - An automatic approach for extracting road vector data from road layers
- A general approach for recognizing text labels in raster maps (Chapter 4), which includes:
  - A supervised approach that handles complex raster maps or maps with poor image quality for separating text layers from the maps
  - An automatic approach for recognizing text labels in text layers

1.6 Outline of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 describes the automatic approach to separate road and text layers from raster maps and to detect the road-intersection templates for map alignment. Chapter 3 presents the road vectorization technique, which includes a supervised approach to separate road layers from raster maps and an automatic approach to extract the road vector data from the separated road layers. Chapter 4 explains the text recognition technique, which includes a supervised approach to separate text layers form raster maps and an automatic approach to recognize the text labels in the separated text layers.

Chapters 3 and 4 first present the supervised techniques for handling complex raster maps or maps with poor image quality for separating the feature layers. Together with the automatic map decomposition technique described in Chapter 2, this approach processes heterogeneous raster maps to separate their road and text layers with minimal user input. The two chapters then describe general techniques for automatically converting the geographic features in the separated road and text layers into a machine-editable format.

Chapter 5 reviews related work. It first presents the map processing research on separating or recognizing general geographic features, and then focuses on work closely related to this thesis. Chapter 6 concludes with the contributions of this thesis and discusses future research.

Chapter 2

Automatic Decomposition and Alignment of Raster Maps

This chapter describes my first and second contributions, which include my automatic map decomposition technique called the Automatic Map Decomposer (AutoMapDecomposer) for separating the road and text layers from raster maps and my automatic road-intersection extraction technique called the Automatic Intersection Detector (AutoIntDetector) for extracting the road intersections from the separated road layer [Chiang et al., 2005, 2008]. Chen et al. [2008] utilize the techniques in this chapter to detect road-intersection templates (i.e., the positions of road intersections, the number of roads intersecting at each intersection, and the orientations of the intersecting roads) from a set of raster maps from various sources. They match the extracted road-intersection templates to georeferenced orthoimagery to identify the geospatial extent of the map and align the map to the orthoimagery. By using their technique, we can further align the separated feature layers and recognized features with other geospatial data to generate a hybrid view and create context for the integrated data, such as annotating roads by aligning an extracted text layer from a street map to imagery.

Figure 2.1 shows the overall approach of the AutoMapDecomposer and AutoIntDetector. To separate the overlapping feature layers from raster maps, the AutoMapDecomposer exploits the distinctive geometric properties of the road lines and text characters to automatically separate the road and text pixels from raster maps. The AutoIntDetector then links the extracted road pixels and traces the centerlines of the linked road objects to generate the road geometry for detecting the road intersections. The output of the AutoIntDetector is a set of road-intersection positions, the number of roads that meet at each intersection, and the orientation of each intersecting road. Both the AutoMapDecomposer and AutoIntDetector do not assume any prior knowledge of the input maps, such as the color of the roads, vector data, or legend information.

2.1 Automatic Decomposition of Raster Maps

The AutoMapDecomposer includes two distinctive steps: the first step is to extract the binary map, which contains the foreground pixels of the raster map. With the binary map, the second step separates the road pixels from the text pixels by removing the foreground pixels that do not hold the properties that constitute road lines.

2.1.1 Automatically Extracting Foreground Pixels from Raster Maps

Since the foreground pixels of the raster maps contain the feature layers (i.e., road and text layers), the first step for extracting the layers is to extract the foreground pixels, which is a problem that has no universal solution on all types of images [Henderson et al., 2009; Lacroix, 2009; Leyk and Boesch, 2010; Sezgin, 2004]. The AutoMapDecomposer utilizes a technique that analyzes the grayscale histogram to classify the histogram values

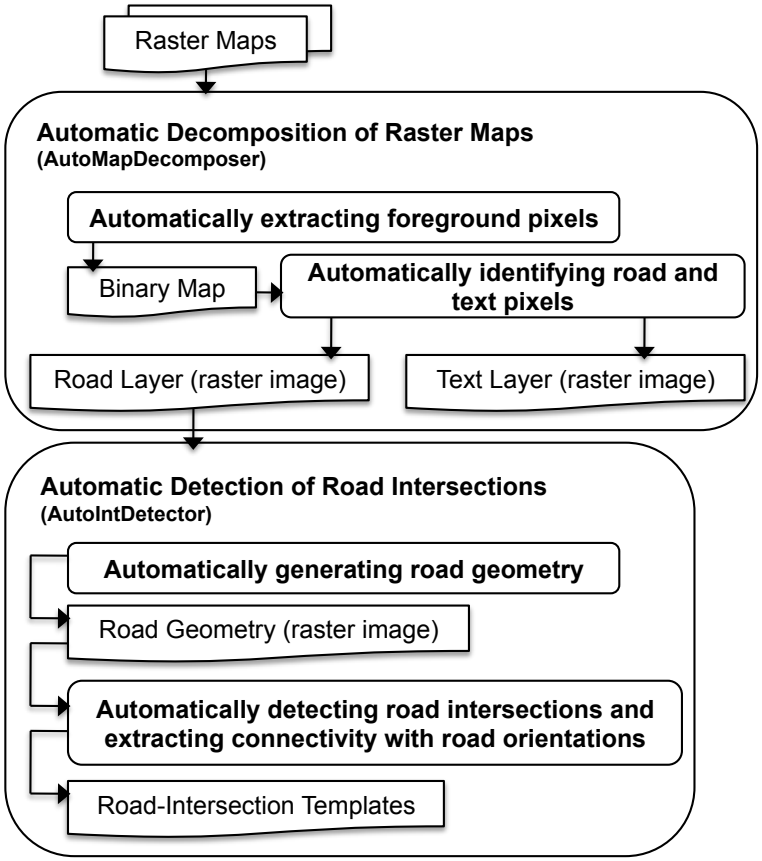


Figure 2.1: The overall approach of the AutoMapDecomposer and AutoIntDetector

(i.e., the luminosity levels) into the background and foreground clusters for extracting the foreground pixels. An edge detector [Ziou and Tabbone, 1998], such as the Canny edge detector [Canny, 1986], is sometimes used to extract the foreground features instead of histogram analysis; however, the edge detector is sensitive to noise, which makes it difficult to use for common raster maps without careful manual tuning [Habib et al., 1999]. My grayscale-histogram analysis (GHA) algorithm is based on the heuristics of the luminosity usage of raster maps and generates a better result of the feature layers for further feature recognition tasks. The heuristics of the luminosity usage of raster maps are:

- The background luminosity levels of a raster map have a dominant number of pixels; and the number of foreground pixels is significantly smaller than the number of background pixels.
- The foreground luminosity levels have high contrast against the background luminosity levels.

To generate the grayscale histogram of the input raster map, the AutoMapDecomposer first converts the original input raster map to an 8-bit grayscale (i.e., 256 luminosity levels) image by assigning the average value of the red, green, and blue strength of each map pixel as the pixel's luminosity level. For the color map shown in Figure 2.2(a), Figure 2.5(a) shows the converted grayscale map.

With the grayscale histogram, the GHA first partitions the histogram into luminosity clusters. Then, the GHA classifies each of the clusters as either a background or foreground cluster. Figure 2.3 shows the pseudo-code of the GHA.

(a) An example of USGS topographic maps (b) The example map in grayscale

Figure 2.2: An example USGS topographic map after converting to grayscale

Figure 2.4 shows the grayscale histogram of Figure 2.5(a). In the grayscale histogram, the X-axis represents the luminosity levels, from black (0) to white (255), and the Y-axis represents the number of pixels for each luminosity level. For a peak in the histogram, the GHA exploits the triangle method by Zack et al. [1977] to find the boundaries of a luminosity cluster that contains the peak. To find the left/right boundary of a luminosity cluster, the GHA first constructs a line called the triangle line from the peak to the origin/end point in the histogram. Then, the GHA computes the distance between each Y-value and the triangle line. The GHA identifies the luminosity level that has its Y-value under the triangle line and has the maximum distance to the triangle line as the cluster's boundary. The dashed line in Figure 2.4 indicates the identified left boundary of *Cluster 1*. If no luminosity level exists with its Y-value under the triangle line, such as when constructing the triangle line between *Peak 3* and the origin point to find the

```
// The list for storing the clusters of luminosity levels
 ClusterList;

 Function void GrayscaleHistogramAnalysis (Histogram histogram)
    int P = FindPeak(histogram);
    int left_boundary;
    int right_boundary;
    If (P is closer to WHITE than to BLACK) {
        right_boundary = WHITE;
        left_coundary = IdentifyLeftBoundaryUsingZack(P);
    } else {
        left_boundary = BLACK;
        right_boundary = IdentifyRightBoundaryUsingZack(P);
    }

    Cluster FirstBackgroundCluster = new Cluster(P, left_coundary, right_boundary);
    FirstBackgroundCluster.isBackgroundCluster = true;
    ClusterList.Add(FirstBackgroundCluster)

    histogram.RemoveHistogramValuesOFIdentifiedCluster(FirstBackgroundCluster);

    P = FindPeak(histogram);
     while (P exists) {
         int left_boundary= IdentifyLeftBoundaryUsingZack(P);
         int right_boundary= IdentifyRightBoundaryUsingZack(P);
         Cluster C = new Cluster(P, left_boundary, right_boundary);
         ClusterList.Add(C);
         histogram.RemoveHistogramValuesOFIdentifiedCluster(C)
         P = FindPeak(histogram);
    }

    Cluster FirstForegroundCluster =
    ClusterList.FindTheFarestClusterFrom(FirstBackgroundCluster);
    FirstForegroundCluster.isBackgroundCluster = false;

    For each Cluster C in ClusterList {
        if (C.pixel_count is closer to FirstBackgroundCluster.pixel_count
            than to FirstForegroundCluster.pixel_count)
             C.isBackgroundCluster = true;
        else
             C.isBackgroundCluster = false;
    }
```

Figure 2.3: The pseudo-code of the grayscale-histogram analysis (GHA) algorithm

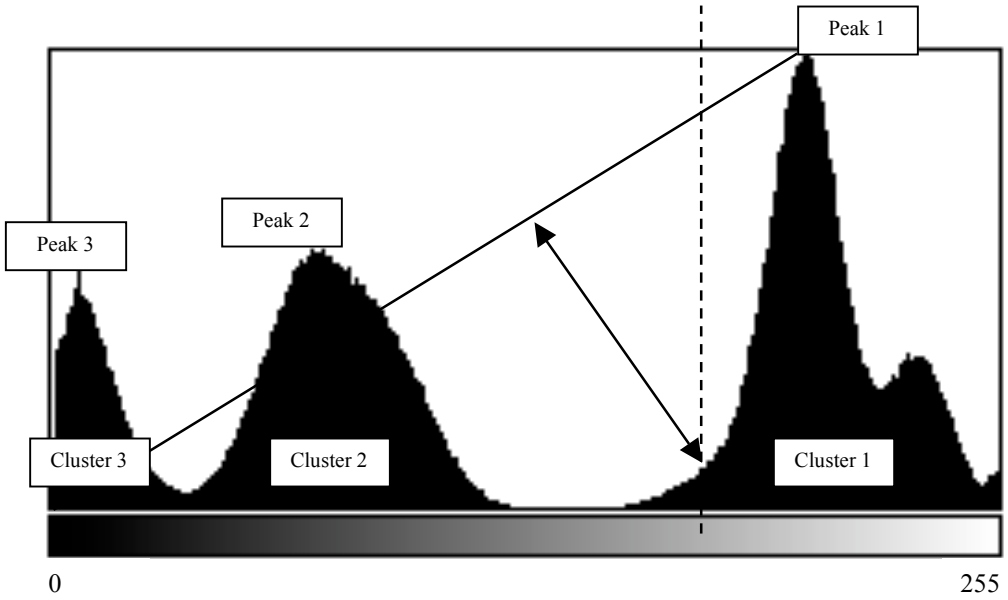


Figure 2.4: Identifying *Cluster 1*'s left boundary using the Triangle method

left boundary of *Cluster 3*, then the GHA selects the origin or end points as the cluster boundary. For example, the left boundary of *Cluster 3* is the origin point.

Since the background luminosity levels have a dominant number of pixels, the GHA starts searching for the global maximum of the histogram values to identify the first background cluster. As shown in Figure 2.4, the GHA first finds *Peak 1* and then checks whether *Peak 1* is closer to white (255) or black (0) in the histogram to determine which portion of the histogram (i.e., the luminosity levels to the left or right of *Peak 1*) contains the foreground luminosity levels. If the global maximum is closer to white, such as *Peak 1*, the GHA uses white as the right boundary of the first background cluster. This is because the foreground generally has high contrast against the background. For example, if the background contains a light gray of luminosity level 200, we often find the foreground luminosity levels spreading in the histogram from 0 to 200 instead of 200 to 255. As a result, for *Cluster 1* in Figure 2.4, the GHA identifies the end point (white) as the

cluster's right boundary and then locates the left boundary using the triangle method. If *Peak 3* is the global maximum, the GHA selects the origin point (black) as the cluster's left boundary and then uses the triangle method to locate the right boundary.

After the GHA identifies the first background cluster, the algorithm searches for the next peak and uses the triangle method to locate the cluster boundaries until every luminosity level in the histogram belongs to a cluster. In the example, the GHA first identifies *Cluster 1* as the first background cluster. Then, since the foreground colors usually have a high contrast against the background colors, the cluster that is the farthest from the first background cluster in the histogram is the first foreground cluster (i.e., *Cluster 3* in the example).

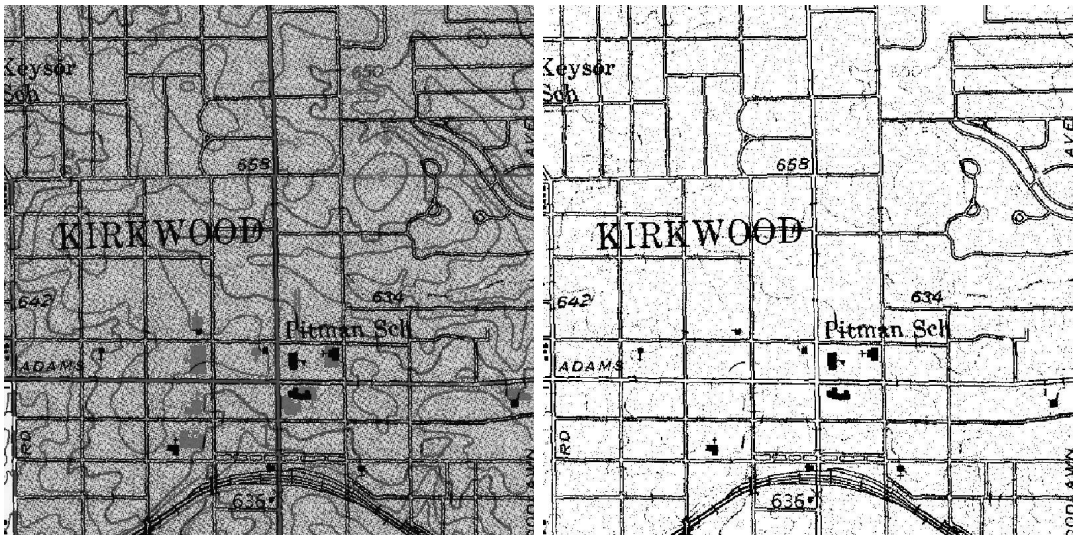
After the GHA identifies the first foreground cluster, for the remaining clusters, the algorithm classifies them as either the foreground or background clusters based on the number of pixels in each cluster. This is because if a cluster uses a number of pixels similar to the first background cluster, the cluster could contain some of the luminosity levels of the map background or the luminosity levels used to paint homogeneous-region features, such as parks, lakes, etc. Therefore, if the number of pixels of a cluster is closer to the number of pixels of the first background cluster than the first foreground cluster, the GHA classifies it as a background cluster; otherwise the cluster is a foreground cluster. Formally, the the GHA calculates the pixel ratio for the classification, which is defined as:

$$PixelRatio = \frac{ClusterPixels}{TotalNumbersOfPixels} \tag{2.1}$$

	Cluster 1	Cluster 2	Cluster 3
Cluster Pixels	316,846	237,876	85,278
Pixel Ratio	50%	37%	13%

Table 2.1: Number of pixels in each cluster

Table 2.1 shows that the number of pixels in *Cluster 2* is closer to *Cluster 1* (background) than *Cluster 3* (foreground), so *Cluster 2* is a background cluster. After the GHA classifies each of the clusters, the algorithm removes the background clusters and the result is a binary map containing foreground pixels only, such as the example shown in Figure 2.5(b).



(a) The example map in grayscale (b) The binary map

Figure 2.5: An example USGS topographic map after extracting the foreground pixels

2.1.2 Automatically Identifying Road and Text Pixels in Foreground Pixels

After the AutoMapDecomposer extracts the foreground pixels, the result is a binary map that is a union of overlapping feature layers. To separate the text layer from the

road layer, the AutoMapDecomposer exploits the geometric properties of road lines and characters in a raster map with a text/graphic separation algorithm to separate text and line pixels. Text/graphics separation algorithms [Bixler, 2000; Cao and Tan, 2002; Fletcher and Kasturi, 1988; Li et al., 1999, 2000; Myers et al., 1996; Nagy et al., 1997; Tang et al., 1996; Velázquez and Levachkine, 2004] are designed for grouping small connected components (i.e., small CCs) and separating them from linear structures (i.e., elongated lines). Then, the AutoMapDecomposer automatically detects the road format and the road width using the set of identified line pixels to remove linear structures that are not road lines.

2.1.2.1 Text/Graphics Separation

The AutoMapDecomposer utilizes the text/graphics separation algorithm from Cao and Tan [2002] to first separate the text layer and linear structures from the raster map based on the following geometric properties of road lines and text labels in a raster map:

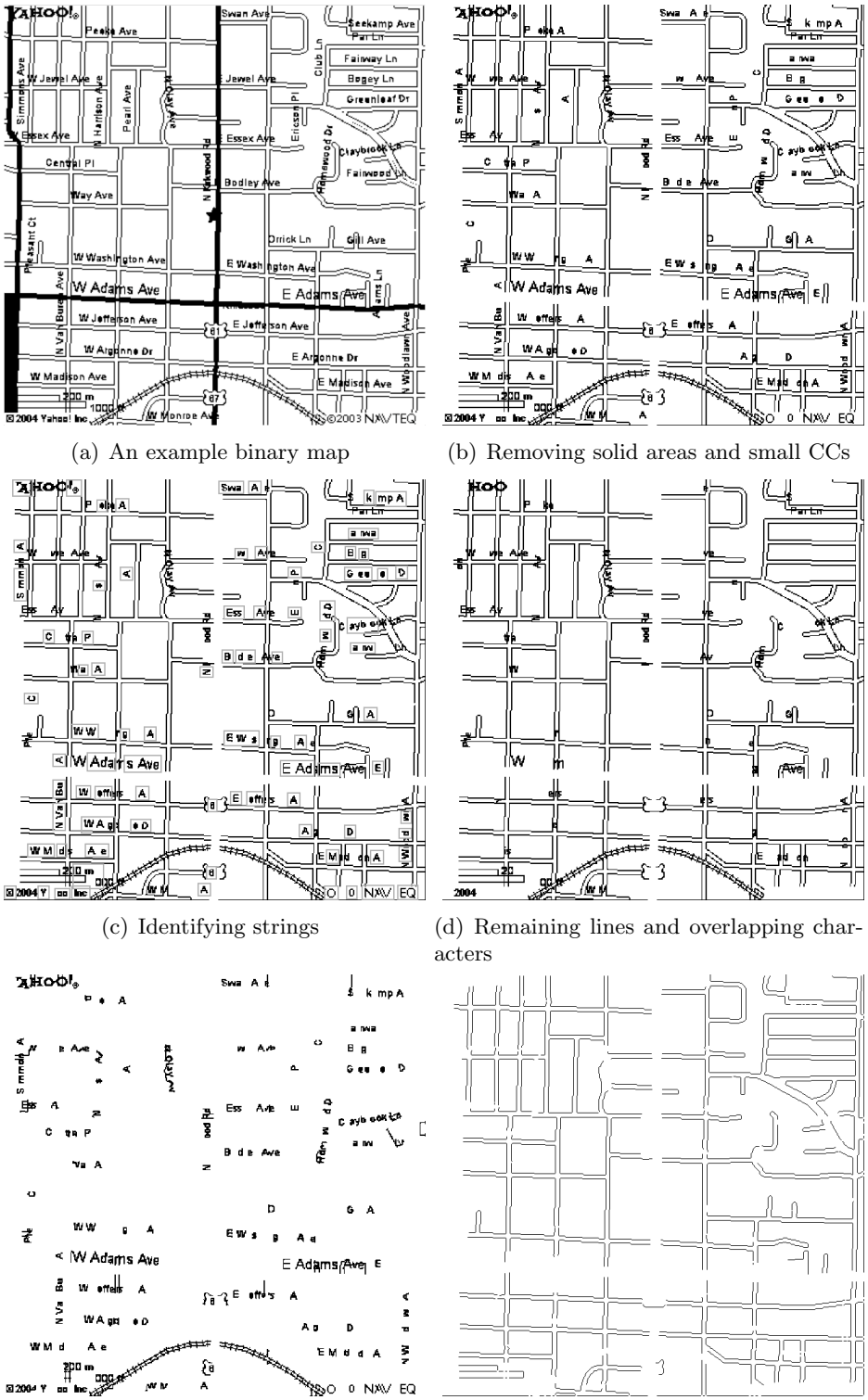
- Road lines are connected to each other (i.e., road network) so that a road layer usually has a small number of connected components or even only one large connected component when the entire road layer is connected.
- Characters are isolated small connected components that are close to each other.
- Character strokes are generally shorter than lines that represent roads.

The text/graphics separation algorithm first removes solid areas using morphological operators and then removes small connected components in the binary map using a size threshold. Figure 2.6(b) shows the remaining foreground pixels after the solid areas and

small connected components are both removed. Then, from each removed small connected component, the text/graphics separation algorithm searches and groups neighboring connected components to identify string objects. The gray boxes in Figure 2.6(c) show the identified string objects, and Figure 2.6(d) shows the remaining foreground pixels after the string objects are removed. Finally, the remaining foreground pixels are divided into line segments and the length of each line segment is checked to determine if the line segment belongs to a linear structure (e.g., road lines) or a text object. Since the line segments that constitute characters (i.e., strokes) are usually smaller than the ones that constitute lines, a length filter can be used to separate the characters that overlap with linear structures.

The AutoMapDecomposer adds up the identified strings and the line segments of the characters that overlap with linear structures to produce the text layer as shown in Figure 2.6(e). There are objects that are not text in the text layer, and these non-text objects will be filtered out using an optical character recognition component during the text recognition.

Figure 2.6(f) shows the road layer after the text layer is removed from the binary map. The road layer might contain linear features that are not roads, such as contour lines. The AutoMapDecomposer removes the linear features that are not roads in the next subsection. Moreover, the extracted road layer contains broken lines since the characters that overlap with the road lines are removed. The broken lines are reconnected to generate the road geometry during the detection of road intersections in Section 2.2.1 of this chapter.



(a) An example binary map

(b) Removing solid areas and small CCs

(c) Identifying strings

(d) Remaining lines and overlapping characters

(e) The extracted text layer

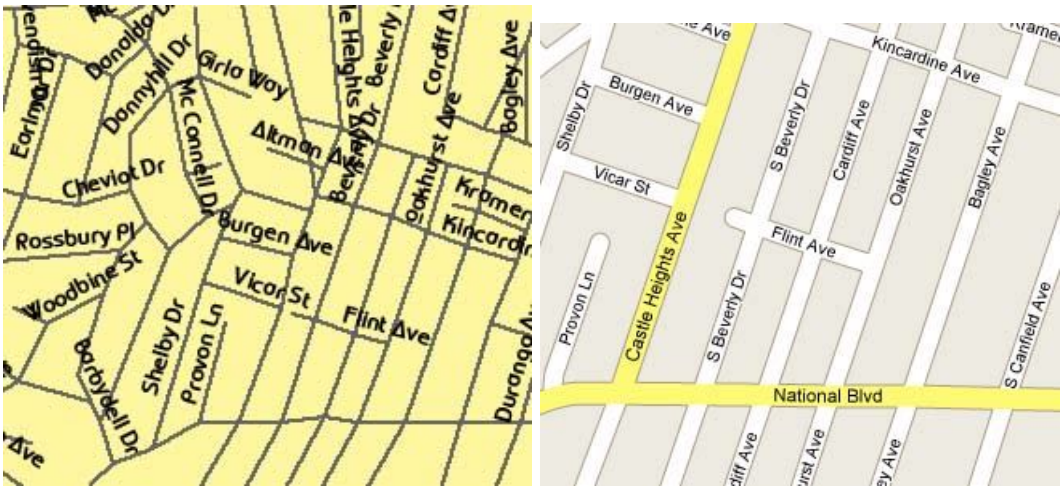
(f) The extracted road layer

Figure 2.6: Text/graphics separation using an example double-line map from Yahoo Maps

2.1.2.2 Parallel-Pattern Tracing (PPT)

To further remove the linear structures that are not roads in the road layer, the AutoMapDecomposer utilizes the Parallel-Pattern Tracing (PPT) algorithm that traces the parallel patterns of lines to detect the road format and road width based on the following geometric properties of road lines in a raster map:

- The majority of the roads share the same road format: single-line or double-line roads, as the examples shown in Figure 2.7, and the majority of the roads have the same road width within a map.
- In a double-line map, the linear structures in single-line format are not roads but can be other geographic feathers, such as contour lines.



(a) A single-line map from TIGER/Line (b) A double-line map from Google Maps

Figure 2.7: Example single-line and double-line maps

In this subsection, I first explain the parallel patterns that the PPT identifies and then describe how the AutoMapDecomposer exploits the PPT to detect the road format and road width.

For a pixel C in the grid domain (i.e., the raster format), there are four possible one-pixel width straight lines constituted by C and C's eight connecting pixels, which represent the lines of 0 degrees, 45 degrees, 90 degrees, and 135 degrees respectively, as shown in Figure 2.8. These four lines are the basic elements that constitute straight lines with various slopes passing through the pixel C. In Figure 2.9, for example, to represent a 30-degree line, one of the basic elements, the 45-degree line-segment, is drawn in black, then additional gray pixels are added to tilt the line.

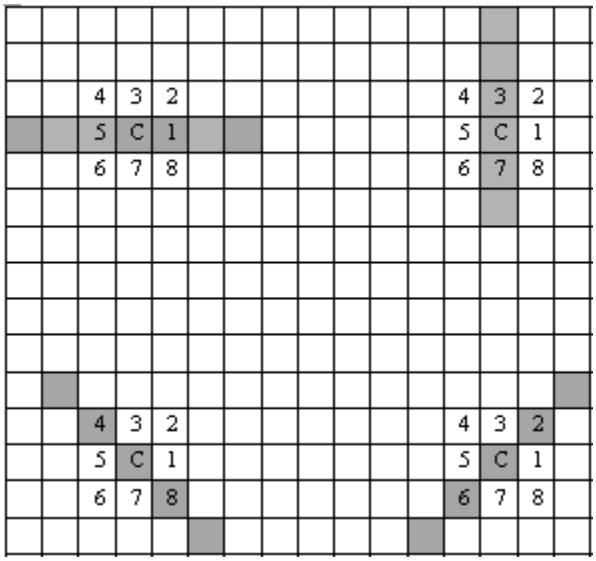


Figure 2.8: Basic single-line elements

Based on the four basic single-line elements, if a pixel C is on a double-line road, there are eight possible parallel patterns of double-line roads as shown in Figure 2.10 (the dashed cells are the possible corresponding parallel lines). Hence, if the PPT detects any of the eight parallel patterns for a given foreground pixel, the PPT classifies the pixel as a double-line road pixel.

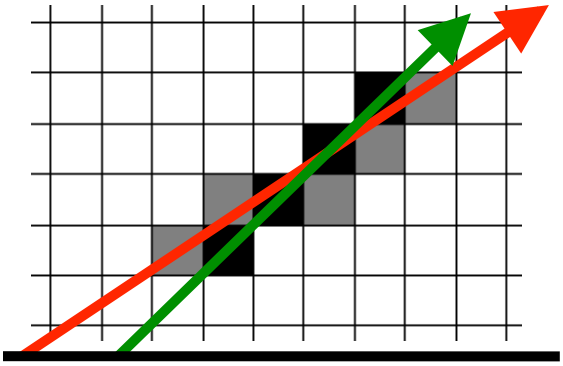


Figure 2.9: Using the 45^o basic element (black cells) to constitute a 30^o line segment (both black and gray cells)

Figure 2.11 shows the pseudo-code of the PPT. To detect the parallel pattern, for a given road width (RW), the PPT searches for the corresponding foreground pixels at a distance of RW in the vertical and horizontal directions. For example, Figure 2.10 shows the detection of the parallel patterns using the crosses at the pixel C with the length of RW. The crosses locate at least two road-line pixels for each of the eight patterns; one is in the horizontal direction and the other one is in the vertical direction.

Figure 2.12 shows two examples of double-line roads. If a foreground pixel is on a horizontal or vertical road line, such as the pixel C in Figure 2.12(a), the PPT can find two foreground pixels along the road line within a distance of RW and at least another foreground pixel on the corresponding parallel road line in a distance of RW. If the orientation of the road line is neither horizontal nor vertical as shown in Figure 2.12(b), the PPT can find one foreground pixel in each of the horizontal and vertical directions on the corresponding road lines at a distance of RW as shown in Figure 2.12(b).

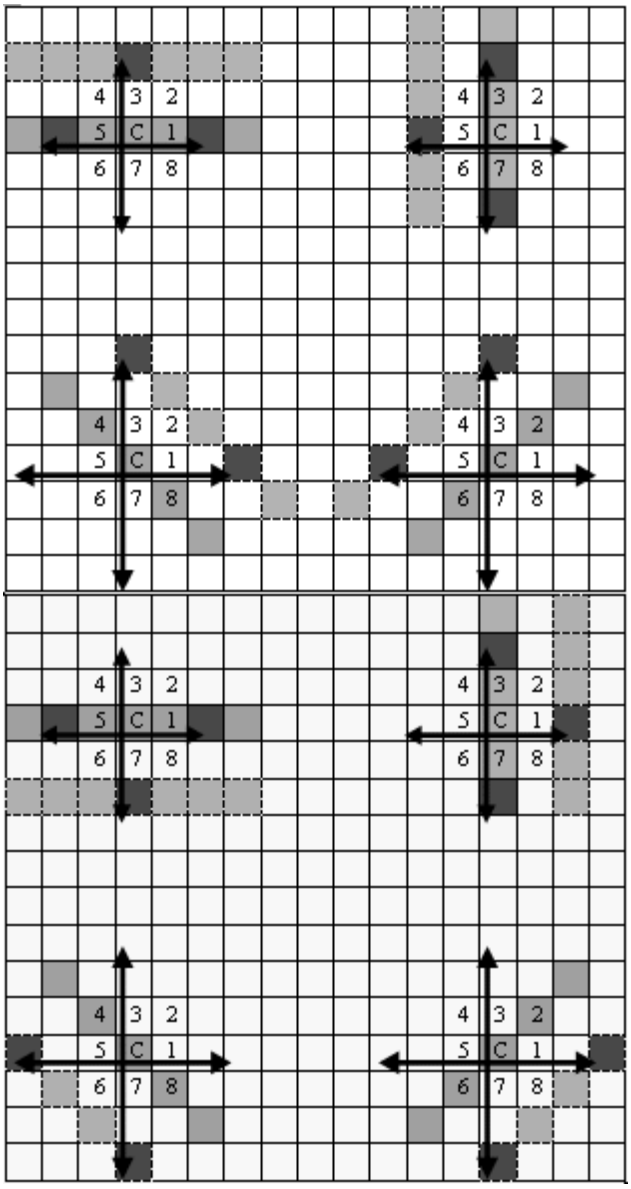
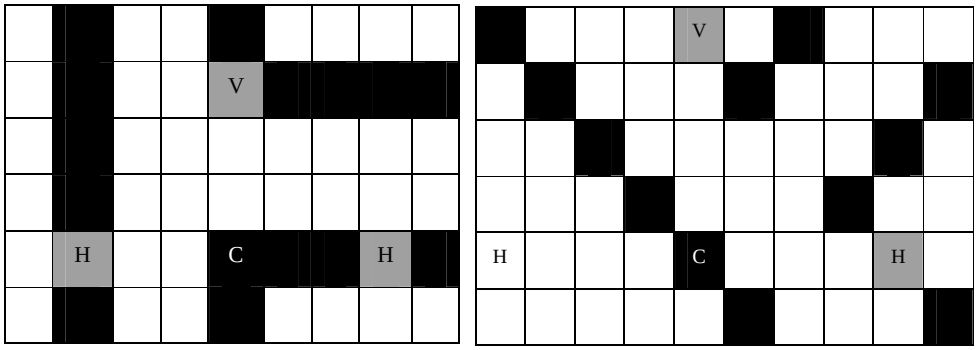


Figure 2.10: Basic double-line elements

```
image; // The input image
width, height; // The image width and height
K; //  Test from one-pixel to K-pixel wide

/*  The number of pixels
that have the parallel pattern at each
road width */
ppt_pixel_cnt;

void main()  // Program starts here
    ppt_pixel_cnt= new int[K];
    for (int i = 0; i < K; i++)
        ppt_pixel_cnt[i] = PPT(i, false);
    // Determing road format
    int RW = FindPeak(ppt_pixel_cnt);
    /* Removing single-line pixels for
    double-line road layer */
    if(RW exists)
        PPT(RW, true);
```

```
Function int PPT(int i, bool remove_singleline_pixels)
for(int y = 0; y < height; y++) {
    for(int x = 0; x < width; x++) {
        if(IsForeground(x,y)) {
            if ((InsideImage (x + k, y) &&
                IsForeground (x + k, y) ||
                InsideImage (x - k, y) &&
                IsForeground (x + k, y)) &&
                (InsideImage (x, y + k) &&
                IsForeground (x, y + k) ||
                InsideImage (x, y - k) &&
                IsForeground (x, y - k))
                 ppt_pixel_cnt[i]++;
             else if (remove_singleline_pixels)
                 image[x, y] = BACKGROUND;
    } // end if
   } // end for
} // end for
```

Figure 2.11: The pseudo-code of the Parallel-Pattern Tracing algorithm



(a) Road width is three pixels (b) Road width is four pixels

Figure 2.12: Examples of double-line roads and the parallel patterns

By detecting the parallel pattern, the PPT determines if a foreground pixel is a double-line road pixel or a single-line road pixel at a given *RW*. To detect the road layer format, the AutoMapDecomposer applies the PPT on the binary map varying the road width from one to K pixels as the main function shown in the pseudo-code in Figure 2.11. K is the maximum width in pixels of a double-line road in the map. For example, to detect a double-line road of 20 meters wide in a 2 meters-per-pixel map, K needs to be at least 10 pixels. A bigger K ensures that the PPT can find wider roads, but it requires more processing time.

After the AutoMapDecomposer applies the PPT, the algorithm computes the ratio of the number of double-line road pixels to the number of total foreground pixels in the binary map (double-line ratio) at each *RW*. Figure 2.13 shows the double-line ratios of various double-line and single-line maps. When the *RW* is one pixel, the double-line ratios are close to one (100%). After *RW* increases, the double-line ratios start to decline. This is because the foreground pixels tend to be near each other, and it is easier to find corresponding pixels even if the PPT does not have the correct road width or the map is not a double-line map.

If the map is a double-line map and the PPT applies on the map using the correct road width, a peak appears in the chart as shown in Figure 2.13(a) since most of the road pixels have one of the parallel pattens detected by the PPT. For example, the double-line high-resolution map from ESRI Maps has a peak at two pixels. The double-line high-resolution maps from MapQuest Maps and Yahoo Maps and the double-line map from USGS topographic maps have peaks at four pixels in the chart. The maps from TIGER/Line and the maps from ESRI Maps and MapQuest Maps that are not high

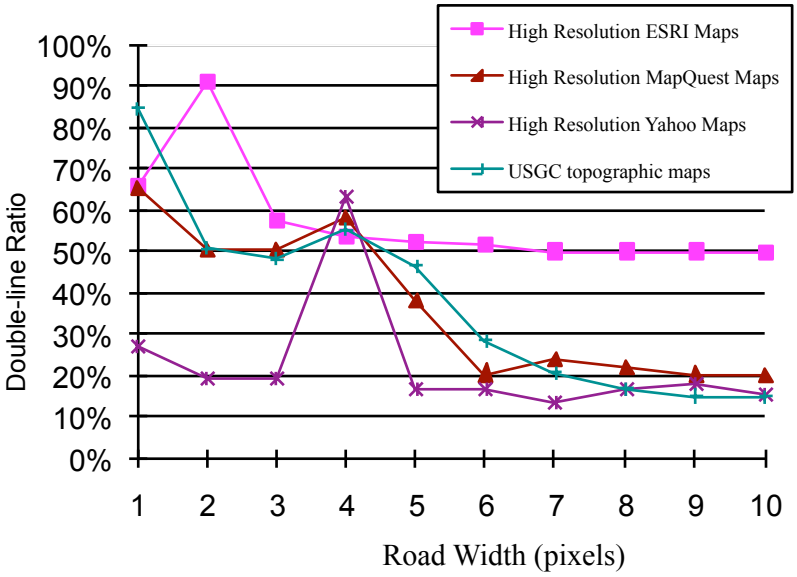
resolution are all single-line maps, which do not have any peaks in the chart as shown in Figure 2.13(b).

Using this method, the AutoMapDecomposer determines the road format automatically and also obtains the road width by searching for a peak. For example, from Figure 2.13(a), the AutoMapDecomposer determines that the USGS topographic map is a double-line map with road width equal to four pixels. Hence, the AutoMapDecomposer applies the PPT setting *RW* to four pixels on the map to remove the foreground pixels that do not have any of the parallel patterns detected (i.e., single-line pixels) as shown in Figure 2.14, where the contour lines are removed.

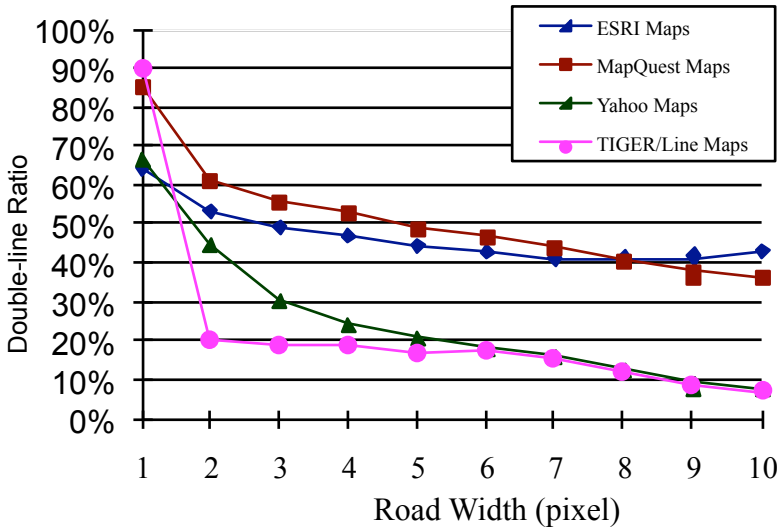
There are some exceptions to using the PPT to trace the parallel patterns. In Figure 2.15, the PPT detects *Pixel 1* to *Pixel 8* as double-line pixels since the gray pixels are the corresponding pixels of *Pixel 1* to *Pixel 8* in the horizontal and/or vertical directions. For the *Pixel A* to *Pixel D*, the PPT cannot detect a parallel pattern for any of the pixels, so they are removed. Although the removal of *Pixel A* to *Pixel D* results in gaps between the line segments, the PPT detects the majority of road pixels and the gaps will be reconnected in the next steps to generate the road geometry and detect the road intersections.

2.2 Automatic Detection of Road Intersections

After applying the AutoMapDecomposer, we have a road layer that contains broken lines because of the removal of overlapping features, such as characters or contour lines. Figure 2.16 shows an example TIGER/Line map after the text/graphics separation and

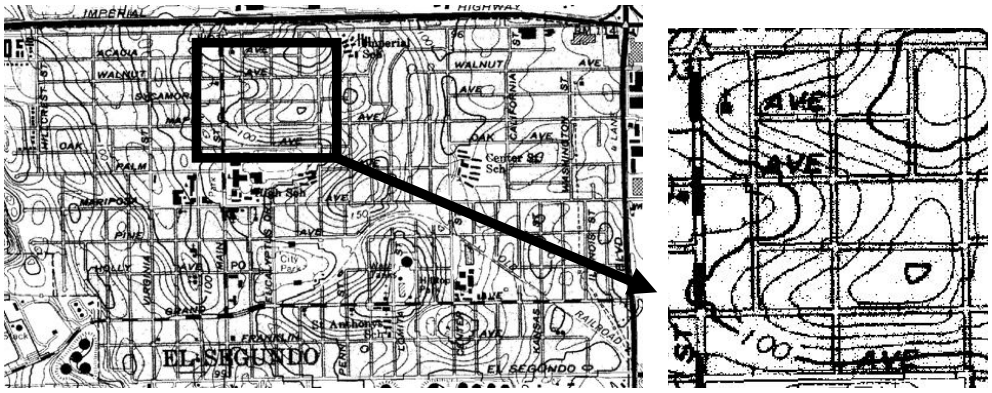


(a) Double-line format road layers

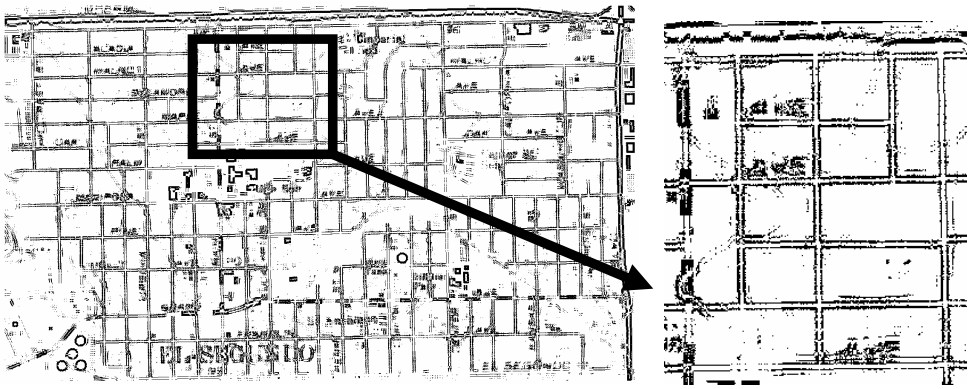


(b) Single-line format road layers

Figure 2.13: Using the Parallel-Pattern Tracing algorithm to detect road format and road width



(a) Before applying the PPT



(b) After applying the PPT

Figure 2.14: Applying the Parallel-Pattern Tracing algorithm on an example USGS topographic map

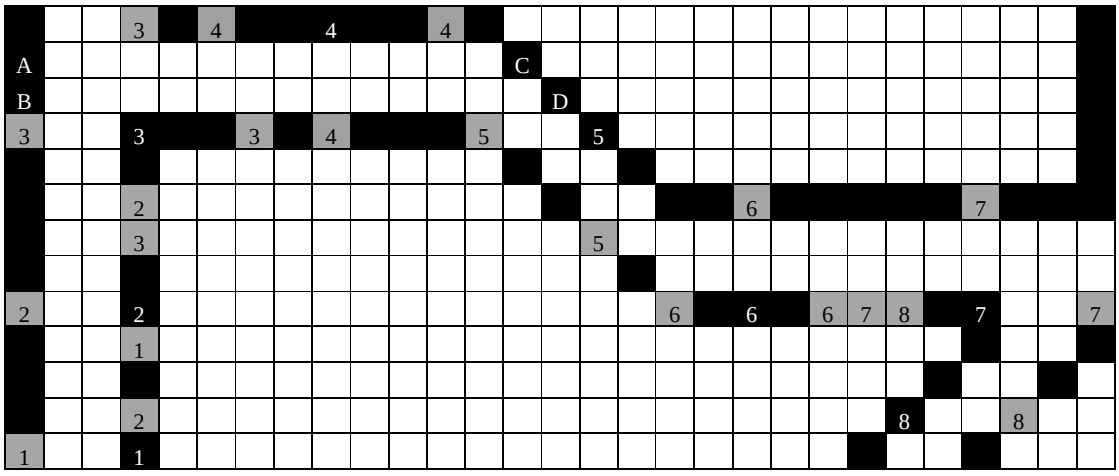


Figure 2.15: Examples of the Parallel-Pattern Tracing algorithm exceptions

the road lines are broken. Moreover, if the road layer is in double-line format, such as the example in Figure 2.6, we need to merge the parallel lines to produce the road geometry. The AutoIntDetector utilizes the binary morphological operators [Pratt, 2001] to reconnect the lines and build the road geometry for detecting the road intersections in the road layer.

(a) An example binary map

(b) Removing solid areas and small CCs

(c) Identifying strings and overlapping characters

(d) The extracted road layer

Figure 2.16: Separating the road layer using an example single-line map from TIGER/Line

2.2.1 Automatically Generating Road Geometry

The binary morphological operators are based on the hit-or-miss transformation with various size masks [Pratt, 2001], which are often used in various document analysis algorithms as fundamental operations [Agam and Dinstein, 1996]. The hit-or-miss transformation first uses a 3x3 pixels binary mask to scan over the input binary image. If the mask matches the underlying pixels, it is a "hit"; otherwise, it is a "miss". Each morphological operator uses a different mask to perform the hit-or-miss transformation and has a different action as the result of a "hit" or "miss".

2.2.1.1 Binary Dilation Operator

The effect of the binary dilation operator is to expand the region of foreground pixels [Pratt, 2001] and the AutoIntDetector uses it to thicken the lines and reconnect adjacent pixels. As shown in Figure 2.17, if a background pixel has any foreground pixel in its eight adjacent pixels (i.e., a "hit"), the dilation operator converts the background pixel to a foreground pixel (i.e., the action resulting from the "hit"). For example, Figure 2.18 shows that after two iterations, the binary dilation operator fixes the gap between the two lines. Moreover, if the roads are in double-line format, the binary dilation operator combines the two parallel lines to a single line. The number of iterations determines the maximum gap size that the AutoIntDetector needs to fix. For example, the AutoIntDetector can connect the gaps smaller than six pixels with three iterations of the binary dilation operator. For double-line raster maps, the AutoIntDetector selects the number of iterations based on the width of the roads in order to merge parallel lines. Figure 2.19(a) shows the result after performing the binary dilation operator on Figure 2.6(f) where the

parallel road lines are merged into single lines, except on intersection area that is larger than the dynamically generated dilation iterations. For single-line maps, the AutoIntDetector utilizes three iterations to fix gaps smaller than six pixels, which is usually enough to merge most of the gaps since the gaps result from removing small overlapping components. Figure 2.19(b) shows the result after performing three iterations of the binary dilation operator on Figure 2.16(d) where the gaps are connected.

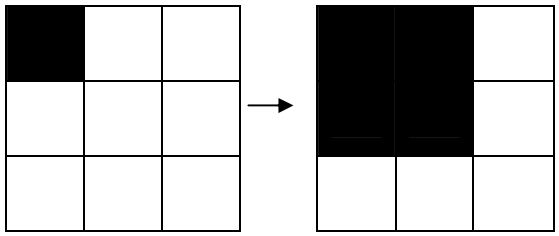


Figure 2.17: Dilation (background is shown in white cells)

2.2.1.2 Binary Erosion Operator

The idea of the binary erosion operator is to reduce the region of foreground pixels [Pratt, 2001]. The AutoIntDetector uses the binary erosion operator to reduce the area of each thickened line and maintain an orientation similar to the original orientation prior to

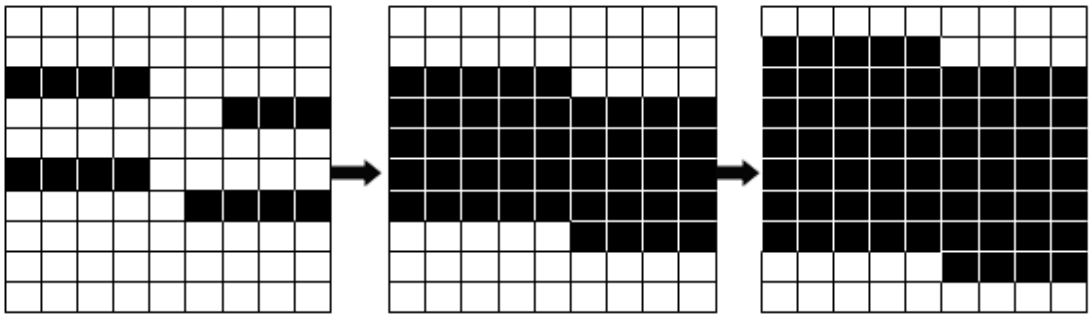
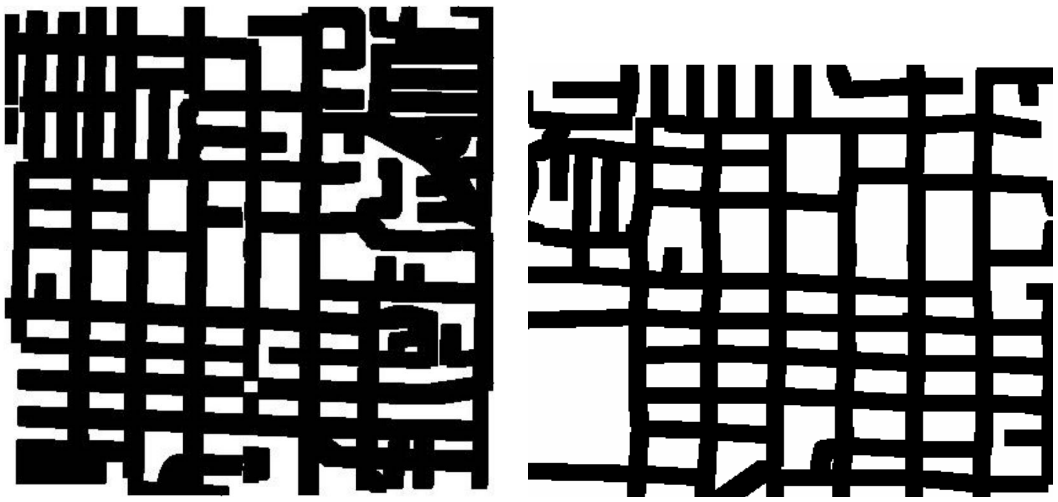


Figure 2.18: The effect of the binary dilation operator (background is shown in white cells)



(a) An example result of a double-line road layer from Yahoo Maps

(b) An example result of a single-line road layer from TIGER/Line

Figure 2.19: Example results after applying the binary dilation operator

applying the binary dilation operator. If a foreground pixel has any background pixel in its eight adjacent pixels (i.e., a "hit"), the binary erosion operator converts the foreground pixel to a background pixel (i.e., the action resulting from the "hit") as shown in Figure 2.20. For example, Figure 2.21 shows that after two iterations, the binary erosion operator reduces the width of the thickened lines. Figure 2.22(a) and Figure 2.22(b) show the results after applying the binary erosion operator on Figure 2.19(a) and Figure 2.19(b) respectively.

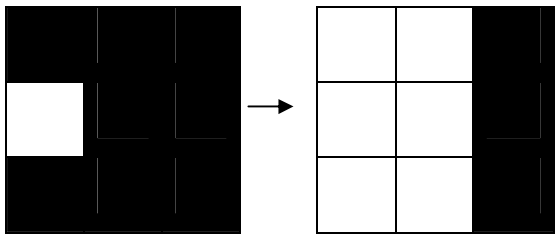


Figure 2.20: Erosion (background is shown in white cells)

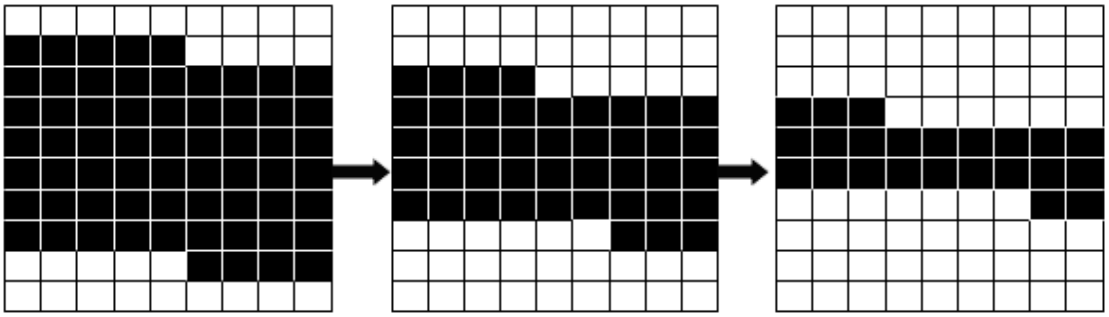
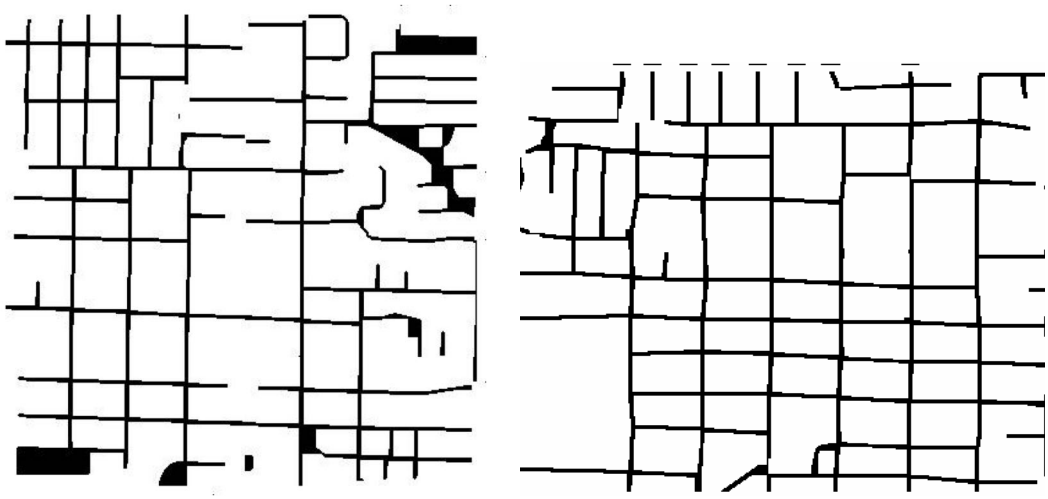


Figure 2.21: The effect of the binary erosion operator (background is shown in white cells)



(a) An example result of a double-line road layer from Yahoo Maps

(b) An example result of a single-line road layer from TIGER/Line

Figure 2.22: Example results after applying the binary dilation and erosion operators

2.2.1.3 Thinning Operator

After applying the binary dilation and erosion operators, the road lines are in various widths. To extract the centerlines of the roads to represent the road geometry, the AutoIntDetector utilizes the thinning operator to produce one-pixel-wide roads. Figure 2.23 shows the effect of the thinning operator. The AutoIntDetector does not use the thinning operator immediately after the binary dilation operator because the binary erosion operator has the opposite effect of the binary dilation operator, which prevents the orientations of the road lines near the intersections from being significantly distorted as shown in Figure 2.24. The AutoIntDetector utilizes a generic thinning operator that is a conditional erosion operator with an extra confirmation step [Pratt, 2001]. The first step of the thinning operator is to mark every foreground pixel that connects to one or more background pixels (i.e., the same idea as the binary erosion operator) as a candidate to convert to the background. Then, the confirmation step checks if the conversion of a candidate causes any disappearance of original line branches to ensure that the basic structures of the original objects are not compromised. Figure 2.25(a) and Figure 2.25(b) show the results after performing the thinning operator on Figure 2.22(a) and Figure 2.22(b), respectively.

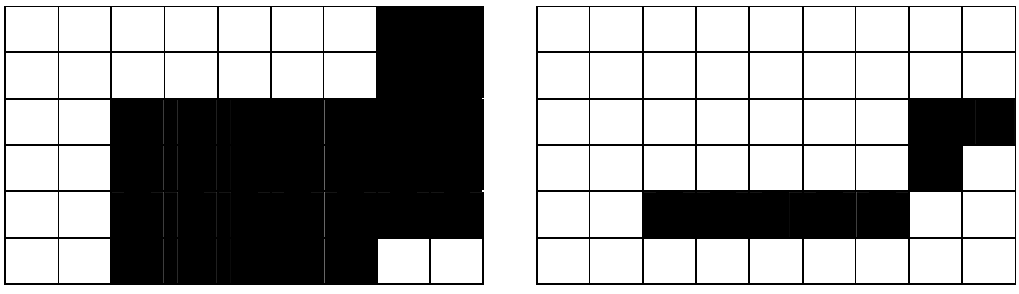
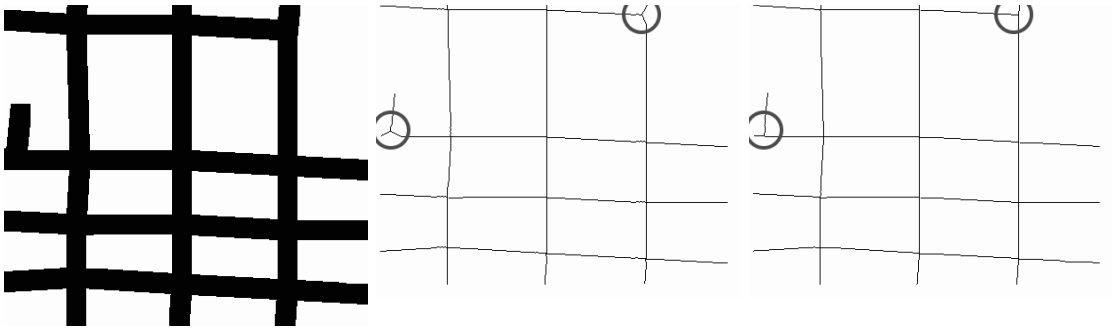
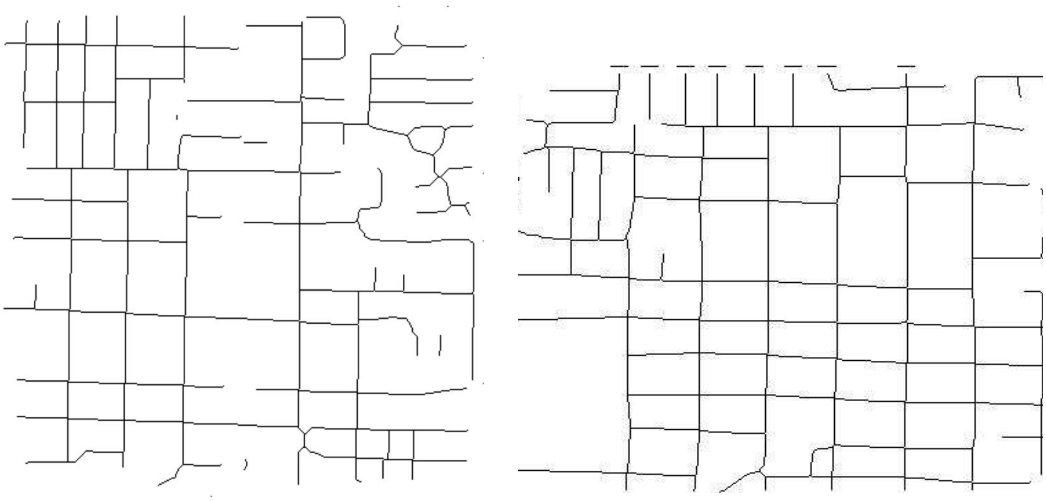


Figure 2.23: The thinning operator (background is shown in white cells)



(a) After the dilation (b) Without the erosion (c) With the erosion

Figure 2.24: Example results after applying the thinning operator with and without the erosion operator



(a) An example result of a double-line road layer from Yahoo Maps

(b) An example result of a single-line road layer from TIGER/Line

Figure 2.25: Example results after applying the binary dilation, binary erosion, and thinning operators

2.2.2 Automatically Detecting Road Intersections from Road Geometry

In this step, the AutoIntDetector automatically extracts the road intersections, connectivity (i.e., the number of roads that meet at an intersection), and orientations of the roads intersecting at each intersection from the road geometry.

2.2.2.1 Detecting Road-Intersection Candidates

A road intersection is a point at which more than two lines meet with different tangents. To detect possible intersection points, the AutoIntDetector starts by using an *interest operator*. The *interest operator* detects *salient points* in an image as starting points for image recognition algorithms. The AutoIntDetector uses the *interest operator* proposed by Shi and Tomasi [1994] and implemented in OpenCV[1] to find the *salient points* as the road-intersection candidates.

The *interest operator* checks the color variation around every foreground pixel to identify *salient points* and assigns a quality value based on the color variation to each identified *salient point*. For example, Figure 2.26 shows the *salient points* of *Pixel 1* to *Pixel 5*. *Pixel 1* and *Pixel 3* have higher quality values than other *salient points* since *Pixel 1* and *Pixel 3* have more intersecting lines resulting from higher color variations around the pixels.

The AutoIntDetector discards the *salient point* that lies within a predefined radius R of some *salient points* with higher quality values and uses the remaining *salient points* as the road-intersection candidates. With the radius R defined as five pixels, the AutoIntDetector discards *Pixel 2* and keeps *Pixel 1* as a road-intersection candidate since *Pixel*

[1] http://sourceforge.net/projects/opencvlibrary, GoodFeaturesToTrack function

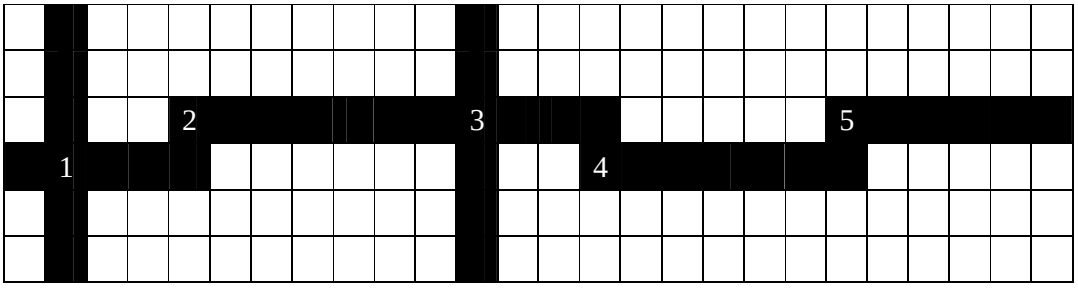


Figure 2.26: Examples of *salient points* (background is shown in white cells)

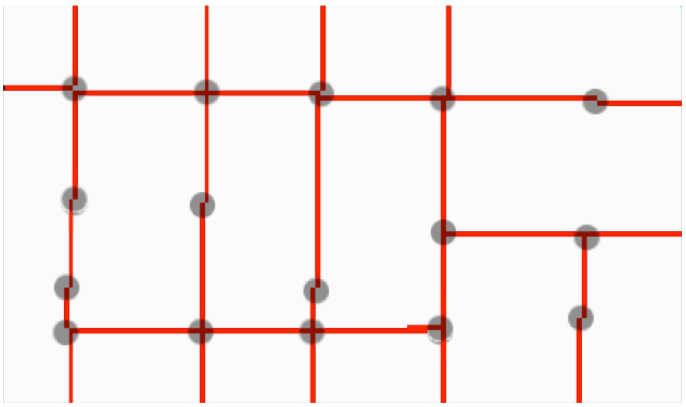
1 has a higher quality value. Similarly, the AutoIntDetector discards *Pixel 4* because it lies within the five-pixel radius of *Pixel 3*. The AutoIntDetector keeps *Pixel 5* as a road-intersection candidate since it is not close to any other *salient points* with a higher quality value. The radius R of five pixels is selected through experimentation. Considering the fact that road intersections are not generally near each other in a raster map, the radius of five pixels is reasonable for detecting road-intersection candidates.

2.2.2.2 Filtering Road-Intersection Candidates

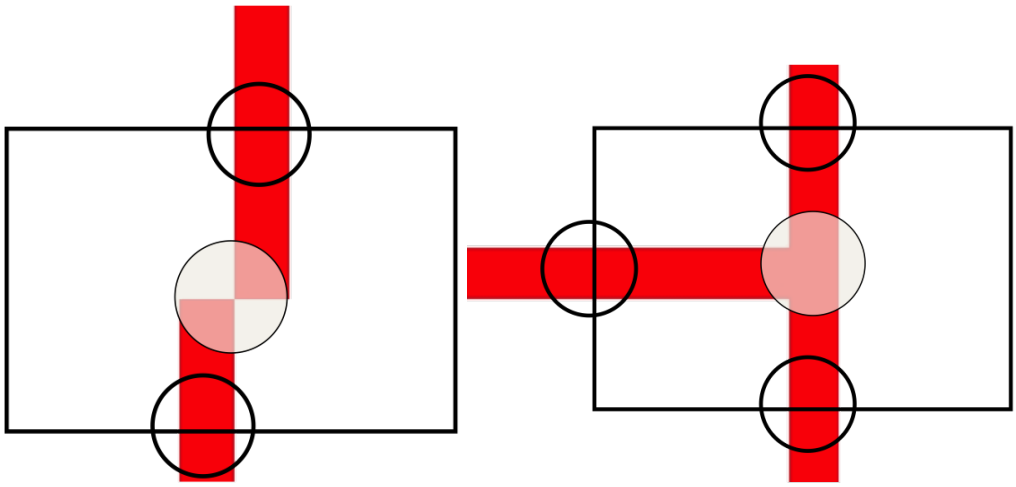
The identified road-intersection candidates are *salient points*, which can be a road pixel on a road line where the slope of the road suddenly changes, such as *Pixel 5* in Figure 2.26. Since every road intersection is a point where two or more line segments meet, the AutoIntDetector uses the connectivity of a *salient point* to determine actual road intersections among the identified road-intersection candidates.

The AutoIntDetector draws a rectangle around each road-intersection candidate, as shown in Figure 2.27, to determine the connectivity. The connectivity of a road-intersection candidate is the number of foreground pixels that intersects with this rectangle since the road lines are all one pixel wide. If the connectivity is less than three,

the AutoIntDetector discards the candidate; otherwise the AutoIntDetector identifies the candidate as a road intersection. Subsequently, since the road lines in a raster map are straight within a small distance (i.e., several meters), the AutoIntDetector links the road intersection to the intersected foreground pixels on the rectangle boundaries to compute the slopes (i.e., orientations) of the intersecting road lines as shown in Figure 2.28.



(a) Intersection candidates



(b) Not an intersection

(c) An intersection

Figure 2.27: The intersection candidates (gray circles) of a portion of an example map from TIGER/Line

The rectangles around road-intersection candidates should cover only straight road lines for the road orientation to be accurate. In the example shown in Figure 2.27,

the AutoIntDetector uses an 11x11-pixels rectangle on the raster map with resolution 2 meters per pixel assuming the road lines are straight within five pixels or ten meters (e.g., the width of the rectangle is a line of length 11 pixels divided into five pixels to the left, one center pixel, and five pixels to the right). Although the effective rectangle size can vary depending on the raster map and the map scale, the AutoIntDetector uses a small rectangle size to assure that even with raster maps of lower resolutions, the assumption that the rectangle should cover only straight road lines is still valid.

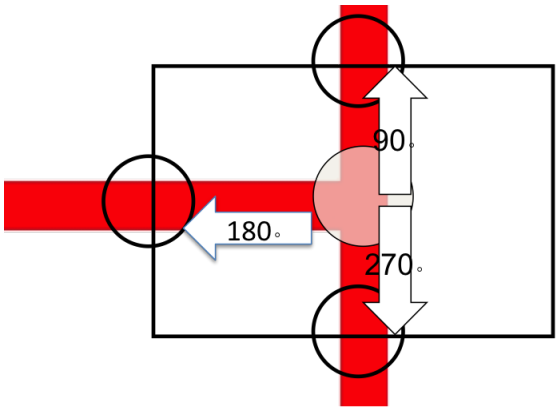


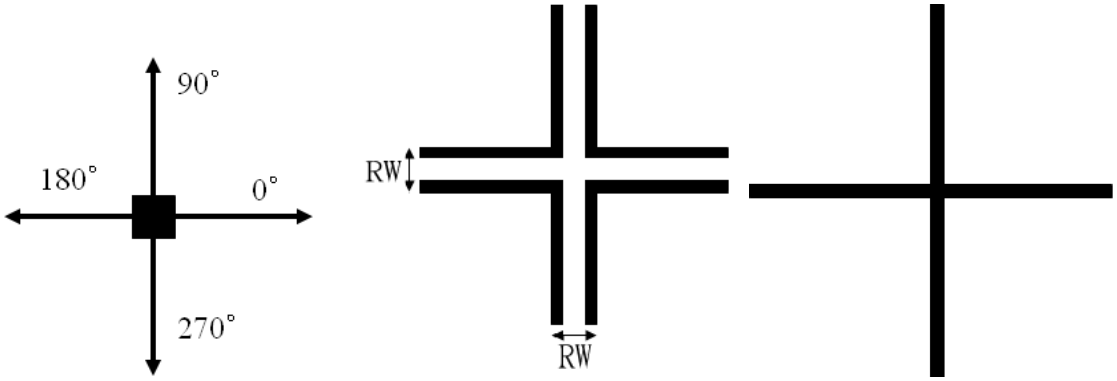
Figure 2.28: Constructing lines to compute the road orientations

The AutoIntDetector does not trace the pixels between the road intersection and the intersected pixels at the rectangle boundaries, which could introduce errors if the intersected pixels are from other road lines that do not intersect at the road intersection or if the road lines within the rectangle are not straight. This usually happens in low-resolution maps; however, in general, the rectangle is much smaller than the size of a street block, and it is unlikely to contain non-intersecting or non-straight road lines. Moreover, the AutoIntDetector saves significant computation time by avoiding the tracing of every possible road line between each of the road intersections and the rectangle boundaries.

2.2.2.3 Localized Template Matching (LTM)

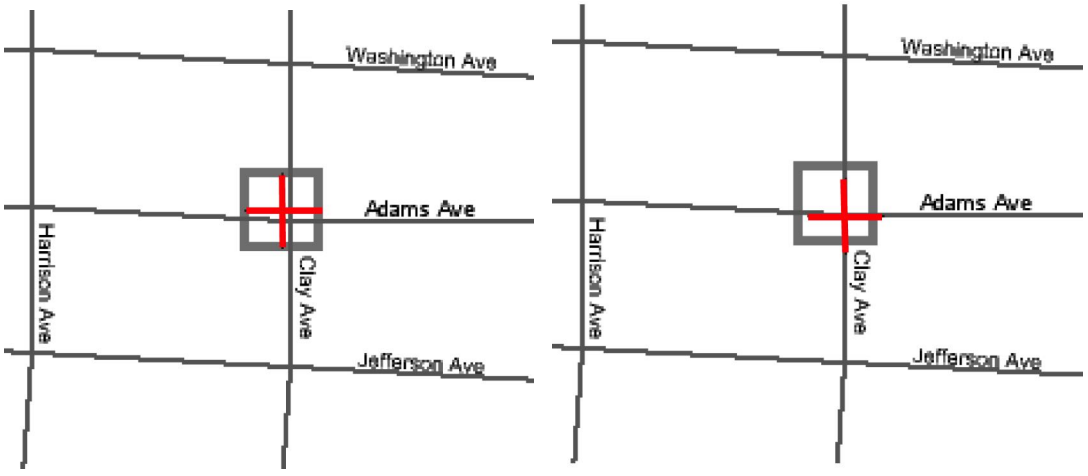
Because of the usage of the binary morphological operators, the AutoIntDetector might shift the road lines in the road geometry from their original positions or even create false branches. Therefore, after the AutoIntDetector extracts the road intersections, the algorithm uses the Localized Template Matching (LTM) algorithm [Chen et al., 2008] to improve the accuracy of the extracted road intersections.

For every extracted road intersection, the AutoIntDetector constructs a road template based on the road format, connectivity, and road orientations. For example, Figure 2.29(a) shows a road intersection with connectivity equal to four, and the orientations of the intersected roads are 0, 90, 180, and 270 degrees, respectively. If the raster map is a double-line map, the AutoIntDetector uses the road width detected by the PPT with one-pixel-wide parallel lines to construct the template as shown in Figure 2.29(b); otherwise, the AutoIntDetector uses one-pixel wide lines to construct a single-line template as shown in Figure 2.29(c). After constructing the templates, the AutoIntDetector utilizes the LTM to search locally from the positions of the extracted road intersection, as shown in Figure 2.30. The LTM locates regions in the binary raster map that are most similar in terms of the geometry to the templates. The outputs of the LTM are the positions of the matched templates and a set of similarity values. For a road intersection, if the similarity is larger than a similarity threshold, the AutoIntDetector adjusts the position of the intersection to the matched point determined by the LTM; otherwise the AutoIntDetector discards the road intersection.



(a) Extracted intersection (b) Double-line format (c) Single-line format

Figure 2.29: Example templates for the double-line and single-line road formats



(a) Searching within a local area (b) Identified a match

Figure 2.30: The Localized Template Matching algorithm

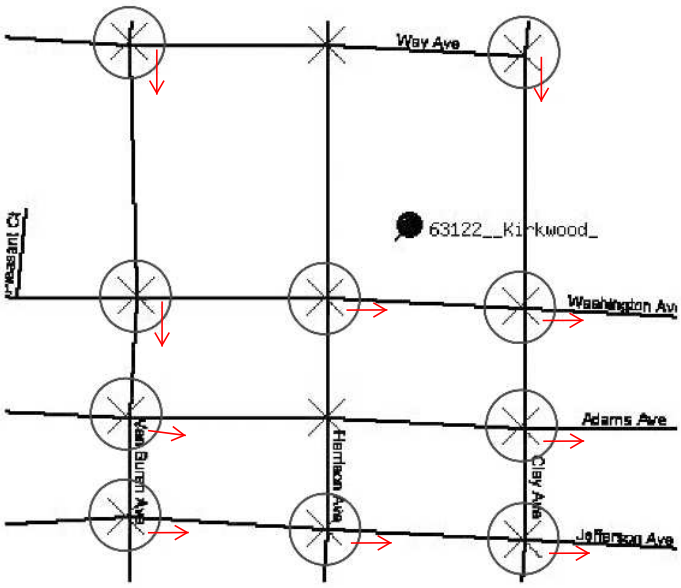
As shown in Figure 2.31, the AutoIntDetector adjusts the circled intersections to the precise location in the original raster map, and the arrows show the directions of the adjustments. The differences are only a few pixels, so the figures need to be studied carefully to see the differences. For example, in the upper-left circle of Figure 2.31(a), AutoIntDetector adjusts the intersection several pixels lower to match the exact intersection location in the original map; for the three circles on the bottom, AutoIntDetector moves the intersections to their right for exact matches.

2.3 Experiments

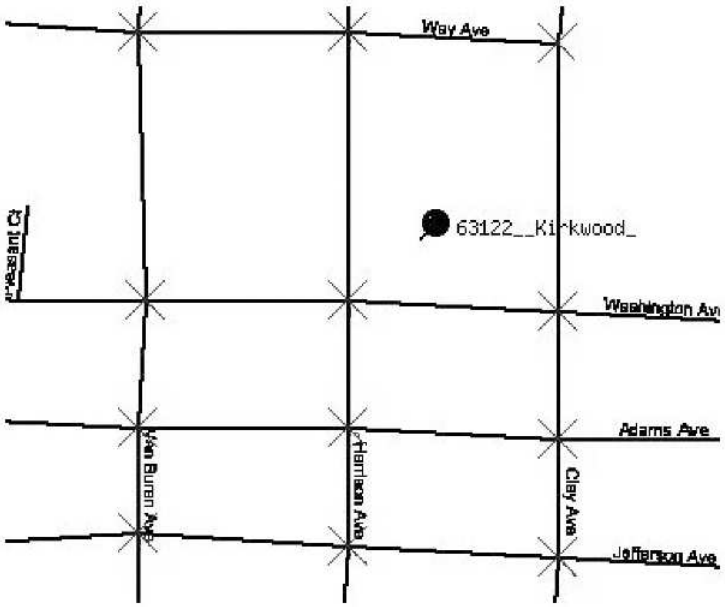
In this section, I report my experiments on the techniques described in this chapter using raster maps from various sources. I experimented with computer-generated maps and scanned maps from 12 sources, including ESRI Maps, MapQuest Maps, TIGER/Line, Yahoo Maps, A9 Maps, MSN Maps, Google Maps, Map24 Maps, ViaMichelin Maps, Multimap Maps, USGS topographic maps, and Thomas Brothers Maps, covering cities in the United States and some European countries.

I arbitrarily selected 80 detailed street maps with scales range from 1.85 to 7 meters per pixel.[2] In addition, I deliberately selected seven low-resolution maps with scales ranging from 7 to 14.5 meters per pixel to test my approach on complex raster maps that have significant overlaps between lines and characters.

[2]Some of the sources do not provide scale information.



(a) Before applying the LTM



(b) After applying the LTM

Figure 2.31: An example TIGER/Line map before/after applying the Localized Template Matching algorithm

2.3.1 Experimental Setup

To demonstrate the capability for handling a variety of maps, I did not use any prior information of the input maps in the experiment. Instead, I used a set of default parameters for all input maps based on an initial experiment on a small set of data (disjoint with my test dataset in this experiment). I could optimize these parameters for one particular source to produce the best results if prior knowledge of the sources were available.

In the text/graphics separation step, the size of a small connected component was 20x20 pixels. In the step to extract the road geometry using the morphological operators, the number of iterations for the binary dilation and erosion operators for a single-line map were three and two, respectively (i.e., a gap smaller than six pixels could be fixed). For a double-line map, I dynamically generated the number of iterations based on the detected road width. In the step to identify actual intersections, connectivity, and road orientations, the rectangle for calculating the connectivity was a 21x21-pixels box (10 pixels to the left, 10 pixels to the right, and one center pixel). In the LTM step, the similarity threshold was 50%.

2.3.2 Evaluation Methodology

The output of the AutoMapDecomposer was the separated road and text layers. The evaluation of a successful layer separation result was carried out by the feature recognition results: the AutoIntDetector could proceed on detecting road intersections from the separated road layer if and only if the AutoMapDecomposer successfully separated the road and text layer from raster maps.

The output of the AutoIntDetector was a set of road-intersection positions, the number of roads that meet at each intersection, and the orientation of each intersecting road. The AutoIntDetector output is the key for allowing a map conflation system (i.e., a conflation system that aligns a map to other geospatial data, such as orthoimagery) to determine the map extent and align the map to other geospatial data. Therefore, the following evaluation metrics of the road intersections are based on supporting an accurate map-alignment result, which are discussed in turn in this subsection.

The ground truth of a road intersection is defined as the intersection points of two or more road lines for single-line maps or any pixel inside the intersection areas where two or more roads intersect for double-line maps. A correctly extracted road intersection is defined as follows: if there exists a road intersection in the original raster map within a N-pixel radius to the extracted road intersection, the extracted road intersection is a correctly extracted road intersection. I report the results varying N from zero to five for a subset of test data in Section 2.3.3. For any other places in this section, I report the results using N equal to five pixels. From my observation, if an extracted intersection is within five-pixel radius to any road intersections in the original map, it is usually an intersection shifted during the extraction; otherwise it is more likely a false positive.

I report the precision (correctness) and recall (completeness)[3] for the accuracy of the extracted road intersections. The precision is given by:

$$Precision = \frac{Number of Correctly Extracted Road Intersections}{Number of Extracted Road Intersections} \tag{2.2}$$

[3]The terms precision and recall are common evaluation terminologies in information retrieval and correctness and completeness are often used alternatively in geomatics and remote sensing [Heipke et al., 1997].

The recall is given by:

$$Recall = \frac{NumberofCorrectlyExtractedRoadIntersections}{NumberofGroundTruthRoadIntersections} \tag{2.3}$$

The precision and recall of the intersection results represent the capability of extracting an abstract representation of the road geometry from a raster map. To achieve the map alignment in a map conflation system, high precision is required to support the matching process to identify the corresponding road network in the other geospatial dataset, such as imagery Chen et al. [2006a, 2008]. Chen et al. [2006a] report that an average of 76% precision can support the map system to align a set of 60 maps to the imagery.

After the map conflation system finds a corresponding road network in the other geospatial dataset for the map alignment, the next step is to transform the map to the geospatial dataset according to the corresponding locations of the road intersections in the map and the geospatial dataset. Therefore, I report the displacement quality of the road-intersection locations, which is the distance between the extracted road intersections and the ground truth. I randomly selected two maps from each source to examine the positional displacement.

The intersection positions, connectivity, and road orientations help to reduce the search space in a map conflation system for finding a match between the map and the other geospatial data [Chen et al., 2008]. Therefore, I evaluate the geometric similarity to evaluate the similarity between the ground truth in the binary map and the extracted intersection positions, connectivity, and road orientations.

For an intersection template T with an image size of w x h pixels and the binary raster map B, the geometric similarity is defined as:

$$GS(T) = \frac{\sum_{y=1}^{h}\sum_{x=1}^{w} T(x,y)B(X+x,Y+y)}{\sqrt{\sum_{y=1}^{h}\sum_{x=1}^{w} T(x,y)^2 \sum_{y=1}^{h}\sum_{x=1}^{w} B(X+x,Y+y)^2}} \tag{2.4}$$

where $T(x,y)$ is one, if the pixel at (x,y) in the image of the intersection template is a foreground pixel; otherwise $T(x,y)$ is zero. $B(x,y)$ is one, if the pixel at (x,y) in the binary map is a foreground pixel; otherwise $B(x,y)$ is zero. In other words, the geometric similarity is a normalized cross correlation between the template and the ground truth, which ranges from zero to one [Chen et al., 2006a].

2.3.3 Experimental Results

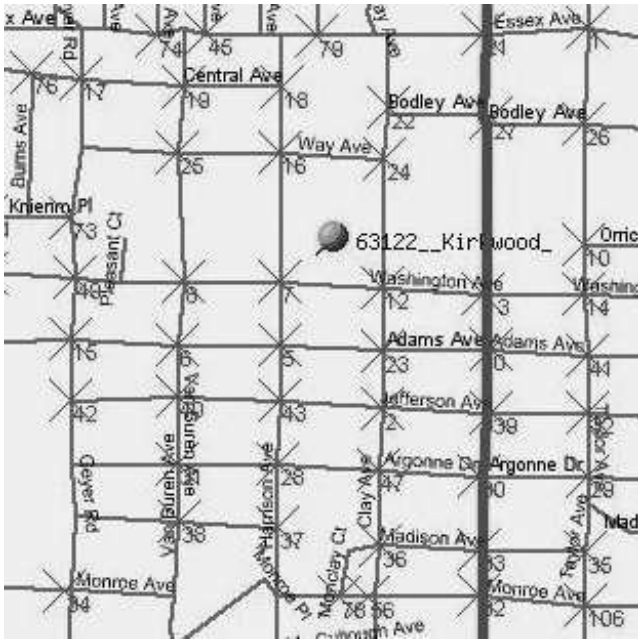
The AutoMapDecomposer successfully separated the road and text layers from every test map in the experiments and then the AutoIntDetector processed the separated road layers for detecting the road intersections. Table 2.2 shows the numerical results of the average numbers of the precision, recall, and F-measure of my experiments. The average precision was 95% and the recall was 75% using the set of parameters discussed in Section 2.3.1. In particular, for map sources like the A9 Maps, Map24 Maps, and ViaMichelin Maps, the precision numbers were 100% because of the fine quality of their maps (e.g., less noise and roads of the same width). In my experiments, the USGS topographic maps had the lowest precision and recall besides the low-resolution raster maps. This was because the USGS topographic maps had more feature layers than other map sources and the image quality of USGS topographic maps was not as good as the computer-generated raster maps due

to the scanning and compression processes. Figure 2.32 shows two example results from my experiments. In these figures, an "X" marks one extracted road intersection and the numbers were only for manually verifying the results.

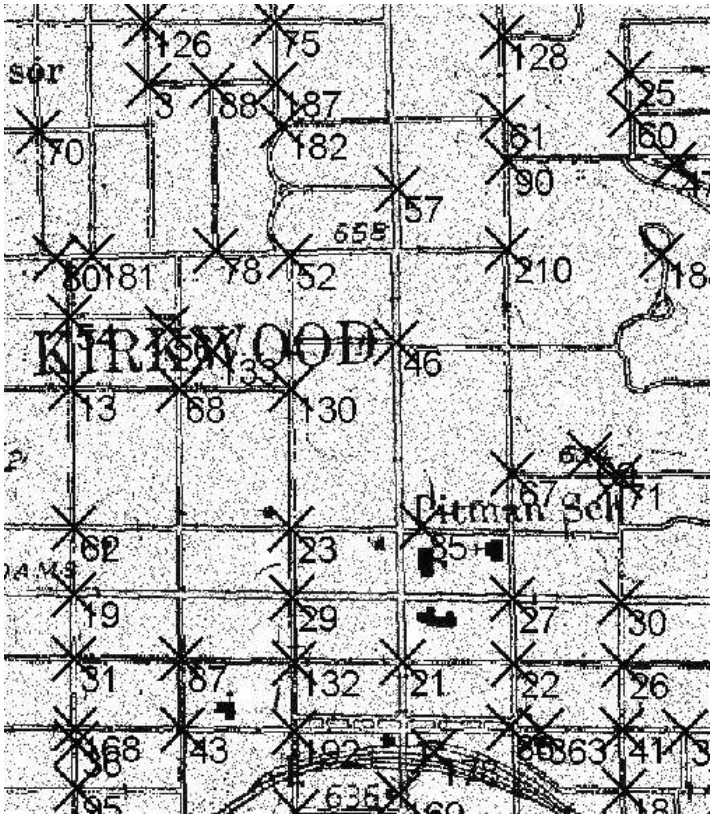
Map Source	**Map Count**	**Precision**	**Recall**	**F-Measure**
ESRI Maps	10	93%	71%	81%
MapQuest Maps	9	98%	66%	79%
TIGER/Line Maps	9	97%	84%	90%
Yahoo Maps	10	95%	76%	84%
A9 Maps	5	100%	93%	97%
MSN Maps	5	97%	88%	92%
Google Maps	5	98%	86%	91%
Map24 Maps	5	100%	82%	90%
ViaMichelin Maps	5	100%	98%	99%
Multimap Maps	5	98%	85%	91%
USGS topographic maps	10	82%	60%	69%
Thomas Brothers Maps	2	98%	65%	79%

Table 2.2: The average precision, recall, and F-measure for the various map sources

The average positional displacements for the 24 randomly selected maps (two from each source) was were pixels and the Root Mean Squared Error (RMSE) was 0.82, which shows that the majority of the extracted intersections were within one-pixel radius to the actual intersections in the map. Figure 2.33 shows the recall and precision using the positional displacement, N, which varies from zero pixels (i.e., the extracted intersection was at the exact location of the ground truth) to five pixels. Intuitively, the precision and recall were higher with a larger N since more extracted intersections were considered as correctly extracted ones, which means that for the applications that do not require the exact locations of the intersections, higher numbers for precision and recall could be achieved. For example, for an application that utilizes the extracted intersections as seed templates to search for road pixels in aerial imagery [Koutaki and Uchimura, 2004],



(a) An example TIGER/Line map



(b) An example USGS topographic map

Figure 2.32: Example results of the automatic road-intersection extraction

the application needs as many intersections as possible and has a low requirement for positional accuracy (i.e., N can be larger). For a map conflation system, the requirement for positional accuracy is high (i.e., a smaller N) since the conflation system needs accurate intersection positions to match another set of road intersections from a second source [Chen et al., 2008].

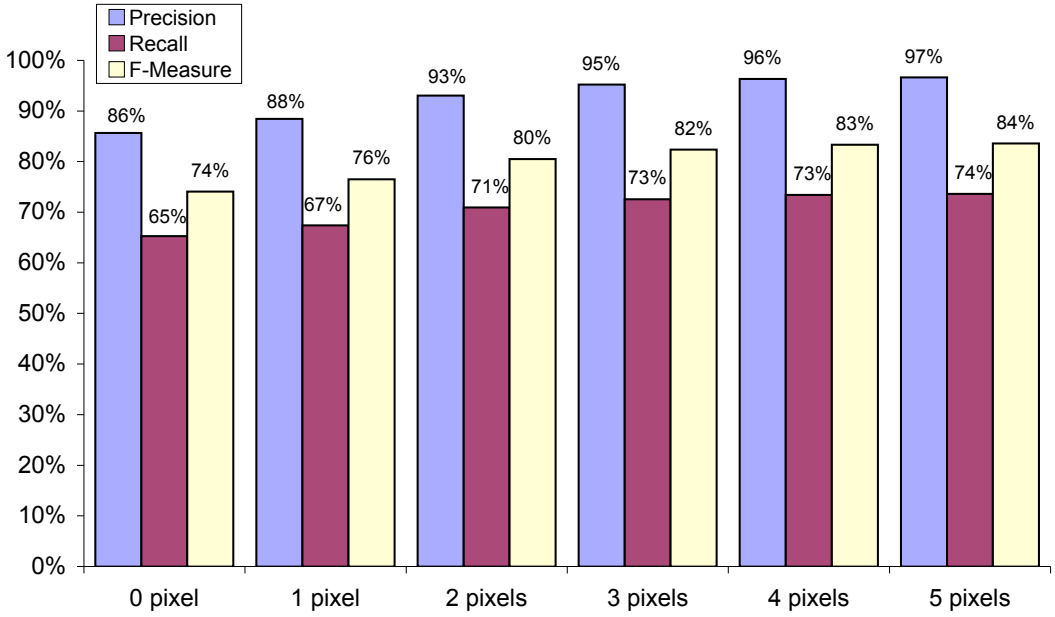


Figure 2.33: The precision and recall with respect to the positional displacements

The average geometric similarity between the extracted intersection and the ground truth is 72%. I could further improve the similarity by tracing the line pixels to generate the templates when identifying the connectivity, but pixel tracing would require more computation time. In the map conflation system built by Chen et al. [2008], they utilized the techniques described in this chapter and successfully used the road connectivity and orientations to reduce the time of finding a matching between two sets of road intersections from a raster map and imagery.

For comparison, I selected seven low-resolution maps (scales range from 7 meters per pixel to 14.5 meters per pixel) to conduct experiments on more complex raster maps with significant overlaps between lines and characters. Table 2.3 shows the experimental results of the seven low-resolution maps compared to the set of 80 high-resolution maps. The low-resolution maps (i.e., scales lower than 7 meters per pixel) had a significantly lower recall number. This was because the characters and symbols in the low-resolution maps overlap the lines more frequently, as shown in Figure 2.34. Since the text/graphics separation algorithm is used to remove characters and labels, the algorithm also removed many of the road lines in the low-resolution maps. Moreover, the size of street blocks in the low-resolution maps were sometimes smaller than the rectangle size used to determine the connectivity, which could lead to inaccurate identification of the road orientations.

Maps	**Precision**	**Recall**
Scales higher than 7 meters per pixel (80 maps)	95%	75%
Scales lower than 7 meters per pixel (7 maps)	83%	27%

Table 2.3: Experimental results with respect to the map scale

Figure 2.34: An example low-resolution raster map (TIGER/Line, 8 meters per pixel)

In comparison to the road-intersection-extraction results from closely related work [Habib et al., 1999; Henderson et al., 2009], Habib et al. [1999] work on maps that contain only road lines to extract the road intersections and their paper does not offer numeric results. Henderson et al. [2009] work on the road layer extracted using user specified colors from USGS topographic maps. The approach of Henderson et al. [2009] achieved 93% recall and 66% precision in an experiment using the USGS topographic maps. In comparison, my approach achieved 82% recall and 60% precision on USGS topographic maps. My recall number was lower because I did not employ any prior knowledge to separate the road layer from the USGS topographic maps and hence some of the road pixels might be missing during the automatic road-layer separation. In addition, the morphological operators could generate distorted road geometry during the process of merging parallel road lines. Although my experiments on the USGS topographic maps had a lower recall number, my approach is not limited to a specific type of map.

2.3.4 Computation Time

I built the experiment platform using Microsoft Visual Studio 2003 running on a Microsoft Windows 2003 Server powered by a 1.8 GHz Intel Xeon 4 Dual Processors CPU with one GB RAM. The USGS topographic maps were the most informative raster maps (i.e., more foreground pixels) in my experiments, which took less than one minute to extract the road intersections from an 800x600 pixels topographic map with around 120k foreground pixels. Other map sources required fewer than 20 seconds for an image size less than 500x400 pixels. The LTM algorithm took about 0.25 seconds to match an intersection point. The dominant factors for the computation time were the image size and the number

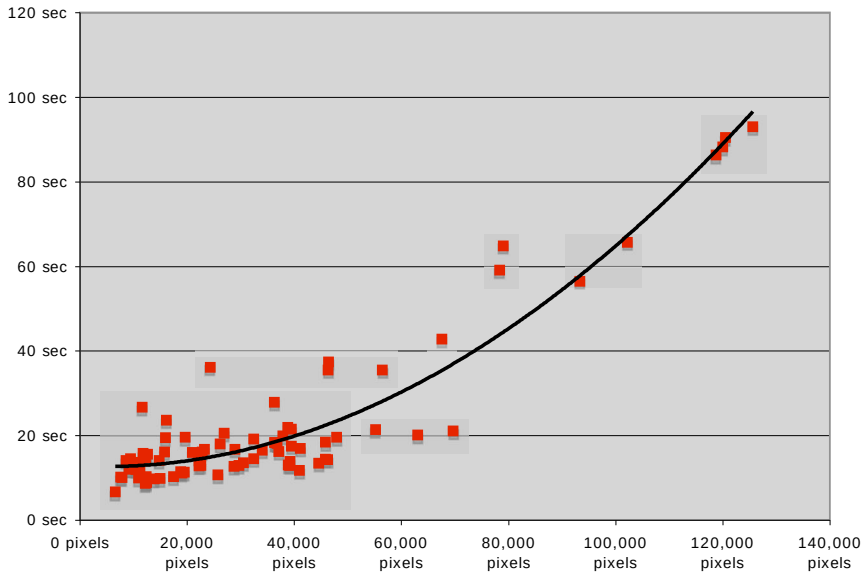


Figure 2.35: One of the dominant factors of the computation time is the number of foreground pixels

of foreground pixels in the raster maps. Figure 2.35 shows the computation time with respect to the number of foreground pixels in a raster map. The implementation was not fully optimized and improvements could still be made to speed up the processes, such as multi-threading for processing individual road intersections.

2.4 Conclusion

This chapter presented the AutoMapDecomposer and AutoIntDetector techniques, which automatically separate the road and text layers from raster maps and detect the road intersections from the separated road layer. My experiments on 87 maps from 12 sources show accurate results for extracting the positions and geometry of the road intersections. We utilize the road intersection results from a set of raster maps from various sources

and successfully identify the geospatial coordinates of the map and align the map to the orthoimagery [Chen et al., 2006a].

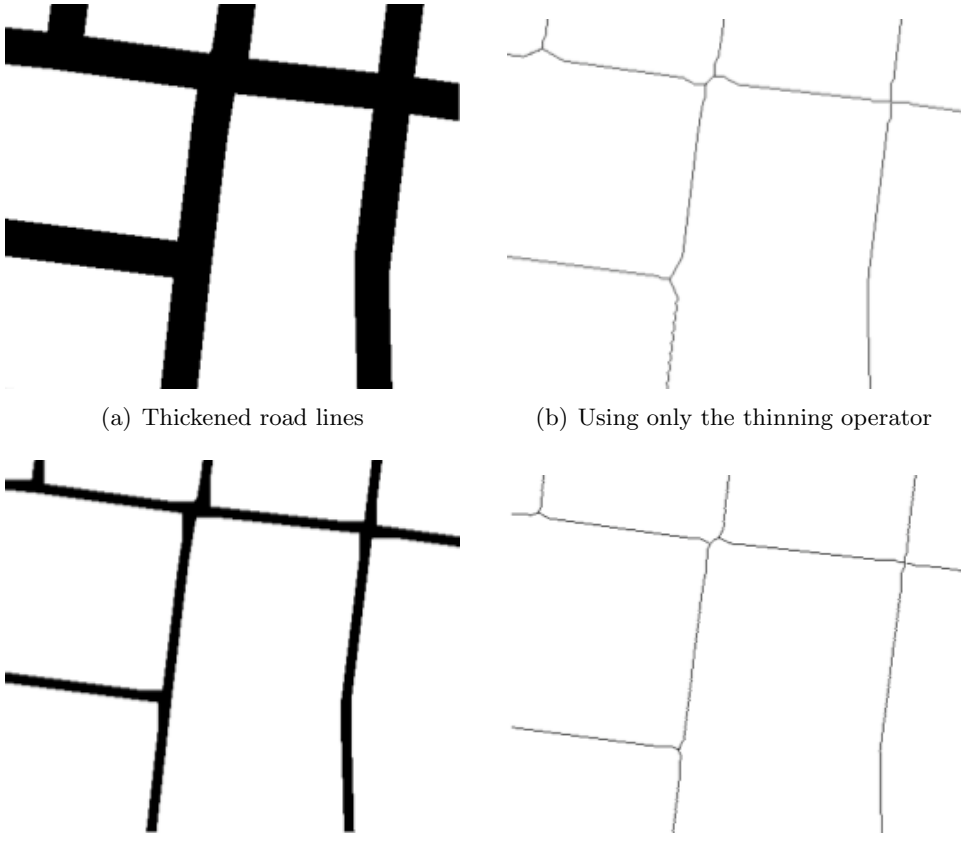
Chapter 3

Automatic Extraction of Road Vector Data

In Chapter 2, I described automatic techniques to separate the road and text layers from raster maps (AutoMapDecomposer) and extract road intersections from the separated road layers (AutoIntDetector). The AutoMapDecomposer employs an automatic grayscale-histogram-analysis algorithm to extract the foreground pixels from raster maps, which is based on the hypothesis that the background colors of raster maps have a dominant number of pixels and the number of foreground pixels is significantly smaller than the number of background pixels. However, for raster maps with poor image quality, the maps usually contain numerous colors due to the noise introduced in producing the maps in raster format (e.g., scanning). Therefore, the automatic grayscale-histogram-analysis algorithm does not work well on raster maps with poor image quality; even manual labeling requires significant effort to identify the colors that represent the road pixels.

In addition to the difficulty in processing raster maps with poor image quality, the AutoIntDetector utilizes the binary dilation, binary erosion, and thinning operators for generating the road geometry, which can cause distorted distorted lines. In particular, the thinning operator does not require any parameter tuning; however, the thinning operator

can produce distorted lines around intersections and hence the extracted road geometry is not accurate without manual adjustment [Bin and Cheong, 1998]. As shown in Figure 3.1, if we apply the thinning operator directly on the thick lines shown in Figure 3.1(a), the lines that are near intersections are seriously distorted as shown in Figure 3.1(b). To reduce the extent of the line distortion, the AutoIntDetector first erodes the lines using the erosion operator [Pratt, 2001] with a dynamically generated number of iterations and then applies the thinning operator. Figures 3.1(c) and 3.1(d) show that the extent of the line distortion is smaller after the AutoIntDetector applies the erosion operator; however, the erosion operator did not completely eliminate the distortion.



(a) Thickened road lines

(b) Using only the thinning operator

(c) Eroded road lines

(d) Using the erosion and thinning operators

Figure 3.1: Distorted road lines near road intersections caused by the thinning operator

This chapter presents a general approach for extracting road vector data from raster maps. I first present a supervised technique called the Road Layer Separator (RoadLayerSeparator) that requires only minimal user input for separating road layers from the maps with poor image quality. Then, I describe an automatic technique called the Automatic Road Vectorizer (AutoRoadVectorizer) for extracting accurate road vector data from the road layers. Together with the automatic techniques described in Chapter 2, the AutoMapDecomposer and AutoIntDetector, I can process raster maps with varying map complexity (e.g., overlapping features) and image quality to separate their road layers with minimal user input and automatically detect the road intersections and vectorize the road geometry from the separated road layers.

Figure 3.2 shows the overall approach of the RoadLayerSeparator and the AutoRoadVectorizer. RoadLayerSeparator identifies the road colors to extract road pixels from raster maps by analyzing user labels automatically. The user label does not have to contain only road pixels, which makes the user-labeling task easier and practical. With the separated road layer, the AutoRoadVectorizer utilizes an enhanced Parallel-Pattern-Tracing algorithm to detect the road width and format, which can process large road layers more efficiently since scanned maps are usually large images (a typical 350 dot-per-inch (DPI) scanned map can be larger than 6000x6000 pixels). Then, the AutoRoadVectorizer automatically generates the road geometry using the morphological operators as described in Chapter 2. Finally, the AutoRoadVectorizer automatically corrects the distorted lines around road intersections in the road geometry to produce accurate road vector data.

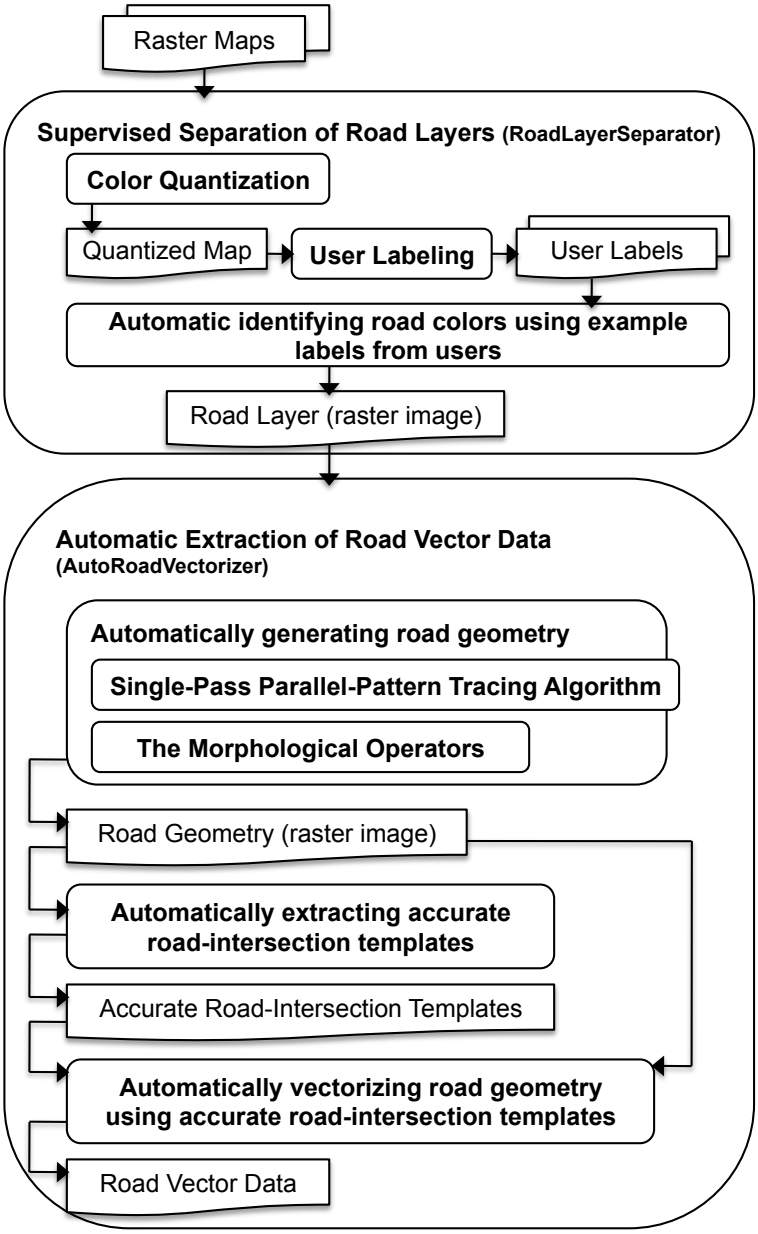


Figure 3.2: The overall approach of the RoadLayerSeparator and AutoRoadVectorizer

3.1 Supervised Separation of Road Layers

The RoadLayerSeparator has three major steps. The first step is to quantize the color space of the input image. Then, the user labels road areas of every road color in the quantized image. Since the quantized image has a limited number of colors, the RoadLayerSeparator can reduce the manual effort in this user-labeling step. Finally, the RoadLayerSeparator automatically identifies a set of road colors from the user labels and generates a color filter to extract the road pixels from the raster map. I describe the details of each step and the labeling criteria in the following subsections.

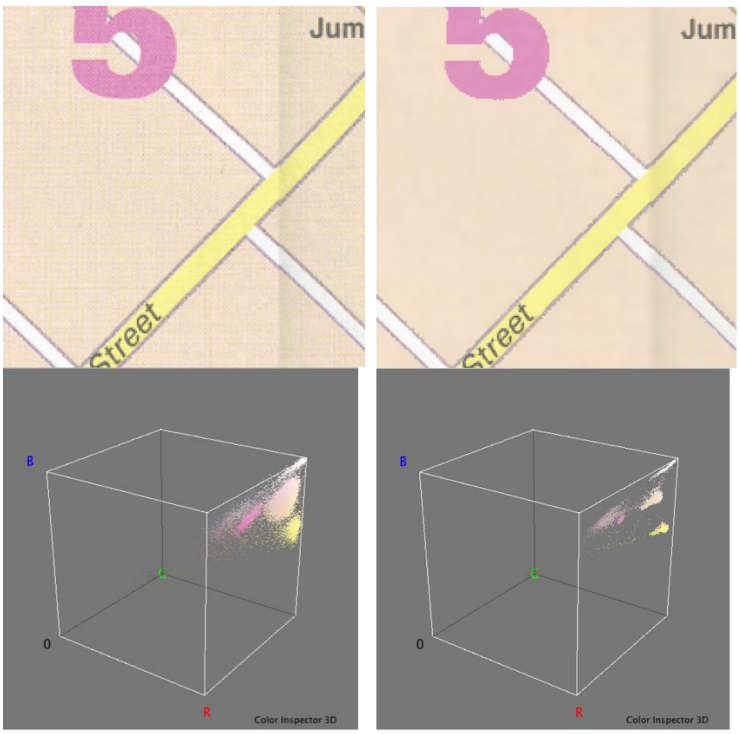
3.1.1 Color Quantization

Distinct colors commonly represent different layers (i.e., a set of pixels representing a particular geographic feature) in a raster map, such as roads, contour lines, and text labels. By identifying the colors that represent roads in a raster map, we can extract the road pixels from the map. However, raster maps usually contain numerous colors due to the scanning and/or compression processes and the poor condition of the original documents (e.g., color variation from aging, shadows from folding lines). For example, Figure 3.3(a) shows a 200x200 pixels tile cropped from a scanned map. The tile has 20,822 distinct colors, which makes it difficult to manually select the road colors. To overcome this difficulty, the RoadLayerSeparator applies color quantization algorithms to group the colors of individual feature layers into clusters based on the assumption that the color variation within a feature layer is smaller than the variation between feature layers.

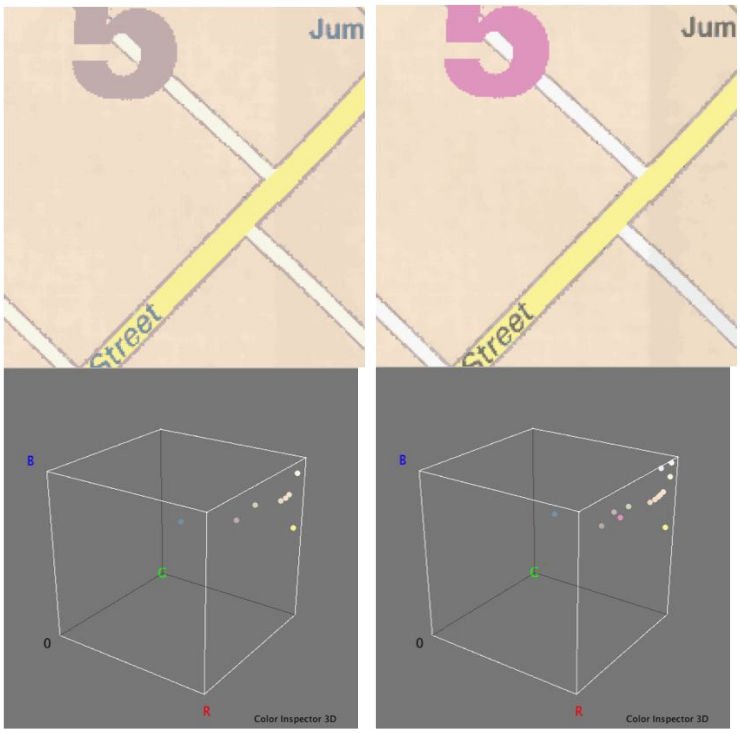
Therefore, after applying the color quantization algorithms, the RoadLayerSeparator can extract individual feature layers by selecting specific color clusters.

The RoadLayerSeparator first applies the Mean-shift algorithm [Comaniciu and Meer, 2002], which helps to preserve the edges of map features (e.g., road lines) while reducing noise. The Mean-shift algorithm works in a multi-dimensional space, which is a combination of the spatial space (i.e., the image coordinates, X and Y) and the HSL color space (hue, saturation, and luminance). The RoadLayerSeparator uses the HSL color space because the HSL space is a uniform color space. For a pixel in the raster map, $P(x, y)$, the corresponding node in the five-dimensional space (i.e., X and Y from the image coordinates plus H, S, and L from the color space) is $N(x, y, h, s, l)$, where h, s, and l represent the color of P.

To reduce the noise in the raster map, for a pixel, $P(x, y)$, the Mean-shift algorithm starts from computing the mean node, $M(x_m, y_m, h_m, s_m, l_m)$, from $N(x, y, h, s, l)$'s neighboring nodes. The mean node's position consists of the mean values on each of the axes X, Y, H, S, and L of N's neighboring nodes within a local area (the RoadLayerSeparator uses a spatial distance of 3 pixels and a color distance of 25 to define the local area). If the distance between $M(x_m, y_m, h_m, s_m, l_m)$ and $N(x, y, h, s, l)$ is larger than a small threshold (the RoadLayerSeparator uses the small threshold to limit the running time for the Mean-shift algorithm to converge), the Mean-shift algorithm shifts $N(x, y, h, s, l)$ to $M(x_m, y_m, h_m, s_m, l_m)$ and recalculates the mean node within the new local area. After the Mean-shift algorithm converges (the distance between the mean node and N is no longer larger than the threshold), the H, S, and L values of N is used as $P(x, y)$'s color. In



(a) An example tile (b) The Mean-shift result



(c) The K-means result, K=8 (d) The K-means result, K=16

Figure 3.3: An example map tile and the color quantization results with their red, green, and blue (RGB) color cubes

the example shown in Figure 3.3, the Mean-shift algorithm reduces the number of colors in Figure 3.3(a) by 72% as shown in Figure 3.3(b).

To further merge similar colors in the raster maps, the RoadLayerSeparator applies the K-means algorithm [Forsyth and Ponce, 2002] to generate a quantized image with at most K colors. The K-means algorithm can significantly reduce the number of colors in a raster map by maximizing the inter-cluster color variance; however, since the K-means algorithm considers only the color space, it is very likely that the resulting map has merged features with a small K. For example, Figure 3.3(c) shows the quantized map with K as eight and the text labels have the same color as the road edges. Therefore, the user would need to select a larger K to separate different features, such as in the quantized map in Figure 3.3(d) with K as 16.

For large maps with thousands of colors, the results after the Mean-shift algorithm can still have numerous numbers of colors and hence the K-means algorithm will require significant processing time to classify the colors. In this case, the RoadLayerSeparator utilizes a common color quantization method called the Median-cut [Heckbert, 1982] after the Mean-Shift algorithm to generate an image with at most 1,024 colors as the input to the K-means algorithm, which helps to reduce the processing time of the K-means algorithm. The Median-cut algorithm cannot replace the K-means algorithm because the design of the Median-cut algorithm is to be able to keep image details in the quantized image, such as the image texture, and the goal of my color quantization is to have a single color representing a single feature in the map (i.e., eliminating the image texture).

3.1.2 User Labeling

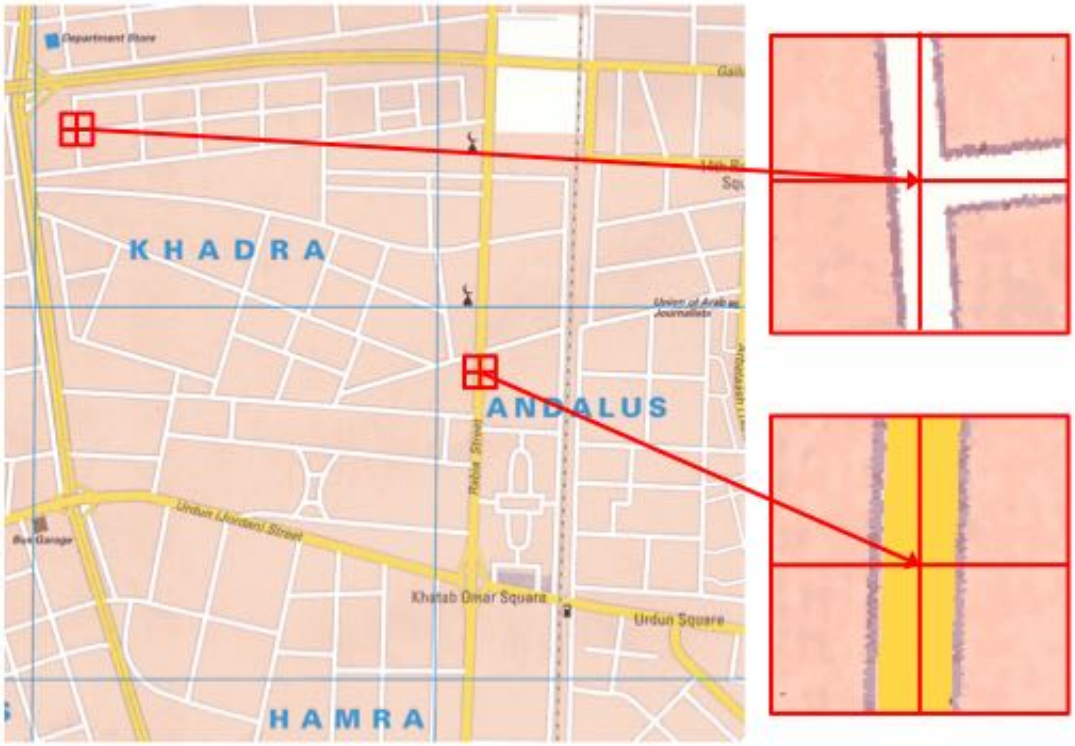
In this user-labeling step, the RoadLayerSeparator first generates a set of quantized maps in different quantization levels using various K in the K-means algorithm. Then, the user selects a quantized map that contains road lines in different colors from other features and provides a user label for each road color in the quantized map. A user label is a rectangle that should be large enough to cover a road intersection or a road segment. To label the road colors, the user first selects the size of the label. Then, the user clicks on the approximate center of a road line or a road intersection to indicate the center of the label. The user label should be (approximately) centered at a road intersection or at the center of a road line, which is the constraint the RoadLayerSeparator exploits to identify the road colors in the next step. For example, Figure 3.4(a) shows an example map and Figure 3.4(b) shows the quantized map and the labeling result to extract the road pixels. The two user labels cover one road intersection and one road segment, which contain the two road colors in the quantized map (i.e., yellow and white).

3.1.3 Automatically Identifying Road Colors Using Example Labels

Each user label contains a set of colors, and some of the colors represent roads in the raster map. The RoadLayerSeparator exploits two geometric properties of the road lines in a user label to identify the road colors of a given user label, namely the centerline property and the neighboring property.

The centerline property is: because the user labels are centered at a road line or a road intersection, the pixels of a road color will be a portion of one or more linear objects that are near the image center. For example, the RoadLayerSeparator decomposes a user label

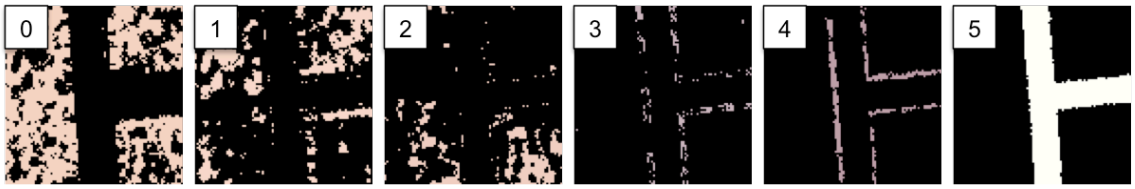
(a) An example scanned map



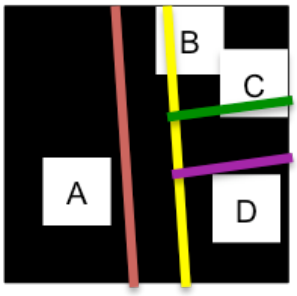
(b) The quantized map and user labels

Figure 3.4: An example of the supervised extraction of road pixels

shown in the top-right of Figure 3.4(b) into a set of six images shown in Figure 3.5(a) (background is shown in black) so that every decomposed image each contains only one color from the user label. The pixels in the decomposed images, *Image 3*, *4*, and *5*, constitute the road lines in the user label, and their pixels constitute a portion of several linear objects that pass through or near the image centers. For example, in *Image 3* and *4*, the pixels are a portion of four linear objects that pass near the image center as the four lines A, B, C, and D shown in Figure 3.5(b).



(a) The decomposed images



(b) Four linear objects

Figure 3.5: Identifying road colors using the centerline property (background is shown in black)

The neighboring property is: the pixels of road colors should be spatially near each other. For example, the majority of pixels in *Image 3* can find an immediate neighboring pixel in *Image 4* and *5* and vice versa, but the majority of pixels in *Image 0* cannot find an immediate neighbor in *Image 3*, *4*, and *5*.

To exploit the centerline property, for each decomposed image, the RoadLayerSeparator extracts the skeletons of every connected object in the image using the thinning operator and applies the Hough transformation [Forsyth and Ponce, 2002] to identify a set of Hough lines from the skeletons. The Hough transformation is a technique to identify lines (i.e., the Hough lines) that are constituted from imperfect objects. In our case, the imperfect objects are the non-background pixels in the decomposed images. Since the image center of a user label is the center of a road line or a road intersection, if the detected Hough lines are near the image center, the lines are most likely to be part of the road lines. Hence, the RoadLayerSeparator detects the Hough lines in each decomposed image and computes the average distance between the detected Hough lines to the image center as a measure to determine if the foreground pixels (non-black pixels) in a decomposed image represent roads in the raster map.

Figure 3.6 shows the detected Hough lines of each decomposed image, where the Hough lines that are within a distance threshold to the image centers are drawn in red and others are drawn in blue (the colors are only used to help explain the idea). In Figure 3.6, the decomposed images that contain road pixels (*Image 3*, *4*, and *5*) have more red lines than blue lines and hence the average distances between their Hough lines to their image centers are smaller than the other decomposed images. Therefore, *the decomposed image that has the smallest average distance is classified as the road-pixel image (i.e., the color of the foreground pixels in the decomposed image represents roads in the raster map).* The other decomposed images with average distances between one pixel to the smallest average distance are also classified as road-pixel images. This criterion allows the user label to be a few pixels off (depending on the size of the user label)

from the actual center of the road line or road intersection in the map, which makes the user labeling easier. In our example, *Image 5* has the smallest average distance, so the RoadLayerSeparator first classifies *Image 5* as a road-pixel image. Then, since *Image 4* is the only image with its average distance within a one-pixel distance to the smallest average distance, the RoadLayerSeparator also classifies *Image 4* as a road-pixel image.

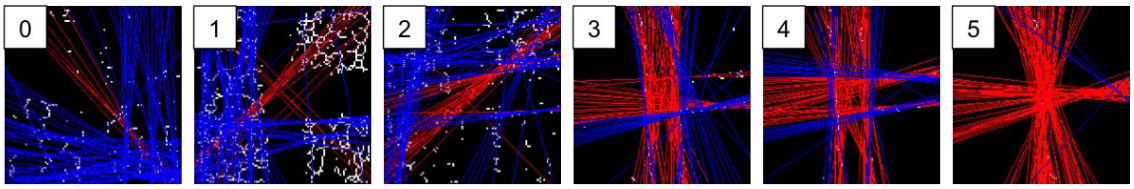


Figure 3.6: The identified Hough lines (background is shown in black)

The road-pixel image classification method based on the detected Hough lines relies on the numbers of detected Hough lines. So if a color that is used on roads has a smaller number of pixels compared to the major road colors, such as the *Image 3* shown in Figure 3.5(a), the image will not be classified as a road-pixel image using the Hough-line method. Therefore, I present the Edge-Matching algorithm for the RoadLayerSeparator to exploit the neighboring property to determine if any of the decomposed images that are not classified as road-pixel images using the Hough-line method (i.e., *Image 0* to *3*) is a road-pixel image.

The Edge-Matching algorithm utilizes a road template generated using the already classified road-pixel images and compares the unclassified images with the road template to identify road-pixel images. In our example, the road template is the combination of *Image 4* and *5* as shown in Figure 3.7 (background is shown in black). Next, the Edge-Matching algorithm uses the road template to evaluate *Image 0* to *3* in turn. For a

color pixel, $C(x, y)$, in a given decomposed image to be evaluated, the Edge-Matching algorithm searches a 3x3 pixels neighborhood centered at (x, y) in the image of the road template to detect if there exists any road pixels. If one or more neighboring road pixels exist, the Edge-Matching algorithm marks the pixel $C(x, y)$ as a road pixel since it is spatially near one or more road pixels. After the Edge-Matching algorithm examines every foreground pixel in a given decomposed image, if more than 50% of the foreground pixels in that image are marked as road pixels, then the decomposed image is identified as a road-pixel image.

Figure 3.7: A road template example (background is shown in black)

Figure 3.8 shows an example of the Edge-Matching algorithm. The first row shows the foreground pixels of the *Image 0* to *3* (background is shown in black) and the second row is the match with the road template. The bottom row shows the results after the Edge-Matching algorithm processes each of the images, where the non-black pixels are the matched pixels. Only *Image 3* has more than 50% of its foreground pixels identified as matched pixels, so that *Image 3* is a road-pixel image and others are discarded.

The RoadLayerSeparator processes every user label and identifies a set of road-pixel images from each label. Then, the algorithm uses the foreground colors in the road-pixel images (i.e., the identified road colors) to generate a color filter for scanning the

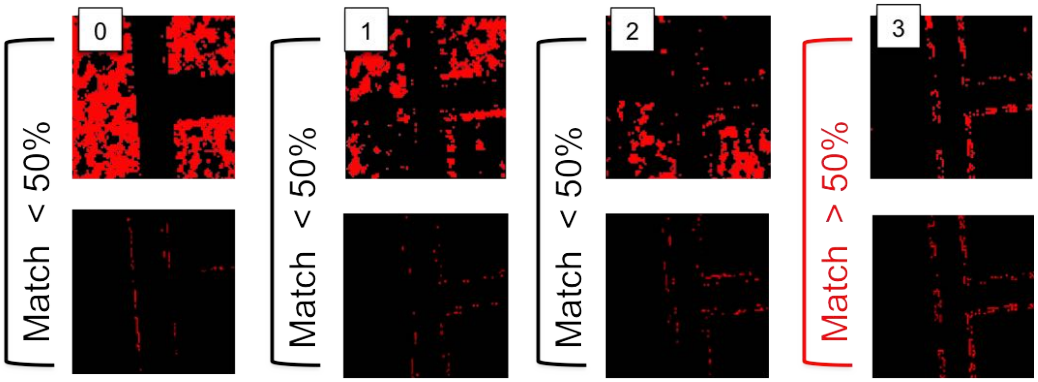


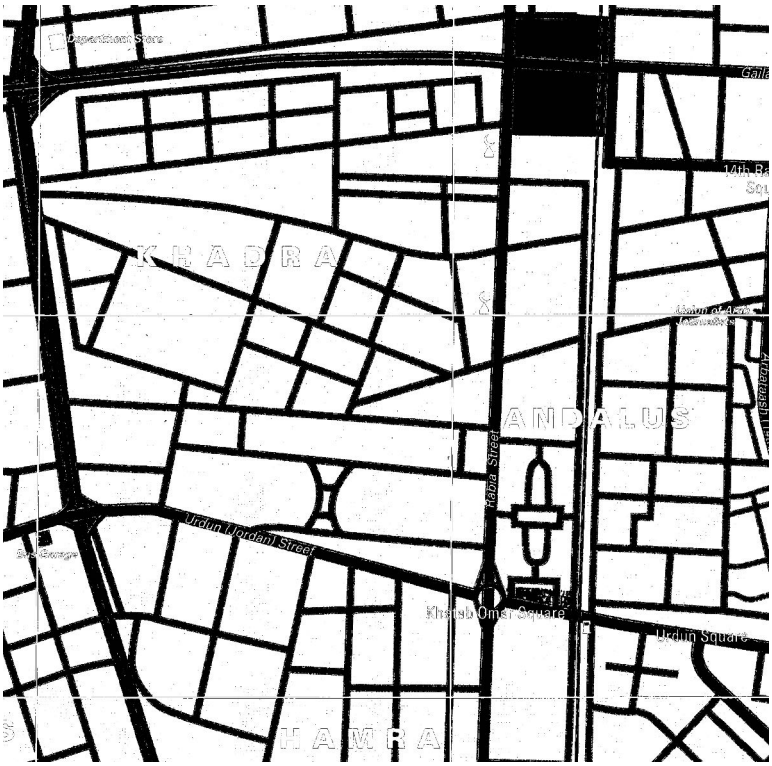
Figure 3.8: The Edge-Matching results (background is shown in black)

quantized map to extract the road pixels. The RoadLayerSeparator fills up the pixels in the quantized map where colors in the color filter are shown with black pixels (i.e., the road pixels) and the other pixels with white pixels. Figure 3.9(b) shows the extracted road pixels (i.e., the separated road layer) of Figure 3.9(a).

3.2 Automatic Extraction of Road Vector Data

In this section, I present the AutoRoadVectorizer, which automatically generates the road geometry from the separated road layer and then vectorizes the road geometry. Since scanned maps are usually large images, the AutoRoadVectorizer includes an enhanced Parallel-Pattern-Tracing algorithm for detecting the road width and road format of large maps more efficiently. Then, the AutoRoadVectorizer exploits the techniques from Chapter 2 to identify the road centerlines from the extracted road pixels as the road geometry automatically. The AutoRoadVectorizer corrects the distortions near road intersections by extracting accurate road-intersection templates and then generates accurate road vector data. Finally, instead of processing the entire map at once, I present

(a) An example scanned map



(b) The separated road layer (background is shown in white)

Figure 3.9: An example results of the RoadLayerSeparator

the divide-and-conquer approach of the AutoRoadVectorizer that divides the map into overlapping tiles and extracts and merges the road vector data from each tile.

3.2.1 Single-Pass Parallel-Pattern Tracing (SPPT)

In Chapter 2, I described the Parallel-Pattern-Tracing algorithm (PPT) for identifying the road format and road width. The PPT checks each foreground pixel to determine if there exists any corresponding pixel in the horizontal and vertical directions at a certain road width. If the PPT finds a corresponding pixel in each direction, the algorithm classifies the pixel as a parallel-pattern pixel of the given road width. By applying the PPT iteratively from one-pixel wide road width to K-pixel wide, we can identify the road width and road format of the majority of the roads in the map by analyzing the number of parallel-pattern pixels at each of the tested road widths.

The implementation of the PPT contains two convolution masks. One convolution mask works in the horizontal direction and the other one works in the vertical direction to find the corresponding pixels. The sizes of the convolution masks are based on the road width. For example, if the road width is one-pixel wide, the size of the convolutions mask is 3x3 pixels. If the road width is two-pixel wide, the size of the convolution masks is 5x5 pixels. Therefore, for N foreground pixels, to iteratively apply the PPT on the road width from one-pixel wide to K-pixel wide, the number of steps is:

$$PPT(N, K) = N * \sum_{r=1}^{K} (2r + 1)^2 \tag{3.1}$$

The time complexity is $O(NK^3)$, which requires significant computing time when we have a large map (a large N) and/or we run the PPT with more iterations (a large K).

To improve the time complexity, I present the Single-Pass Parallel-Pattern-Tracing algorithm (SPPT), which only requires a single-pass scan on the image. Figure 3.10 shows the pseudo-code of the SPPT. The SPPT starts from the upper-left pixel in the image and scans the image one row at a time from left to right.

```
// The image width and height
width, height;
// Test from one-pixel to K-pixel wide
K;
// The boolean arrays storing the information about the existence of corresponding pixels
 horizontal = new boolean[width * height][K]; vertical = new boolean[width * height][K];
// The SPPT results: the number of parallel- pattern pixels at each road width
ppt_pixel_count = new int[K];
```

```
Function void SPPT()
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        if (IsForeground(x,y)) {
            int idx = y * width + x;
            for (int i = 0; i < K; i++) {
                boolean h = horizontal[idx][i];
                if (x + i < width) { // Within the image
                    horizontal [y * width + x + i][i] = true;
                    if (h || IsForeground(y, x + i) h = true;
                    else h = false;
                }
                boolean v = vertical[idx][i];
                if (y+i < height) { // Within the image
                    vertical[(y + i) * width + x, i] = true;
                    if (v || IsForeground(y + i, x) v = true;
                    else v = false;
                }
            if (h && v) ppt_pixel_count[i]++;
        } // end for
      } // end if
    } // end for
} // end for
```

```
/* Check if a corresponding pixel in the horizontal direction has been found (h is true) or check the existence of a foreground pixel to the left */
```

```
/* Check if a corresponding pixel in the vertical direction has been found (v is true) or check the existence of a foreground pixel towards the bottom */
```

Figure 3.10: The pseudo-code of the Single-Pass Parallel-Pattern Tracing (SPPT) algorithm

To determine the number of parallel-pattern pixels in the image at the road width from one-pixel wide to K-pixel wide, the SPPT scans every foreground pixel's neighboring pixels in the horizontal and vertical directions. For the horizontal direction, the SPPT starts from the neighboring pixel to the left of a foreground pixel and scans K pixels. For the vertical direction, the SPPT starts from the neighboring pixel below a foreground pixel and scans K pixels. For a foreground pixel, P, if the SPPT detects a corresponding pixel, C, at P's C_{th} neighbor to the left or towards the bottom, the SPPT records that P and C both have at least one corresponding neighbor in the horizontal/vertical direction at the road width equal to C, which eliminates searching in the pixel's right/top direction. Therefore, for N foreground pixels, to iteratively apply the SPPT on the road width from one-pixel wide to K-pixel wide, the number of steps is:

$$SPPT(N, K) = 2 * N * K \tag{3.2}$$

The time complexity is $O(NK)$, which is significantly less than using the convolution masks and enables efficient processing of large maps.

3.2.2 Automatically Generating Road Geometry

With the detected road width and road format, the AutoRoadVectorizer exploits the techniques from Chapter 2 to automatically identify the road centerlines by first applying the dilation operator using the iteration as half of the detected road width to generate thickened road lines. If the road format is double-line format, the iteration of the dilation operator is half of the detected road width plus three for merging the parallel lines into

one line. The AutoRoadVectorizer applies more iterations of the dilation operator for the roads in double-line format because the algorithm needs to expand the road areas farther for filling up areas within road intersections of the parallel lines. Three more iterations fill areas smaller than 6x6 pixels, which is generally large enough to cover the holes in the intersection areas of parallel lines in all orientations after applying the dilation operator using half of the detected road width as the number of iterations. Then, the AutoRoadVectorizer applies the erosion operator with the number of iterations as the detected road width to thin the road lines for minimizing the distortion after applying the thinning operator next. Finally, the AutoRoadVectorizer uses the thinning operator to generate the road centerlines (i.e., the thinned road lines) as the road geometry.

3.2.3 Automatically Extracting Accurate Road-Intersection Templates

A road-intersection template is the position of a road intersection, the orientations of the roads intersecting at an intersection, and the connectivity of a road intersection. Figure 3.11 shows the overall approach to automatically extract accurate road-intersection templates, where the gray boxes indicate the results from using the SPPT algorithm and the Automatic Intersection Detector (AutoIntDetector) technique from Chapter 2.

3.2.3.1 Labeling Potential Distortion Areas

Since the thinning operator produces distorted road geometry near the road intersections and the road width determines the extent of the distortion, the AutoRoadVectorizer can utilize the extracted road intersections and the road width to label the locations of the potential distorted lines. The AutoRoadVectorizer first generates a blob image

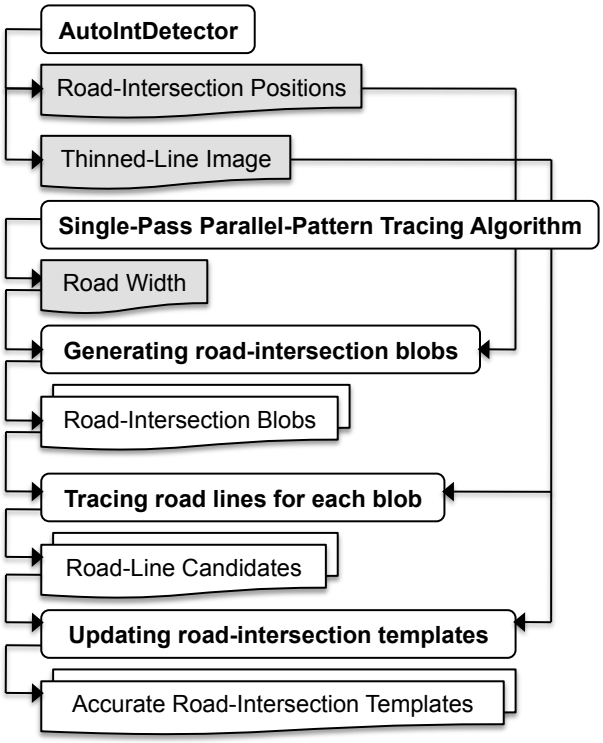


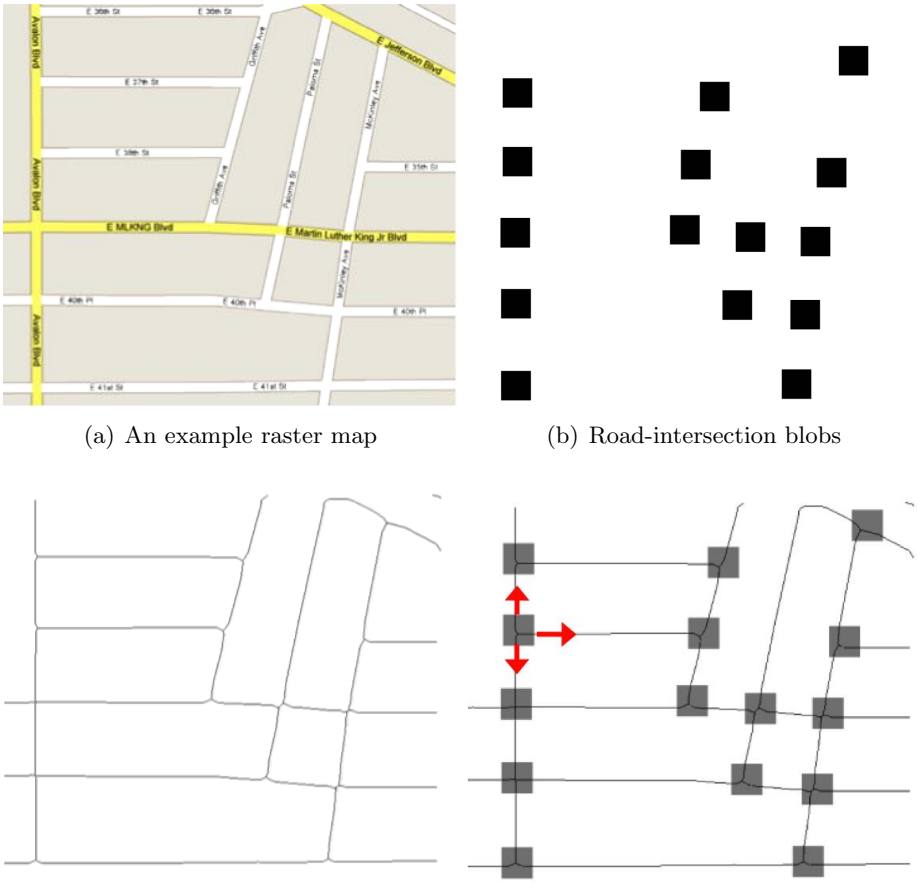
Figure 3.11: The overall approach to extract accurate road-intersection templates from the road layers

with the detected road-intersection points labeled as single foreground pixels. Then, the AutoRoadVectorizer applies the dilation operator to grow a blob for each of the road-intersection points using the road width as the number of iterations. For example, Figure 3.12(a) shows the blob image after the AutoRoadVectorizer applies the dilation operator where the size of each blob is large enough to cover the road area of each road intersection in the original map. Finally, the AutoRoadVectorizer overlaps the blob image with the thinned-line image shown in Figure 3.12(c) to label the extent of the potential distorted lines as show in Figure 3.12(d).

3.2.3.2 Identifying and Tracing Road-Line Candidates

To extract accurate road vector data around the intersections, the AutoRoadVectorizer uses the labeled image shown in Figure 3.12(d) to detect possible road lines intersecting at each road intersection (i.e., road-line candidates) and traces the thinned-line pixels to compute the line orientations. The AutoRoadVectorizer first identifies the contact points between each blob and the thinned-lines by detecting the thinned-line pixels that have any neighboring pixel labeled by the gray boxes. These contact points indicate the starting points of a road-line candidate associated with the blobs. In the example shown in Figure 3.12(d), the road intersection in the second top-left blob has three road-line candidates starting from the contact points that are on the top, right, and bottom of the blob.

For each road-intersection blob, the AutoRoadVectorizer utilizes the Limited Flood-Fill algorithm to trace the thinned-lines from their contact points. Figure 3.13 shows the pseudo-code for the Limited Flood-Fill algorithm. The Limited Flood-Fill algorithm



(a) An example raster map

(b) Road-intersection blobs

(c) Thinned road lines

(d) Labeling distortion areas

Figure 3.12: Generating a blob image to label the distorted lines

first labels a contact point as visited and then checks the eight neighboring pixels of the contact point to find unvisited thinned-line pixels. If one of the eight neighboring pixels is not labeled as visited nor labeled as a potential distorted line pixel, the neighboring pixel is set as the next visit point for the Limited Flood-Fill algorithm to walk through.

```
// The maximum number of line pixels the algorithm is allowed to trace for one line
MaxLinePixel;
// The counter for tracking the number of visited pixels
pixel_count;

Function void limitedFloodFill8(int x, int y)
  if (InsideImage(x,y) && NotVisited(x,y) && pixel_count < MaxLinePixel) {
    pixel_count  = pixel_count + 1;   SetVisited(x,y);
    limitedFloodFill8 (x + 1, y); limitedFloodFill8(x - 1, y - 1); limitedFloodFill8 (x, y + 1);
    limitedFloodFill8(x + 1, y - 1); limitedFloodFill8 (x + 1, y + 1); limitedFloodFill8(x – 1, y);
    limitedFloodFill8 (x - 1, y + 1); limitedFloodFill8(x, y - 1);
    }
  }
```

Figure 3.13: The pseudo-code for the Limited Flood-Fill algorithm

When the Limited Flood-Fill algorithm walks through a new pixel, it records the position of the pixel for later computing the road orientation. The AutoRoadVectorizer limits the number of pixels that the Limited Flood-Fill algorithm can trace from each contact point using a parameter called *MaxLinePixel*. The Limited Flood-Fill algorithm counts the number of pixels that it has visited and stops when the counter is larger than the *MaxLinePixel* variable (the AutoRoadVectorizer uses five pixels in my method). As shown in Figure 3.14, instead of tracing the whole curve starting from the two contact points (i.e., the one on the right and the one on the bottom), the AutoRoadVectorizer utilizes the *MaxLinePixel* to ensure that the Limited Flood-Fill algorithm traces only a small portion of the thinned-lines near the contact points assuming that roads near the

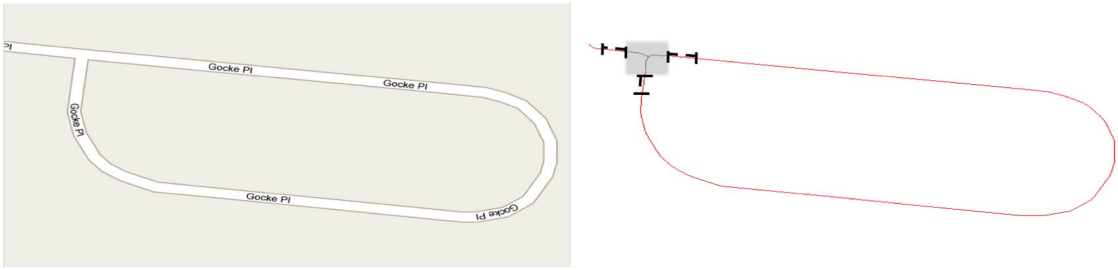


Figure 3.14: Tracing only a small portion of the road lines near the contact points

contact points are straight within a small distance (i.e., several meters within a street block in the raster map).

After the Limited Flood-Fill algorithm walks through the lines from each contact point and records the positions of the line pixels, the AutoRoadVectorizer utilizes the *Least-Squares Fitting* algorithm to find the linear functions of the lines. Assuming a linear function L for a set of line pixels traced by the Limited Flood-Fill algorithm, by minimizing the sum of the squares of the vertical offsets between the line pixels and the line L, the *Least-Squares Fitting* algorithm finds the straight line L that most represents the traced line pixels. The algorithm works as follows:[1]

The target line function for L:

$$Y = m \times X + b \tag{3.3}$$

where

$$m = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2} \tag{3.4}$$

and

$$b = \frac{\sum y - m\sum x}{n} \tag{3.5}$$

[1]The proof of the Least-Squares Fitting algorithm can be found in most statistics textbooks or on the web at http://mathworld.wolfram.com/LeastSquaresFitting.html.

The AutoRoadVectorizer then uses the computed line functions in the next step of updating road-intersection templates to identify actual intersecting road lines and refine the positions of the road intersections.

3.2.3.3 Updating Road-Intersection Templates

There are three possible intersecting cases for the road-line candidates of one intersection as shown in Figure 3.15, where the left images for the three cases are the original maps, the middle images are the thinned lines with the locations of the potential distorted lines labeled by the blob images, and the right images are the traced line functions (i.e., the line functions computed using the *Least-Squares Fitting* algorithm) drawn on a Cartesian coordinate plane.

The top row of Figure 3.15 shows the first case where all the road-line candidates intersect at one point. The middle row of Figure 3.15 shows the second case where the road-line candidates intersect at multiple points and the intersecting points are within a distance threshold to the initially detected road-intersection position. The bottom row of Figure 3.15 shows the third case where the road-line candidates intersect at multiple points and some of the intersecting points are not near the initially detected road-intersection position.

For the first case, the AutoRoadVectorizer adjusts the position of the road intersection to the intersecting point of the road-line candidates. The AutoRoadVectorizer keeps all road-intersection candidates as the intersecting roads of this road-intersection template. The road orientations of this road template are 0 degrees, 90 degrees, and 270 degrees, respectively.

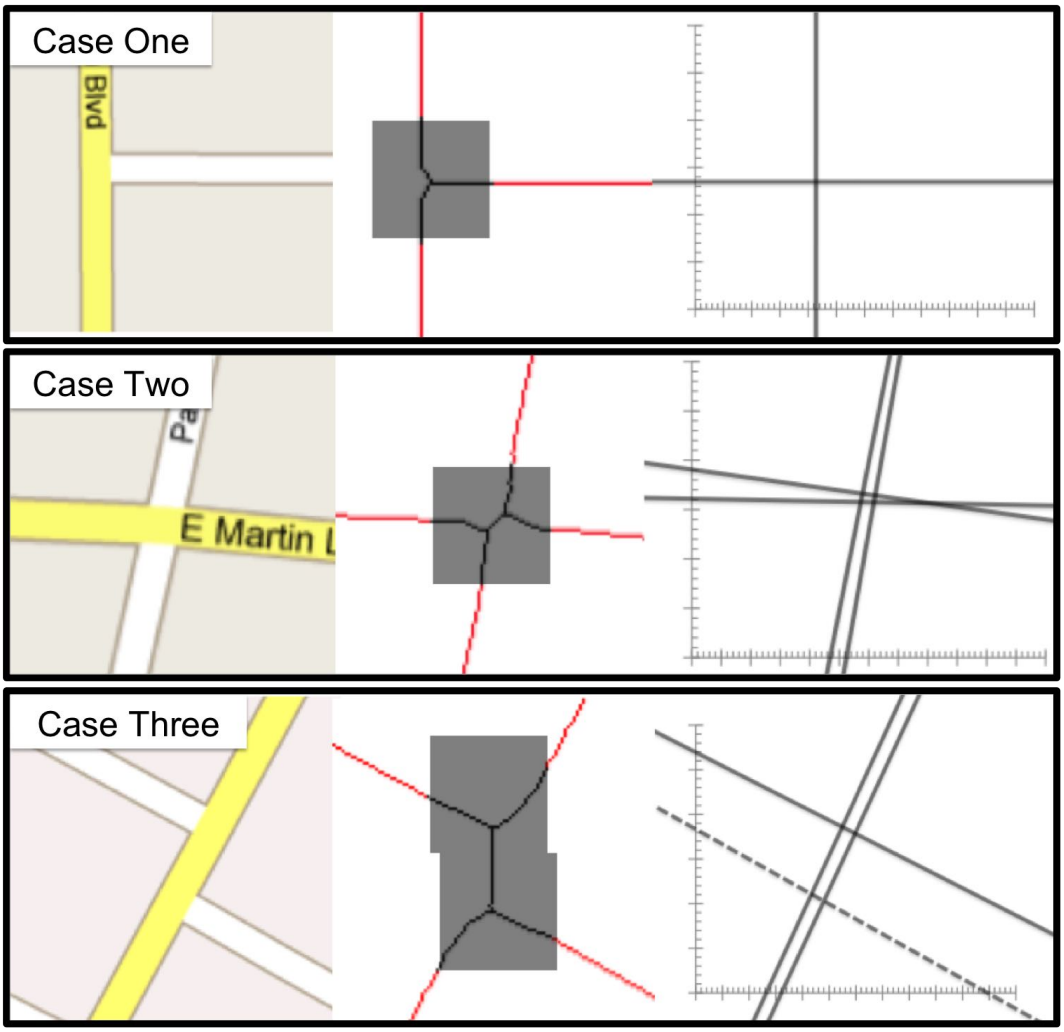
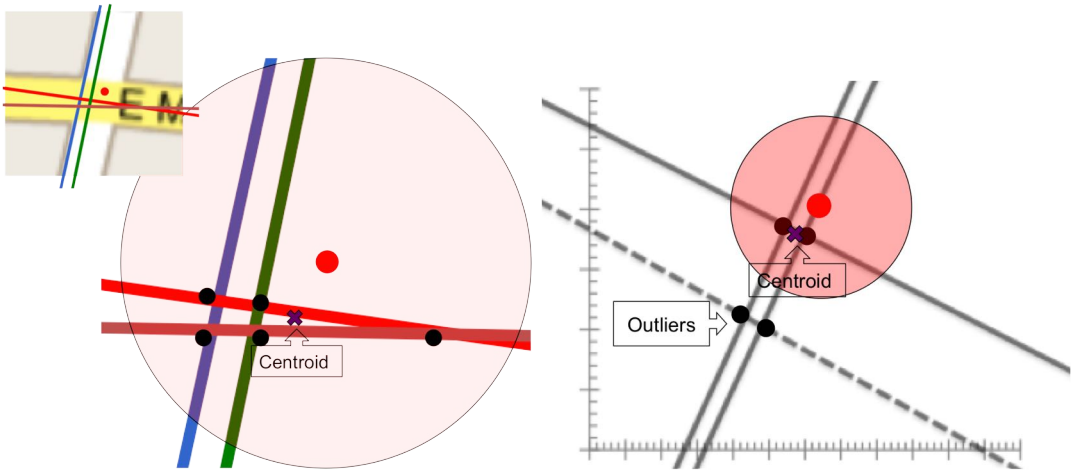


Figure 3.15: The three intersecting cases

For the second case, Figure 3.16(a) shows a detail view where the solid red dot is the initially detected road-intersection position, the green, blue, red, and orange lines are the road-line candidates, the solid black dots are the candidates' intersecting points, and the semi-transparent red circle implies a local area with radius as the detected road width. Since the extent of the distortion depends on the road width, the positional offset between any intersecting point of the road-line candidates and the initially detected road-intersection position should not be larger than the road width. Therefore, for case two, since every intersecting point of the road-line candidates is in the semi-transparent red circle, the AutoRoadVectorizer adjusts the position of the road-intersection template to the centroid of the intersecting points of all road-line candidates. The AutoRoadVectorizer keeps all road-intersection candidates as the intersecting roads of this road-intersection template. The road orientations of this road template are 80 degrees, 172 degrees, 265 degrees, and 355 degrees, respectively.

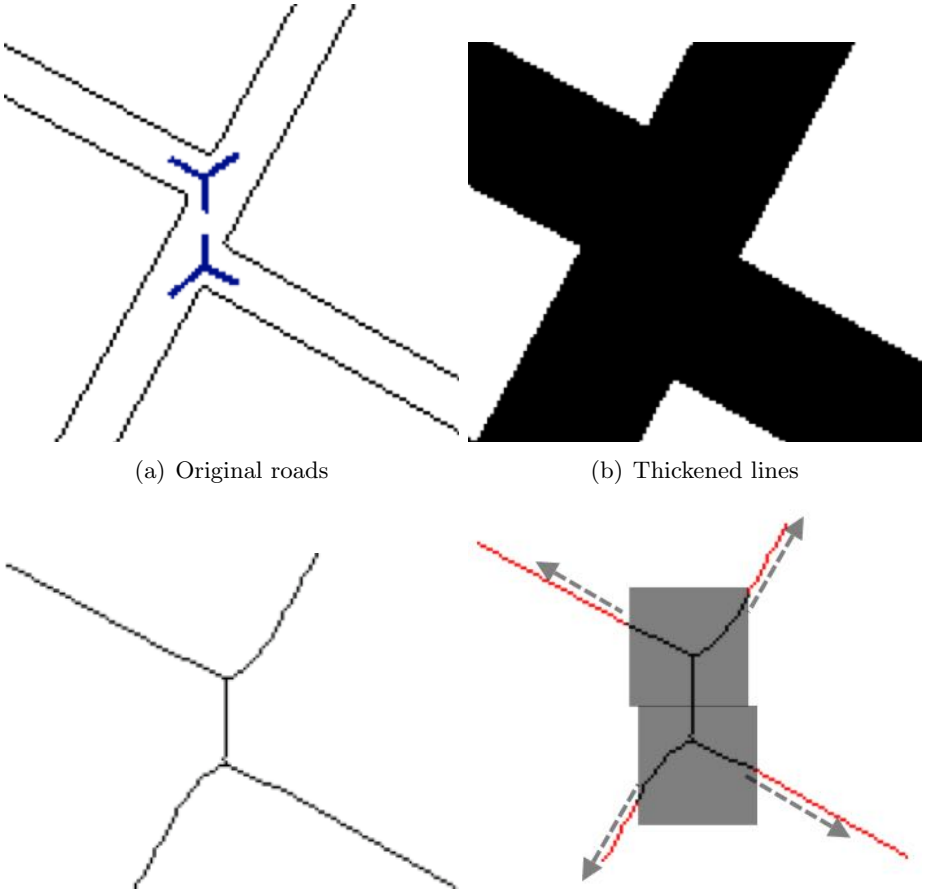


(a) Without outliers (b) Two outliers

Figure 3.16: Adjusting the road-intersection position using the centroid point of the intersections of the road-line candidates

For the third case, when two road intersections are very close to each other, the road geometry between them is totally distorted as shown in Figure 3.17. In this case, the blobs of the two road intersections merge into one big blob as shown in Figure 3.17(d), and the AutoRoadVectorizer associates both road intersections with the four thinned-lines linked to this blob. Figure 3.16(b) shows a detail view of the third case of the merged blobs. Since the two intersecting points where the dashed road-line candidate intersects with two other road-intersection candidates are more than a road width away from the initially detected road-intersection position, the AutoRoadVectorizer discards the dashed road-line candidate. The AutoRoadVectorizer uses the centroid of the remaining two intersecting points as the position of the road-intersection template. Since the AutoRoadVectorizer discards the dashed road-line candidate, the connectivity of this road-intersection template is three and the road orientations are 60 degrees, 150 degrees, and 240 degrees, respectively. This third case shows how the blob image helps to extract correct road orientations even when an intersecting road line is totally distorted by the thinning operator. This case holds when the distorted road line is short and hence likely to have the same orientation as the other intersecting lines. For example, in the third case in Figure 3.15, the distorted line is part of a straight line that goes through the intersection, so it has the same orientation as the 240-degree line.

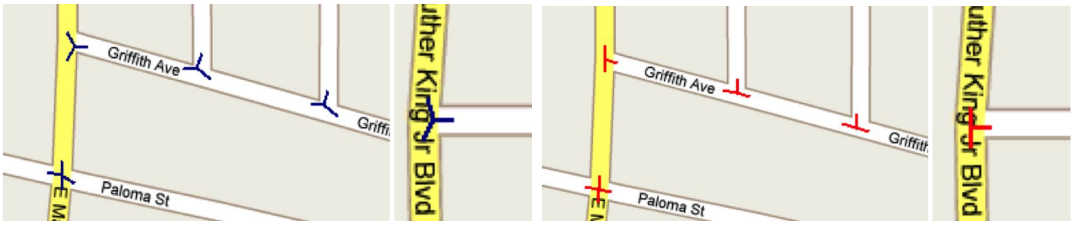
Figure 3.18 shows example results of the accurately extracted road-intersection templates and the results of using the thinning operator only. By utilizing the knowledge of the road width and road format, the AutoRoadVectorizer automatically detects and corrects the distorted lines around road intersections caused by the thinning operator and generates accurate road-intersection templates.



(a) Original roads (b) Thickened lines

(c) Distorted lines (d) Merged blobs

Figure 3.17: Merged nearby blobs



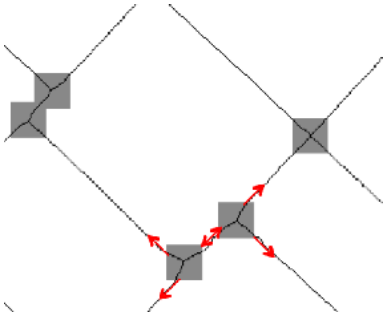
(a) Using the thinning operator only (b) Using the AutoRoadVectorizer

Figure 3.18: Example results of using the thinning operator only and the AutoRoadVectorizer

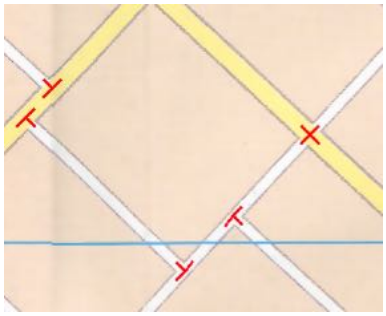
3.2.4 Automatically Vectorizing Road Geometry Using Accurate Road-Intersection Templates

With the accurate positions of the road intersections and the knowledge of potential distorted areas as shown in Figure 3.19(a) to Figure 3.19(b), the AutoRoadVectorizer starts to trace the road pixels in the thinned-line image to generate the road vector data. The thinned-line image contains three types of pixels: the non-distorted road pixels, distorted road pixels, and background pixels. Figure 3.19(a) shows the three types of pixels, which are the black pixels not covered by the gray boxes, black pixels in the gray boxes, and white pixels, respectively. The AutoRoadVectorizer creates a list of *connecting nodes* (CNs) of the road vector data. A CN is a point where two lines meet at different angles. The AutoRoadVectorizer first adds the detected road intersections into the CN list. Then, The AutoRoadVectorizer identifies the CNs among the non-distorted road pixels using a 3x3-pixel-window to check if the pixel has any of the straight-line patterns shown in Figure 3.19(c). The AutoRoadVectorizer adds the pixel to the CN list if the algorithm *does not* detect a straight-line pattern since the road pixel is not on a straight line.

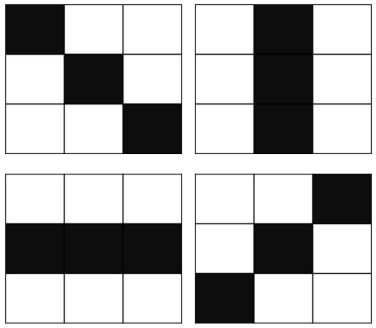
To determine the connectivity between the CNs, the AutoRoadVectorizer traces the road pixels using an eight-connectivity Flood-Fill algorithm shown in Figure 3.20. The Flood-Fill algorithm starts from a CN, travels through the road pixels (both non-distorted and distorted ones), and stops at another CN. Finally, for the CNs that are road intersections, the AutoRoadVectorizer uses the previously updated road intersection positions as the CNs' positions as the main function, shown in Figure 3.20. The CN list and their



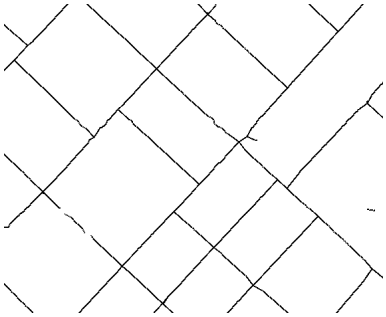
(a) Marking distortions and tracing roads



(b) Accurate road-intersection templates



(c) Straight-line patterns



(d) Extracted road vector data

Figure 3.19: Extracting road vector data from an example map

connectivity are the results of the extracted road vector data. Figure 3.19(d) shows the extracted road vector data. The road vector data around the road intersections are accurate since the AutoRoadVectorizer does not generate any CN using the distorted lines except the road intersections (i.e., does not record the geometry of the distorted lines) and the intersection positions are updated using the accurate road orientations.

```
CNList; // The connecting-node list (CN.x and CN.y are the pixel location)
road_vectors; // The line-segment list (A line segment contains two CN indexes)
// The IDs of the starting and ending CNs of the line segment we  are currently tracing
start_id; end_id;

void main()  // Program starts here
  Foreach CN in the CNList {
    start_id = CN.id;
    SetVisited(CN.x, CN.y);
    floodFill8(CN.x, CN.y);
  }
  // Correct the distortions
  Foreach CN in the CNList {
    if (InsideGrayBox(CN.x, CN.Y) {
     // An intersection
       CN.x =
         GetUpdatedIntersectionLocationX(CN.id);
       CN.y =
         GetUpdatedIntersectionLocationY(CN.id);
  }

Function void floodFill8(int x, int y)
   if (InsideImage(x,y) && NotVisited(x,y)
   && NotBackground(x,y)) {
      if (IsCN(x,y) { // We found a line
          CN end = CNList.FindCNAtLocationXY(x,y);
          road_vectors.AddLine(start_id, end.id);
      } else {
          SetVisited(x,y);
          floodFill8(x + 1, y); floodFill8(x - 1, y - 1);
          floodFill8(x, y + 1); floodFill8(x + 1, y - 1);
          floodFill8(x + 1, y + 1); floodFill8(x – 1, y);
          floodFill8(x - 1, y + 1); floodFill8(x, y - 1);
       }
   }
```

Figure 3.20: The pseudo-code for tracing line pixels

3.2.5 Divide-and-Conquer Extraction of Road Vector Data

The AutoRoadVectorizer utilizes a divide-and-conquer technique (DAC) that divides a raster map into overlapping tiles and processes each tile for extracting its road vector data. Figure 3.21 shows an input scanned map and the DAC divides the desired map region into four overlapping tiles. After the DAC processes all the tiles, the algorithm combines the extracted road vector data from each tile as one set of road vector data for the entire raster map.

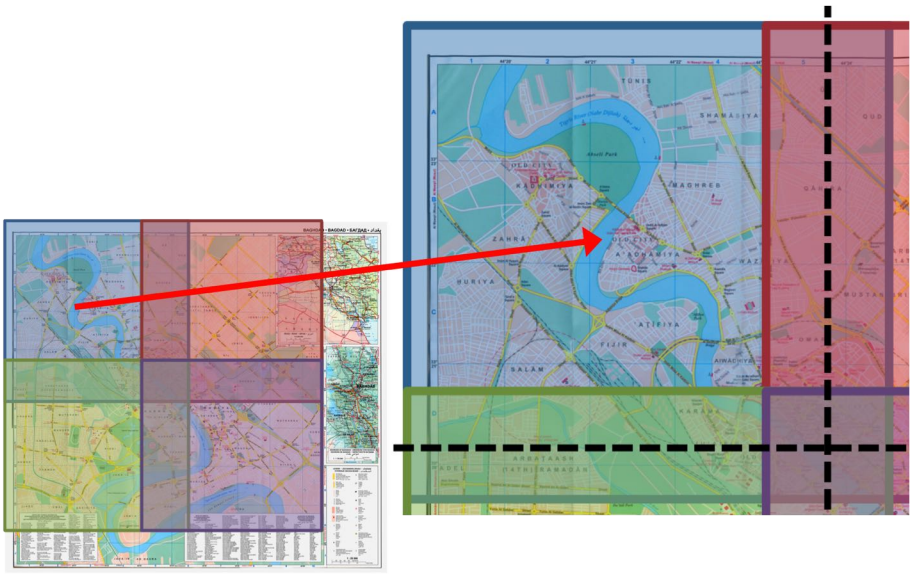
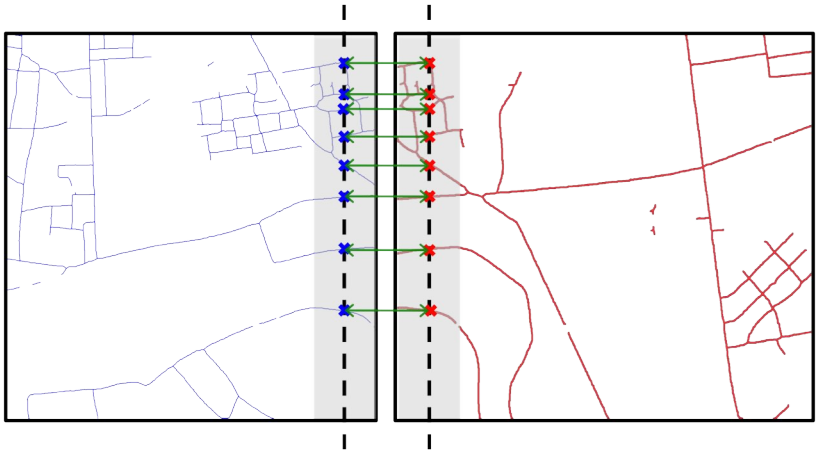


Figure 3.21: Overlapping tiles

For the extracted road vector data of each tile, the DAC cuts the vector data at the center of the overlapping areas as the dashed lines shown in Figure 3.22 and discards the vector data located in the areas between the dashed line and the tile borders. This is because the extracted road vector data near the image borders are usually inaccurate from using the image processing operators (e.g., the morphological operators).

The DAC merges the road vector data from two neighboring tiles by matching the intersections of the dashed lines and the road lines (i.e., the cutting points) of the two neighboring sets of road vector data. For example, Figure 3.22(a) shows two sets road vector data from two neighboring tiles. The DAC first generates a set of cutting points for the left set of road vector data using the intersections of the vertical dashed line and the road lines of the left tile's road vector data. Then, the DAC generates the set of cuttings points for the right set of road vector data. Finally, the DAC merges the two sets of road vector data by connecting two road lines in the two tiles ending at the cutting points of

(a) Cutting the overlapping area (gray)



(b) Connecting the cutting points

(c) The merged road vector data

Figure 3.22: Merging two sets of road vector data from neighboring tiles

the same location as shown in Figure 3.22(b) where each of the horizontal arrows point to a matched pair of cutting points shown as the cross marks. Figure 3.22(c) shows the merged road vector data.

The DAC first merges the road vector data of tiles on the same row from left to right and then it merges the integrated vector data of each row into the final results from top to bottom. The divide-and-conquer approach to process raster maps scales to large images and can take the advantage of multi-core or multi-processor computers to process the tiles in parallel.

3.3 Experiments

I have implemented the supervised road-layer-separation and automatic road-vectorization approaches (the RoadLayerSeparator and RoadVectorizor) described in this chapter as the road vectorization component in a map processing system called Strabo. In this section, I report my experiments on the extraction of road vector data from heterogeneous raster maps using Strabo. I tested Strabo on 16 maps from 11 sources. Table 3.1 shows the information of the test maps and their abbreviations that I use in this section. The set of test maps includes four scanned maps and 12 computer-generated maps.

For the scanned maps, we purchased three paper maps covering the city of Baghdad, Iraq from the International Travel Maps, Gecko Maps, and Gizi Map. I scanned the Bagdad paper maps in 350 DPI to produce the raster maps. I downloaded another scanned map covering the city of Samawah, Iraq from the United Nations Assistance

Map Source (abbr.)	Map Count	Map Type	Image Dimension (pixels)
International Travel Maps (ITM)	1	Scanned	4000x3636
Gecko Maps (GECKO)	1	Scanned	5264x1923
Gizi Map (GIZI)	1	Scanned	3344x3608
UN Iraq (UNIraq)	1	Scanned	4000x3636
Rand McNally (RM)	1	Computer Generated	2084x2756
UN Afghanistan (UNAfg)	1	Computer Generated	3300x2550
Google Maps (Google)	2	Computer Generated	800x550
Live Maps (Live)	2	Computer Generated	800x550
OpenStreetMap (OSM)	2	Computer Generated	800x550
MapQuest Maps (MapQuest)	2	Computer Generated	800x550
Yahoo Maps (Yahoo)	2	Computer Generated	800x550

Table 3.1: Test maps

Mission for Iraq website.[2] The United Nations website does not include the map metadata other than the map scale, which is 15,000:1.

For the computer generated maps, we purchased a map in PDF format covering the city of St. Louis, Missouri from the Rand McNally website.[3] I downloaded a map covering Afghanistan in PDF format from the United Nations Assistance Mission in Afghanistan website.[4] The Afghanistan map shows the main and secondary roads, cities, political boundaries, airports, and railroads of the nation. I cropped ten computer-generated maps from five web-mapping-service providers, Google Maps, Microsoft Live Maps, OpenStreetMap, MapQuest Maps, and Yahoo Maps. The ten computer generated maps cover two areas in the U.S. One area is in Los Angeles, California, and the other one is in St. Louis, Missouri. Figure 3.23 shows examples of the test maps, where the

[2]http://www.uniraq.org

[3]http://www.randmcnally.com/

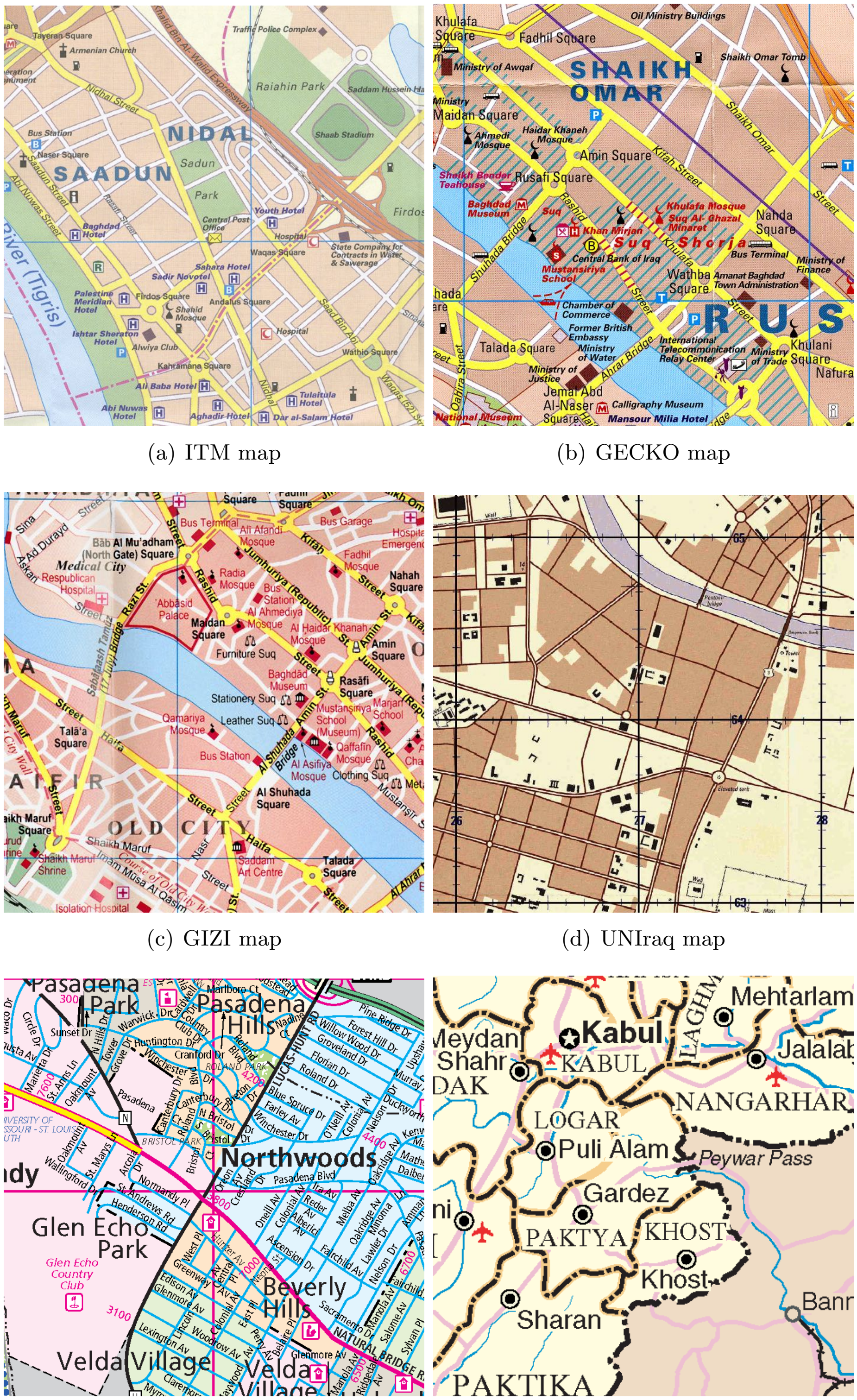
[4]http://unama.unmissions.org/

scanned maps show poor image quality, especially the ones from Gecko Maps and Gizi map with the shadows caused by the fold lines.

3.3.1 Experimental Setup

I utilized Strabo to automatically extract road vector data from the 16 test maps. Strabo successfully generated accurate road vector data for 10 maps among the 16 test maps automatically, which are the maps from the last five sources shown in Table 3.1. The fully automatic road-vectorization function in Strabo did not work well for six of the 16 test maps, which include four scanned maps since the automatic function cannot separate the foreground pixels from the background. The other two maps contain non-roads linear features, which are drawn using the same single-line format as the roads, and hence the automatic function could not separate the road lines from the other map features. Therefore, to achieve the best results, I utilized the supervised road-pixel-extraction function in Strabo to extract the road pixels and then Strabo automatically generated the road geometry and extracted the road vector data.

For comparison, I tested the 16 test maps using the automatic road-vectorization function in R2V from Able Software. R2V allows the user to use one set of color thresholds to extract the road pixels from the map for the automatic road-vectorization function. For raster maps that require more than one set of color thresholds (all of my test map except the UNAfg map require more than one set of color thresholds), the user has to manually specify sample pixels for each of the road color and the sample pixels, which requires significant user effort in R2V. Therefore, for the raster maps that require more than one set of color thresholds to extract their road pixels, I utilized Strabo to first extract the



(a) ITM map

(b) GECKO map

(c) GIZI map

(d) UNIraq map

(e) RM map

(f) UNAfg map

Figure 3.23: Examples of the test maps

road pixels and then utilized the "Auto Vectorize" function in R2V to vectorize the roads without manual pre-processing or post-processing.

3.3.2 Evaluation Methodology

To evaluate my supervised road-pixel-extraction approach, I report the number of user labels that were required for extracting road pixels from a test map using Strabo. For the six maps that I tested for the supervised approach (ITM, GECKO, GIZI, UNIraq, RM, and UNAfg), four maps are scanned maps, which contain numerous colors (ITM, GECKO, GIZI, and UNIraq). Strabo automatically quantized the four scanned maps using the Mean-shift, Median-cut, and K-means algorithms to generate four quantized images in different quantization levels for each map (the four quantization levels have the number of colors as 32, 64, 128, and 256, respectively). Strabo did not apply the color segmentation algorithms on the two computer generated maps (RM and UNAfg) before user labeling. This is because the computer-generated maps contain a smaller number of colors. The UNAfg map has 90 unique colors and there is only one color representing both the major and secondary roads in the map. The RM map has 20 unique colors, with five colors representing roads. The user started using Strabo for the user-labeling task from the quantized image of the highest quantization level (i.e., the quantized image that has the smallest number of colors). If the user could not distinguish the road pixels from other map features (e.g., background) in the quantized image, the user would need to select a quantized image containing more colors (a lower quantization level) for user labeling.

I evaluated the quality of the extracted road-intersection templates using 10 raster maps of the last five sources shown in Table 3.1. I report the quality of the extraction results using the positional offset, orientation offset, and connectivity offset. The positional offset is the average number of pixels between the extracted road intersection templates and the actual road intersections on the raster maps. The actual road intersections in the raster maps are defined as the centroid points of the intersection areas of two or more intersecting road lines. The orientation offset is the average number in degrees between the extract road orientations and the actual road orientations. The connectivity offset is the total number of missed road lines. I manually examine each road intersection in the raster maps to obtain the ground truth of the positions of the road intersections, the connectivity, and the road orientations.

For evaluating the extracted road vector data, I report the accuracy of the extraction results using the road extraction metrics proposed by Heipke et al. [1997], which include the completeness, correctness, quality, redundancy, and the root-mean-square (RMS) difference. I manually drew the centerline of every road line in the maps as the ground truth. The completeness and correctness represent how complete/correct the extracted road vector data are. The completeness is the length of true positives divided by the summation of the lengths of true positives and false negatives, and the optimum is 100%. The correctness is the length of true positives divided by the summation of the lengths of true positives and false positives, and the optimum is 100%. The quality is a combination metric of the completeness and correctness, which is the length of true positives divided by the sum of the lengths of true positives, false positives, and false negatives, and the optimum is 100%. The redundancy shows the percentage of the matched ground truth

that is redundant (i.e., more than one true positive line matched to one ground truth line), and the optimum is 0. The RMS difference is the average distance between the extracted lines and the ground truth, which represents the geometrical accuracy of the extracted road vector data. To generate these metrics, Heipke et al. [1997] suggest using a buffer width as half of the road width in the test data so that a correctly extracted line is inbetween the road edges. In my test maps, the roads range from five to twelve pixels wide. I used a buffer width of three pixels, which means a correctly extracted line is no farther than three pixels to the road centerlines.

3.3.3 Experimental Results

Table 3.2 shows the number of colors in the images used for user labeling and the number of user labels used for extracting the road pixels. For all the scanned maps, only one to nine labels were needed for Strabo to extract the road pixels.

Map Source	Original Map Colors	Quantized Map Colors	User Labels
ITM	779,338	64	9
GECKO	441,767	128	5
GIZI	599,470	64	9
UNIraq	217,790	64	5
RM	20	N/A	5
UMAF	90	N/A	1

Table 3.2: The number of colors in the image for user labeling of each test map and the number of user labels for extracting the road pixels

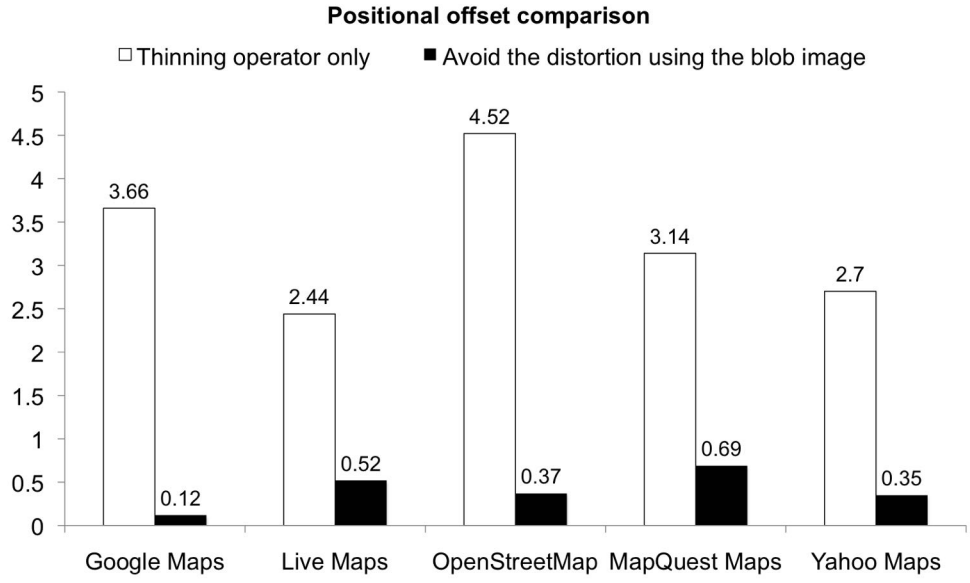
From the 10 raster maps examined for the accuracy of the road-intersection templates, Strabo correctly extracted all the 139 road intersections in the 10 raster maps and extracted 438 of the 451 road lines intersecting at the road intersections. Table 3.3 shows the results. The average positional offset was 0.4 pixels and the average orientation offset

was 0.24 degrees, which shows that the extracted road-intersection templates are very close to the ground truth in these test maps. In order to achieve higher accuracy in the positional and orientation offsets, Strabo needed to discard the lines that are outliers in the step to compute the position of the road-intersection template; Strabo missed 13 lines from a total of 451 lines in the process of tracing the lines to extract the road intersection templates. These 13 lines all belong to the road intersections that are near the boundaries of the map and hence Strabo could not find road lines long enough to compute the correct orientations. I also compare the extracted templates using Strabo with the extracted template using the one-pixel road lines directly generated by the thinning operator. The comparison results of the positional offset and orientation offset are shown in Figure 3.24. My approach in this chapter had a significant improvement on the results of every map source for both of the positional and orientation offsets.

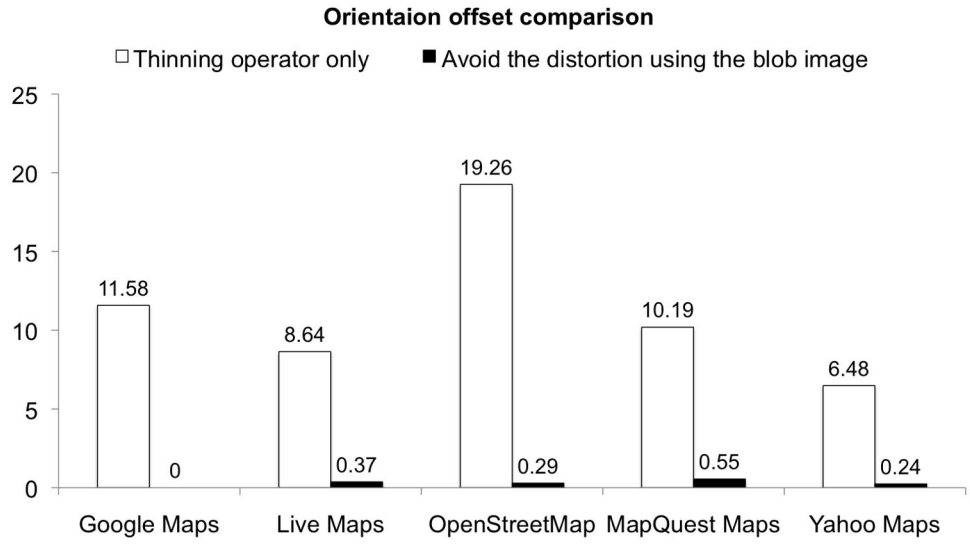
Map Source	**Total Ints.**	**Extracted Lines**	**Missed Lines**	**Avg. Position Offset (pixels)**	**Avg. Orientation Offset (degrees)**
Google	28	87	3	0.12	0
Live	28	91	2	0.52	0.37
OSM	31	95	3	0.37	0.29
MapQuest	25	83	0	0.69	0.55
Yahoo	27	82	5	0.35	0.24

Table 3.3: The average positional offset, average orientation offset, and connectivity offset (the number of missed lines for each map source)

Table 3.4 shows the numeric results from using Strabo and R2V to extract road vector data from the 16 test maps. The average completeness, correctness, quality, redundancy, and RMS differences of Strabo were 96.53%, 97.61%, 94.41%, 0.19%, and 2.79 pixels compared to R2V's 94.9%, 87.4%, 79.73%, 42.81%, and 16.12 pixels. Strabo produced more accurate road vector data with small RMS differences. I emphasize the numbers



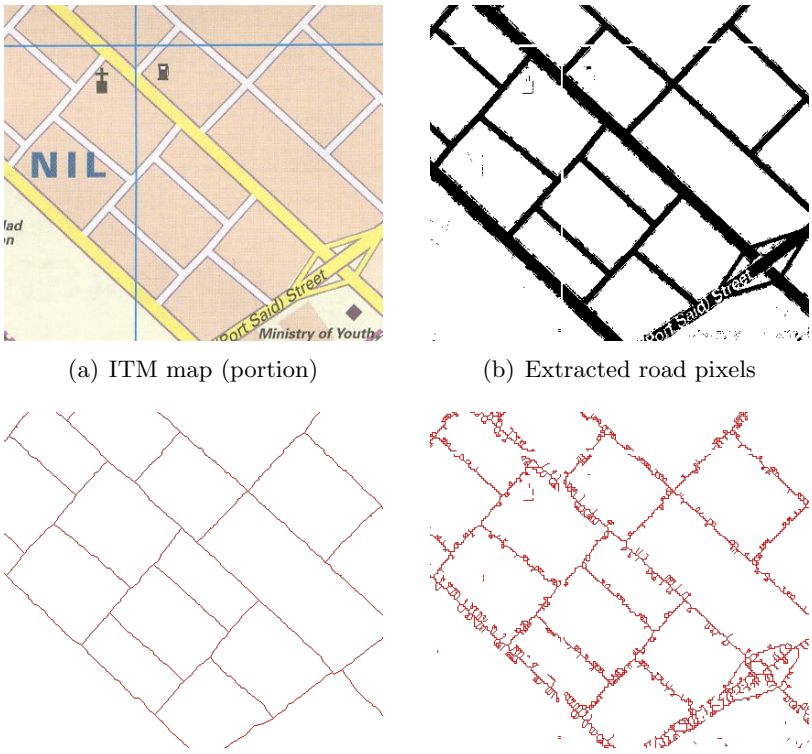
(a) Positional offset comparison (in pixels)



(b) Orientation offset comparison (in degrees)

Figure 3.24: Quality comparison

where R2V generated a better result than Strabo. R2V produced better completeness numbers for five test maps, which is because R2V generated highly redundant lines, while Strabo eliminated small branches, such as the highway ramps shown in Figure 3.25. R2V could achieve better results if I tuned R2V with manually specified pre-processing and post-processing functions (e.g., manually specify the gap size for reconnecting two lines).



(a) ITM map (portion) (b) Extracted road pixels

(c) Strabo results (d) R2V results

Figure 3.25: Example results using Strabo and R2V of a cropped area from the ITM map

Figure 3.26 to Figure 3.30 show some example results where the geometry of the extracted road vector data are very close to the road centerlines for both straight and curved roads, especially the computer generated maps from the web-mapping-service providers. The ITM, GECKO, GIZI, UNIraq, and RM maps had lower than average

Map Source	Comp.	Corr.	Quality	Redundancy	RMS
ITM (Strabo)	90.02%	93.95%	85.08%	0.85%	3.59
ITM (R2V)	**96.00%**	66.91%	65.09%	117.33%	13.68
GECKO (Strabo)	93.75%	94.75%	89.12%	0.61%	2.95
GECKO (R2V)	**96.52%**	76.44%	74.39%	52.64%	8.27
GIZI (Strabo)	92.90%	96.18%	89.59%	0.00%	2.46
GIZI (R2V)	**93.43%**	95.03%	89.08%	39.42%	11.17
UNIraq (Strabo)	88.31%	96.01%	85.19%	0.00%	6.94
UNIraq (R2V)	**94.92%**	78.38%	75.22%	18.82%	**5.19**
RM (Strabo)	96.03%	84.72%	81.85%	1.60%	2.79
RM (R2V)	92.74%	68.89%	65.36%	33.56%	16.03
UNAfg (Strabo)	86.02%	99.92%	85.96%	0.00%	3.68
UNAfg (R2V)	**88.26%**	99.92%	**88.20%**	12.36%	3.98
Google 1 (Strabo)	99.64%	99.74%	99.38%	0.00%	0.85
Google 1 (R2V)	85.67%	80.16%	70.68%	17.80%	19.78
Google 2 (Strabo)	99.60%	100.00%	99.60%	0.00%	0.77
Google 2 (R2V)	81.22%	83.70%	70.13%	19.77%	18.54
LM 1 (Strabo)	99.41%	96.61%	96.06%	0.00%	15.14
LM 1 (R2V)	79.58%	66.67%	56.93%	25.38%	29.39
LM 2 (Strabo)	99.52%	100.00%	99.52%	0.00%	1.02
LM 2 (R2V)	87.26%	76.05%	68.45%	33.12%	18.31
OSM 1 (Strabo)	100.00%	100.00%	100.00%	0.00%	0.82
OSM 1 (R2V)	85.26%	87.63%	76.11%	7.33%	12.23
OSM 2 (Strabo)	99.61%	100.00%	99.61%	0.00%	0.71
OSM 2 (R2V)	95.67%	99.78%	95.47%	6.30%	9.46
MapQuest 1 (Strabo)	100.00%	100.00%	100.00%	0.00%	0.73
MapQuest 1 (R2V)	84.58%	86.81%	74.95%	6.80%	12.27
MapQuest 2 (Strabo)	99.69%	100.00%	99.69%	0.00%	0.76
MapQuest 2 (R2V)	99.43%	100.00%	99.43%	8.25%	1.17
Yahoo 1 (Strabo)	100.00%	99.94%	99.94%	0.00%	0.69
Yahoo 1 (R2V)	85.71%	77.85%	68.91%	79.55%	25.48
Yahoo 2 (Strabo)	99.94%	100.00%	99.94%	0.00%	0.80
Yahoo 2 (R2V)	86.49%	76.49%	68.33%	127.03%	27.51
Avg. (Strabo)	96.53%	97.61%	94.41%	0.19%	2.79
Avg. (R2V)	94.90%	87.41%	79.73%	42.81%	16.12

Table 3.4: Numeric results of the extracted road vector data (three-pixel-wide buffer) using Strabo and R2V

correctness numbers since some of the non-road features were also extracted using the identified road colors and those parts contributed to false positive road vector data. Figure 3.26(a) to Figure 3.26(c) show a portion of the ITM map where the runways are represented using the same color as the white roads and hence were extracted as road pixels. Figure 3.28(a) to Figure 3.28(c) show a portion of the UNIraq map where part of the building pixels were extracted since they share the same colors of the road shadows. Figure 3.29(a) shows two grid lines in pink on the left and right portions of the RM map were also extracted since they have the same color as the major road shown at the center of the map. In addition, Figure 3.29(b) shows the extracted road pixels using the supervised function of Strabo and many characters were extracted since they share the same color as the black lines. Although I removed the majority of the characters automatically, some of the characters were miss-identified as road lines since they touch the road lines and the connected-component analysis approach I used could not remove this type of false positive. Including a user validation step after the road pixels were extracted could further reduce this type of false positives resulting in lower correctness numbers.

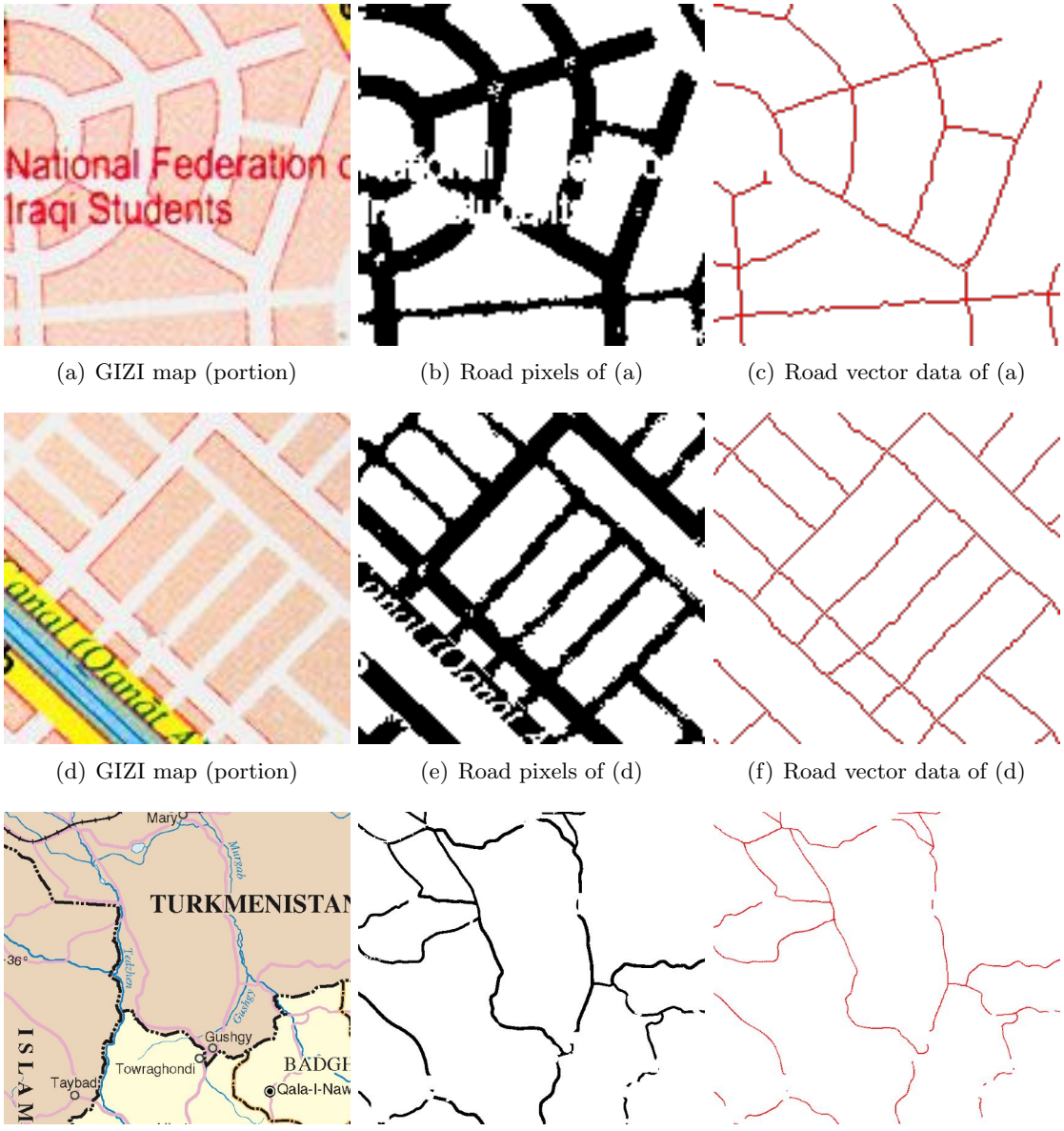
For the lower than the average completeness numbers in the ITM, GECKO, GIZI, and UNAfg maps, some broken lines were not reconnected since the gaps were larger than the iterations of the dilation operator after the non-road overlapping features were removed, such as the gaps in the GECKO and GIZI maps shown in Figure 3.26 and Figure 3.27. The broken lines could be reconnected with post-processing on the road vector data since the gaps are now smaller than they were in the extracted road layers resulting from the dilation operator. For the lower than the average completeness numbers in the scanned

(a) ITM map (portion) (b) Road pixels of (a) (c) Road vector data of (a)

(d) GECKO map (portion) (e) Road pixels of (d) (f) Road vector data of (d)

(g) GECKO map (portion) (h) Road pixels of (g) (i) Road vector data of (g)

Figure 3.26: Examples of the road vectorization results of the ITM and GECKO maps

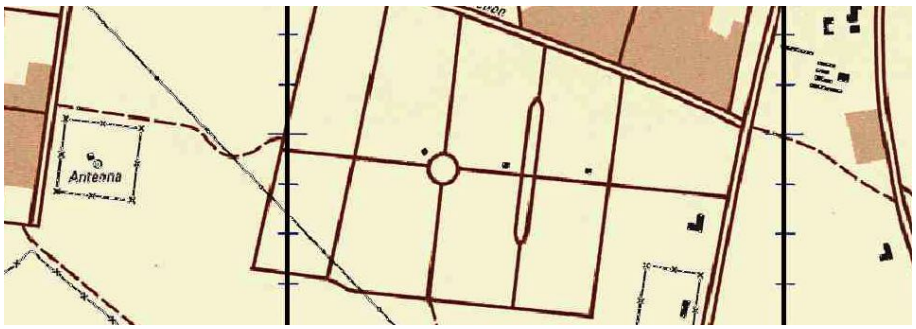


(a) GIZI map (portion) (b) Road pixels of (a) (c) Road vector data of (a)

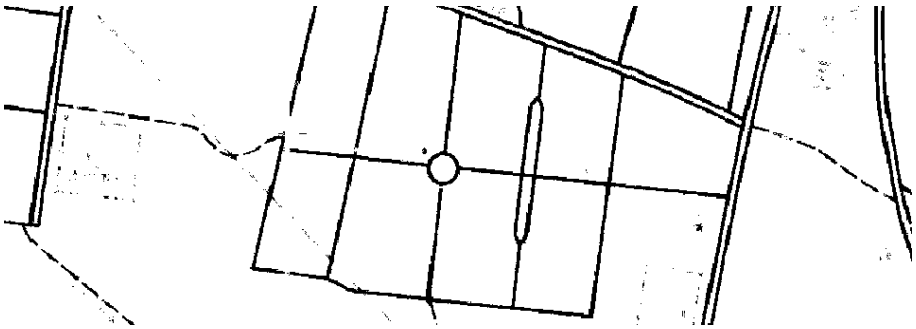
(d) GIZI map (portion) (e) Road pixels of (d) (f) Road vector data of (d)

(g) UNAfg map (portion) (h) Road pixels of (g) (i) Road vector data of (g)

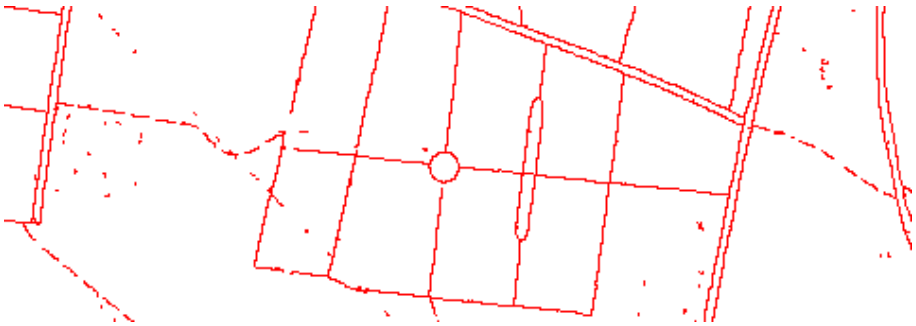
Figure 3.27: Examples of the road vectorization results of the GIZI and UNAfg maps



(a) UNIraq map (portion)

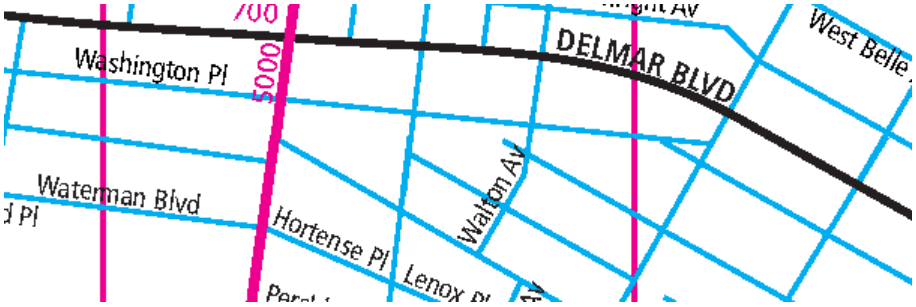


(b) Road pixels of (a)

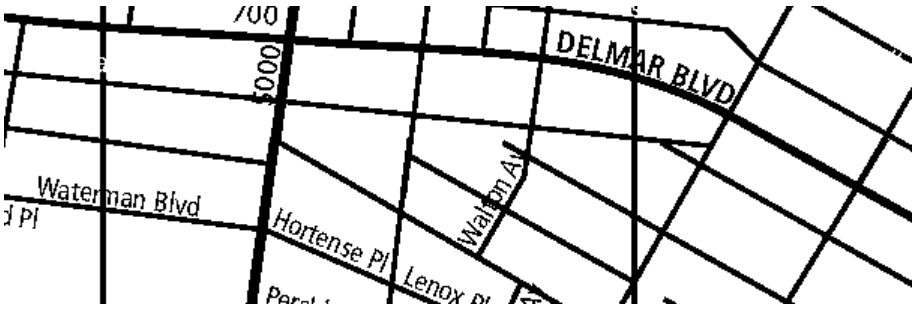


(c) Road vector data of (a)

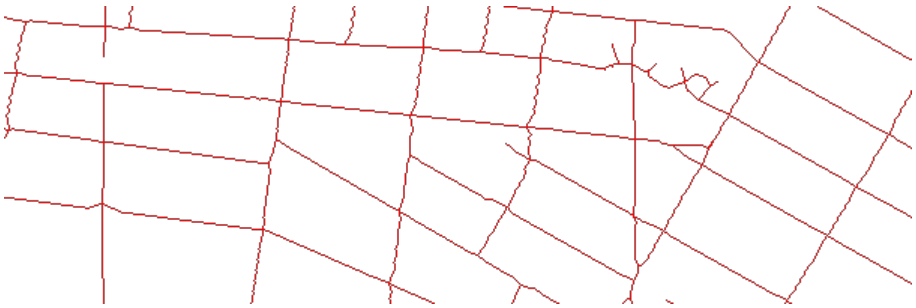
Figure 3.28: Examples of the road vectorization results of the UNIraq map



(a) RM map (portion)

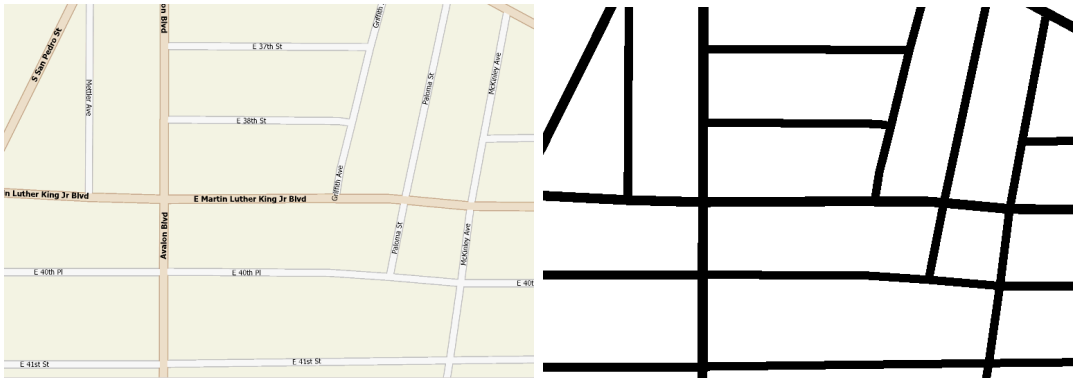


(b) Road pixels of (a)



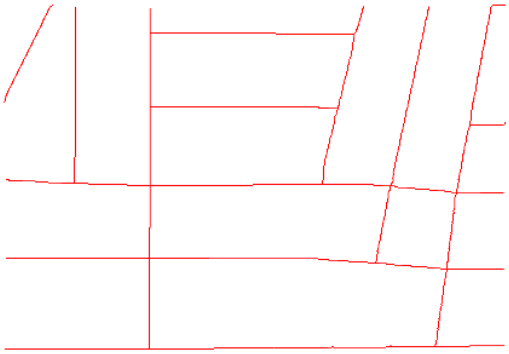
(c) Road vector data of (a)

Figure 3.29: Examples of the road vectorization results of the RM map

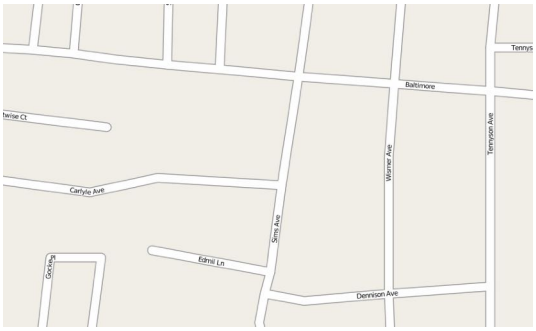


(a) MapQuest map

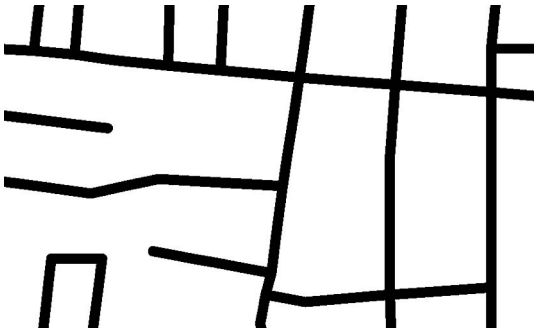
(b) Road pixels of (a)



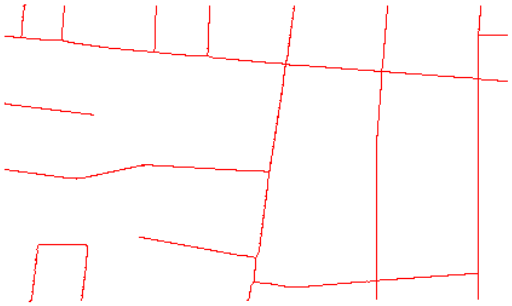
(c) Road vector data of (a)



(d) OSM map



(e) Road pixels of (d)



(f) Road vector data of (d)

Figure 3.30: Examples of the road vectorization results from maps of the MapQuest and OSM maps

UNIraq map, some of the road lines as shown in Figure 3.28(a) are dashed lines and the ground truth were drawn as solid lines.

Strabo's redundancy numbers are generally low since I correctly identify the center-lines for extracting the road vector data. The average RMS differences are under three pixels, which shows that the thinning operator and my approach to correct the distortion result in good quality road geometry. For example, in the results shown in Figure 3.30(a) to Figure 3.30(f), although the extracted road lines are thick, Strabo extracted accurate road vector data around the intersections. The high redundancy numbers of R2V resulted from no manual pre-processing before R2V's automatic function to extract the centerlines of the roads, and the automatic function is sensitive to wide road lines.

3.3.4 Computation Time

I built Strabo using Microsoft Visual Studio 2008 running on a Microsoft Windows 2003 Server powered by a 3.2 GHz Intel Pentium 4 CPU with 4GB RAM. The average processing time for the entire process on vectorizing the road pixels for an 800x550-pixels map was 5 seconds, for a 2084x2756-pixels map was 2 minutes, and for a 4000x3636-pixels map was 3.5 minutes. The dominant factors of the computation time are the image size, the number of road pixels in the raster map, and the number of road intersections in the road layer. The implementation was not fully optimized and improvements could still be made to speed the processes, such as multi-threading on processing map tiles of an input map.

3.4 Conclusion

This chapter presents the RoadLayerSeparator and AutoRoadVectorizer techniques. The RoadLayerSeparator separates road layers from raster maps with poor image quality or complex maps, and the AutoRoadVectorizer extracts accurate road vector data from the separated road layer. My experiments show that together with the AutoMapDecomposer and AutoIntDetector described in Chapter 2, the techniques separated road layers from raster maps of 11 sources with varying color usages and image quality using minimal user input and extracted accurate road vector data from the separated road layers.

Chapter 4

Automatic Recognition of Text Labels

Text labels in raster maps provide valuable geospatial information by associating geographical names with geospatial locations. By converting the text labels in a raster map to machine-editable text, we can produce geospatial knowledge, such as generating a gazetteer for the map coverage area using the recognized text, for understanding the map region when other geospatial data are not ready available. Moreover, we can register a raster map to other geospatial data (e.g., imagery) [Chen et al., 2008; Wu et al., 2007] and exploit the recognized text from the map for indexing and retrieval of the other geospatial data.

The technique of converting the printed text in document images to machine-editable text is called optical character recognition (OCR), which is an active area in both academia research [Mori et al., 1995] and software development [Nagy et al., 2000], such as the commercial products like the ABBYY FineReader,[1] NUANCE OminiPage,[2] and open source

[1] http://www.abbyy.com/

[2] http://www.nuance.com/

software, such as the OCRopus,[3] Tesseract OCR,[4] etc. In general, the first step of an OCR system is "zoning," which analyzes the layout of an input document (e.g., single column and tables) for locating and ordering the text regions (i.e., zones), each containing text lines of the same orientation (usually horizontal or vertical text) [Kanai et al., 1995; Mao et al., 2003]. Then, the OCR system processes each identified zones for recognizing the text.

Current OCR techniques produce a high recognition rate on documents with simple layout (i.e., documents that can be successfully separated into zones). From 1992 to 1996, the Information Science Research Institute (ISRI) of the University of Nevada, Las Vegas conducted five annual tests of OCR accuracy using various state-of-art systems with a set of test images from government reports, legal document samples, magazines, and newspapers. In ISRI's last publication of the annual test of OCR accuracy [Rice et al., 1996], the test data sets contained more than 2,000 pages with nearly five million characters in English, Spanish, and German and the text strings in the test images were mostly of the same orientation. The character accuracy of an OCR system in ISRI's annual tests is defined as follows: N is the number of correctly recognized characters and L is the Levenshtein distance [Levenshtein, 1966] between the recognition results and the ground truth, the character accuracy is given by:

$$CharacterAccuracy = \frac{N - L}{N} \tag{4.1}$$

[3] http://code.google.com/p/ocropus/

[4] http://sourceforge.net/projects/tesseract-ocr/

Rice et al. [1996] report that five of the six tested OCR system achieved over 93% character accuracy on all types of test documents. For the legal document samples in English, which had a simpler layout compared to the magazine pages, the six OCR systems achieved even higher character accuracy numbers from 97.82% to 99.66%. ISRI's experiments also showed that increasing the scanning resolution from 200 to 300 dot-per-inch (DPI) had a substantial improvement on the recognition results.

Although present OCR systems can achieve a high recognition rate on documents with simple layout, text recognition from raster maps is still a challenging task. First, the image quality of the raster maps usually suffers from the scanning and/or image compression processes. Second, the text labels in a raster map can have various font types and sizes and very often overlap with each other or with other features in the map, such as roads. Third, the text labels within a map do not follow a fixed orientation and do not have a specific reading order. Therefore, the document structure analysis techniques for zoning generally do not work on maps [Mao et al., 2003].

This chapter presents a general approach for recognizing text labels in raster maps. . I first present my supervised technique called the Text Layer Separator (TextLayerSeparator) for separating text layers from complex raster maps or the maps with poor image quality, which requires only minimal user input. Then, I describe my automatic technique called the Automatic Text Recognizer (AutoTextRecognizer) that focuses on locating individual text labels in the map and detecting their orientations to then leverage the horizontal text recognition capability of commercial OCR software. Together with the automatic map decomposition technique (AutoMapDecomposer) described in Chapter 2, I can process raster maps with varying map complexity and image quality for separating

their text layers with minimal user input and automatically recognize the text labels in the separated road layers.

Figure 4.1 shows the overall approach of the TextLayerSeparator and AutoTextRecognizer. To overcome the difficulties of processing raster maps with poor image quality and overlapping features, the TextLayerSeparator utilizes a supervised technique that analyzes user labels for extracting the pixels that represent text (i.e., the text layer) from raster maps. With the separated text layer, the AutoTextRecognizer employs three distinct steps for recognizing the text labels. The first step in the algorithm automatically groups characters for identifying text strings by exploiting the fact that the characters in a text string have a similar size and are spatially near each other. For the identified text strings, the second step is to automatically detect their orientations and rotate the strings to the horizontal. Finally, the last step utilizes a conventional OCR product to recognize the characters in the horizontal strings and then generates the recognized text. I describe the details of the TextLayerSeparator and AutoTextRecognizer techniques in the following sections.

4.1 Supervised Extraction of Text Layers

There are three major steps in the TextLayerSeparator for the supervised extraction of text layers from raster maps. The first step is to quantize the color space of the input image. Then, the user provides examples of text areas of the quantized image. The user can also provide labels of non-text areas to help the TextLayerSeparator select text colors. Finally, the last step is to automatically identify a set of text colors from the user

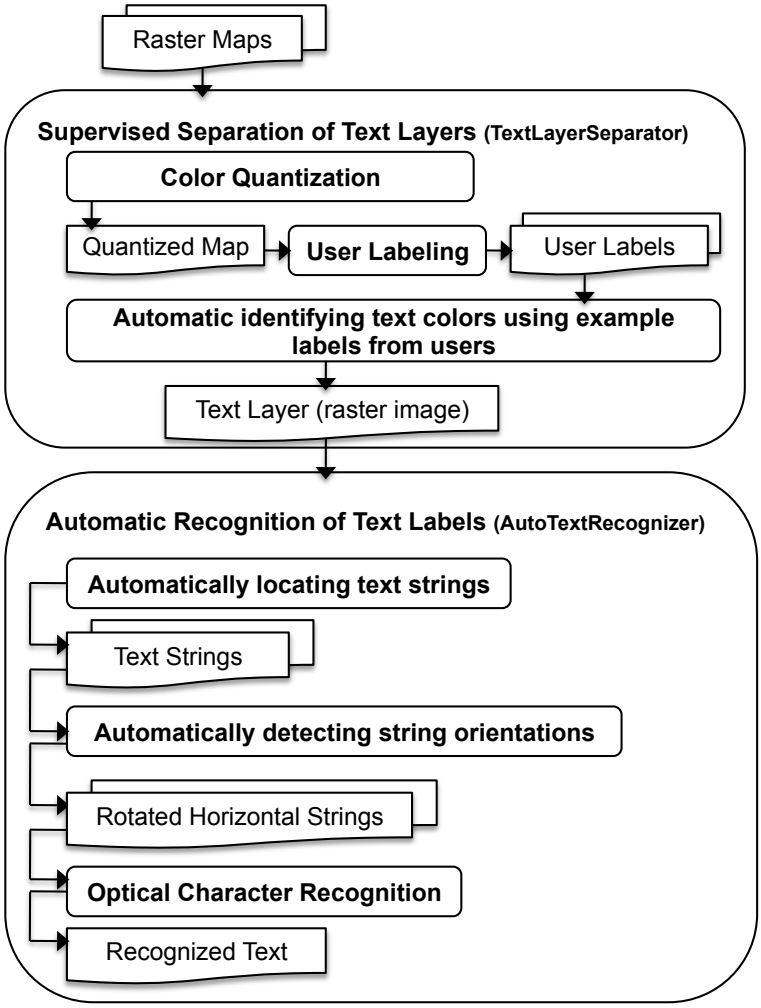


Figure 4.1: The overall approach of the TextLayerSeparator and AutoTextRecognizer

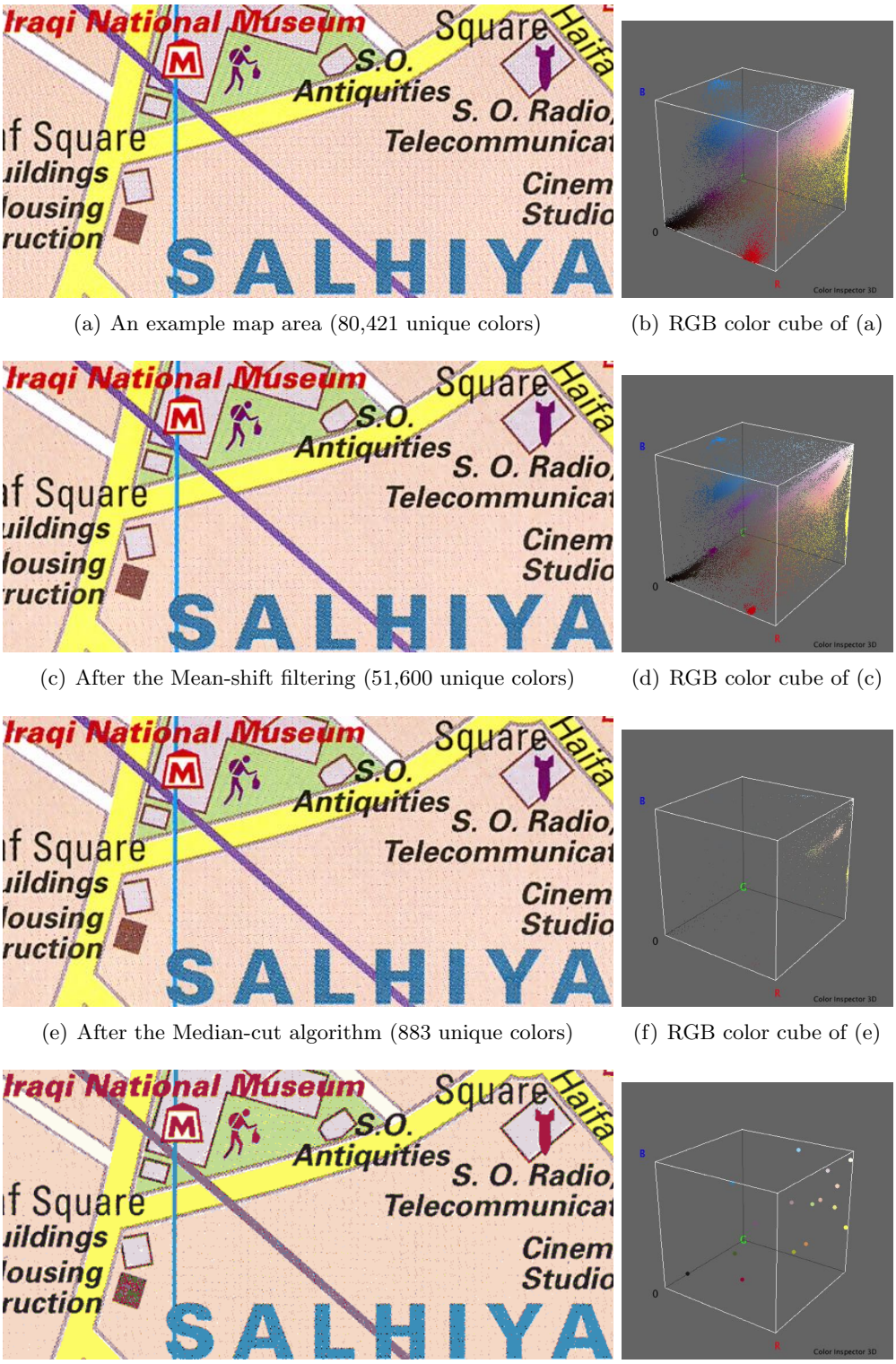
examples and generate a color filter to extract the text layers from the raster map. The user interaction is designed to be consistent with the supervised road layer extraction technique (RoadLayerSeparator) described in Chapter 3.

4.1.1 Color Quantization

Raster maps usually contain numerous colors due to the scanning or compression processes. To extract the text pixels, the TextLayerSeparator applies color segmentation techniques to reduce the number of colors in the maps for generating a color palette with a limited number of colors. Similar to the the RoadLayerSeparator in Chapter 3, the TextLayerSeparator first utilizes the Mean-shift filtering algorithm [Comaniciu and Meer, 2002] to smooth the image and reduce noise. Next, the TextLayerSeparator utilizes the Median-cut [Heckbert, 1982] and K-means to generate a set of quantized images for the user to label text areas. Figure 4.2 shows an example scanned map, the quantized image after the Mean-shift filtering, Median-cut, and K-means algorithms. The Mean-shift filtering algorithm reduces the number of unique colors in the raster map by 37% and then the Median-cut algorithm further produces a quantized map with 883 unique colors as shown in Figure 4.2(d) and Figure 4.2(f). Finally, the K-means algorithm generates quantized maps in different quantization levels using various numbers of K. Figure 4.2(h) shows a quantized map with K equal to 16.

4.1.2 User Labeling

In this user-labeling step, the user selects a quantized map that contains text strings in colors distinct from other features from a set of quantized maps in different quantization



(a) An example map area (80,421 unique colors) (b) RGB color cube of (a)

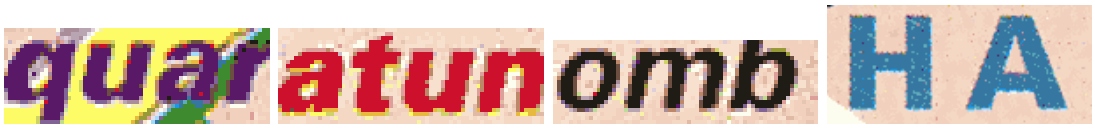
(c) After the Mean-shift filtering (51,600 unique colors) (d) RGB color cube of (c)

(e) After the Median-cut algorithm (883 unique colors) (f) RGB color cube of (e)

(g) After the K-means algorithm with K=16 (16 unique colors) (h) RGB color cube of (g)

Figure 4.2: An example map tile and the color quantization results with their red, green, and blue (RGB) color cubes

levels. Then, the user provides an example label for each text color in the quantized map (i.e., a text label). The text label is a rectangle that should be large enough to cover a string of at lease two characters as the examples shown in Figure 4.3. The user can also provide non-text labels to help the TextLayerSeparator identify text colors. A non-text label is a rectangle that covers *only* non-text colors in the raster map. To generate the text and non-text labels, the user first clicks on the map and then selects the size of the label and rotates the selection area to label non-horizontal text if needed. Figure 4.4(a) shows the user interface and an example text label. Figure 4.4(b) shows that the text label is automatically rotated to the horizontal direction if the user selects a non-horizontal text string.



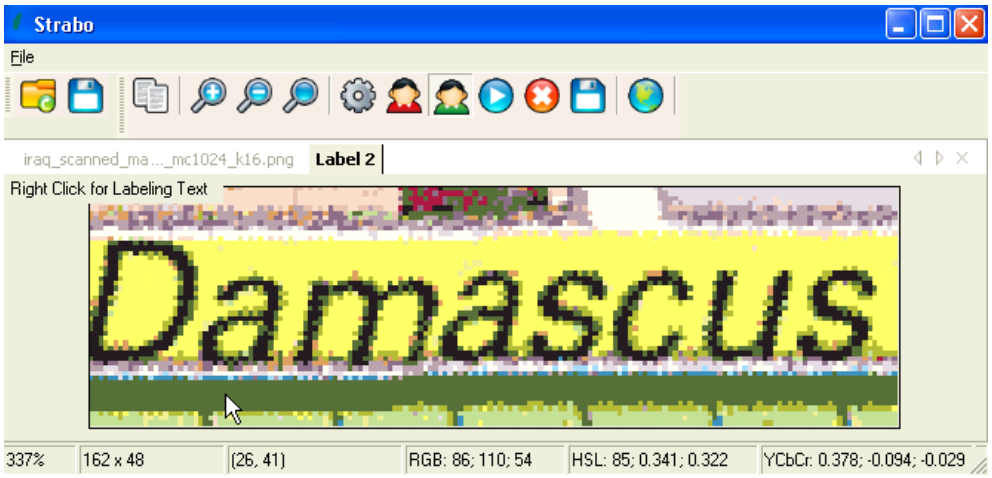
(a) Purple characters (b) Red characters (c) Black characters (d) Blue characters

Figure 4.3: Example text labels for characters of various colors

4.1.3 Automatically Identifying Text Colors Using Example Labels

Each text label contains a set of colors, and some of the colors represent text in the raster map. The TextLayerSeparator exploits the fact that the character pixels in a text label are spatially near each other and constitute a horizontal string, namely the horizontal-string property, to identify the text colors in a given text label. The TextLayerSeparator first decomposes a text label into a set of images so that every decomposed image contains only one color from the text label. For example, for the text label shown in Figure 4.3(a), the first row of Figure 4.5 shows the four decomposed images (background is shown in

(a) The user interface for user labeling



(b) The selected text label

Figure 4.4: The user labeling interface and the selected text label

black). The TextLayerSeparator discards the decomposed images where the number of foreground pixels is fewer than 15% of the total number of pixels since most of their foreground pixels do not have neighboring pixels in the horizontal direction.

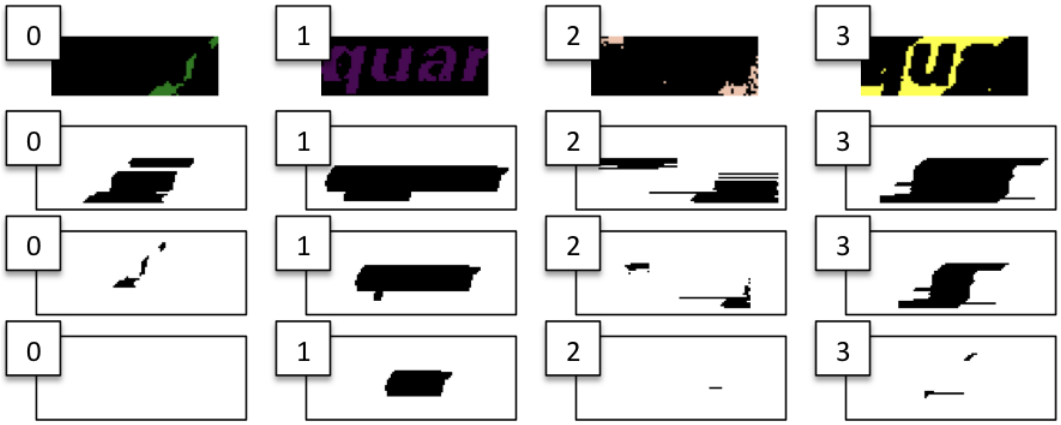


Figure 4.5: The decomposed images and the RLSA results

To identify the decomposed image that contains the text pixels, the TextLayerSeparator first expands the foreground areas in each decomposed image to grow a string blob. Since the height of a text label, H, is larger than the height of any character in the text label and the character height is usually longer than the character width, the horizontal pixel distance between two character pixels in a decomposed image should be less than H. Therefore, the TextLayerSeparator uses the closing operator with structure elements of height equal to one pixel and width equal to H to expand the foreground area for connecting character pixels and to grow a string blob. The closing operator is the dilation operator followed by the erosion operator. For a background pixel in a decomposed image, if there exists a foreground pixel in the horizontal direction within the distance of H, the dilation operator converts the background pixel to the foreground. Then, for each foreground pixel (including the ones converted by the dilation operator), if there

exists a background pixel in the horizontal direction within the distance of H, the erosion operator converts the foreground pixels to the background. The second row and third row in Figure 4.5 shows the results after applying the dilation and erosion operators on the decomposed images shown in the first row (background is shown in white). Finally, the TextLayerSeparator applies the erosion operator with a structure element of height equal to one pixel and width equal to H again to further eliminate false-positive branches of the string blob. The fourth row in Figure 4.5 shows the results after applying the erosion operator (background is shown in white).

The usage of the closing operator followed by the erosion operator is the morphological operator implementation of the run-length smoothing algorithm (RLSA), which is commonly used in document analysis techniques to identify string blocks from character pixels [Najman, 2004; Wong and Wahl, 1982]. Since the character pixels are horizontally near each other in the text label, the TextLayerSeparator uses the number of the remaining foreground pixels after RLSA as a measure to determine if the foreground pixels (non-black pixels) in a decomposed image represent text in the raster map. For example, *Image 1* in Figure 4.5 has the most remaining foreground pixels after the RLSA and hence the color of the foreground pixels in *Image 1* is identified as the text color.

There are two exceptions to using the horizontal-string property to identify text colors in the text label. One exception is that when background with uniform color exists in the text label, the color of the foreground pixels in the decomposed image that represents the background can be mis-identified as the text color. The first row of Figure 4.6 shows the decomposed images of the text label shown in Figure 4.3(c) and the second row shows the foreground pixels after the RLSA. The text label in Figure 4.3(c) has a uniform

background, which is the foreground pixels of *Image 0*. The foreground pixels of *Image 0* are horizontally near each other. In this case, the user would need to provide a non-text label containing the color in *Image 0*. If a non-text label exists, the RLSA algorithm processes only the decomposed images that do not contain their foreground colors that are in a non-text label. For example, with a non-text label having the color in *Image 0*, my algorithm discards *Image 0* and selects the color of the foreground pixels in *Image 1* as the text color since *Image 1* now has the most remaining foreground pixels.

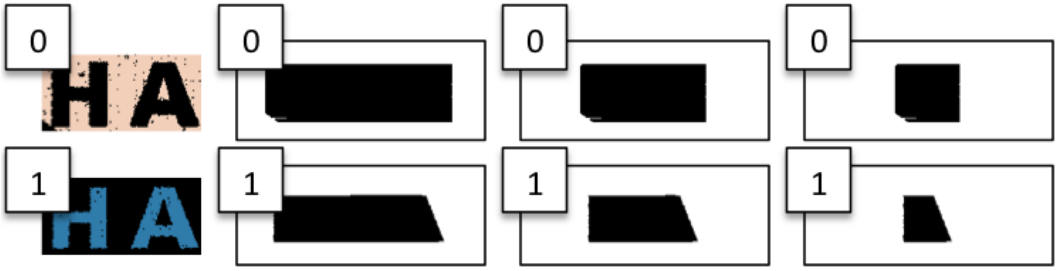


Figure 4.6: An example of a text label with uniform background

A second exception is when the characters in the text label are represented by multiple colors and the number of pixels of every color is less than 15% (i.e., non-solid characters). The top-left corner of Figure 4.7 shows a text label from Google Maps and the 28 rectangles are the decomposed images from the text label (background is shown in black). Only the decomposed image on the top-right corner that represents the background of the text label has more than 15% of foreground pixels. In this case, if no text color is identified after excluding the colors in the non-text label, all the colors in the text label except the ones in the non-text label are identified as the text colors. Figure 4.8 shows the identified text colors (background is shown in white) when the non-text label exists.

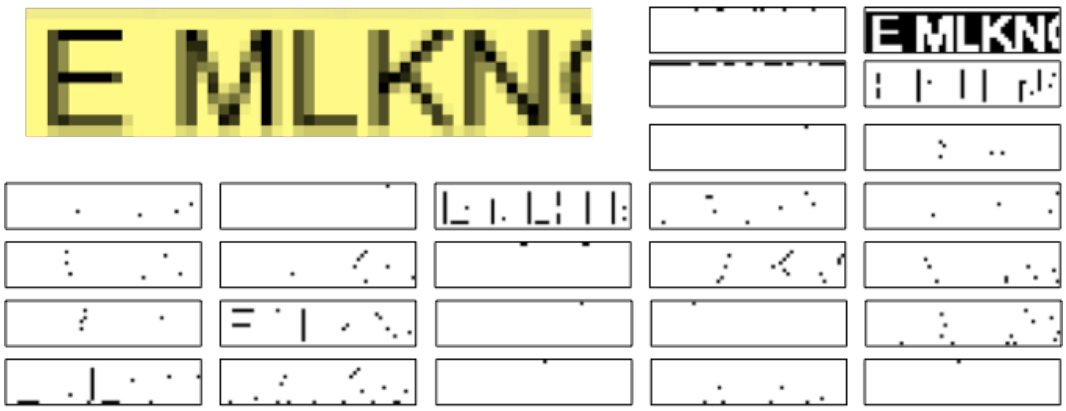


Figure 4.7: An example of non-solid characters

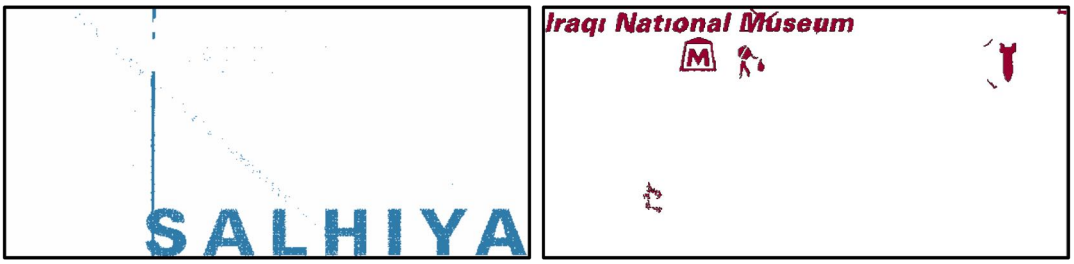
(a) The non-text label (b) The identified text colors

Figure 4.8: Using non-text label to identify text colors from raster maps with non-solid characters

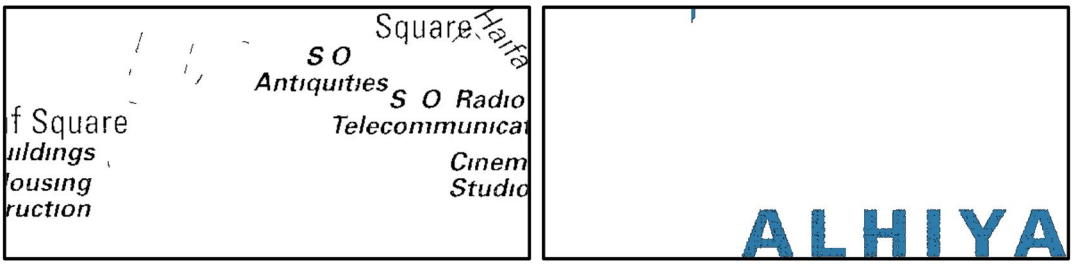
The TextLayerSeparator processes every text and non-text label and identifies a set of text colors from each text label. Then, for each pixel in the quantized map, if the pixel's color is not one of the text colors, the TextLayerSeparator converts the pixel to the background. For example, to extract the text layer of red characters from the map in Figure 4.2, the user selects the text label in Figure 4.9(a) and then the TextLayerSeparator generates the text layer in Figure 4.9(e) automatically (background is shown in white). Similarly, to extract the text layer of black pixels, the user selects the text label in Figure 4.9(b) and then the TextLayerSeparator generates the text layer in Figure 4.9(f) automatically. For the text layer of blue pixels, the user first provides the text label in Figure 4.9(c) but the uniform background is mis-identified as the text color. Therefore, the user provides the non-text label in Figure 4.9(d) together with the text label in Figure 4.9(c) to extract the text layer of blue characters shown in Figure 4.9(g).

(a) The text label for red characters (b) The text label for black characters (c) The text label for blue characters (d) The non-text label

(e) The text layer of red characters (f) The text layer of black characters



(g) The text layer of blue characters (h) The text layer of red characters after post-processing



(i) The text layer of black characters after post-processing (j) The text layer of blue characters after post-processing

Figure 4.9: Example results of the supervised text-layer extraction

There are still non-text objects in the extracted text layer if the non-text objects have the same color as the text, such as the scattered small objects in the three text layers or the lines in Figure 4.9(g). Therefore, the TextLayerSeparator applies post-processing using the connected-component analysis to filter out the non-text connected components based on the size of a connected component. For a connected component, A, and its bounding box, Abx, the size of A is given by:

$$Size = Max(Abx.Height, Abx.Width) \tag{4.2}$$

For a text layer, the text labels used to generate the text layer contain a set of horizontal text strings in the text layer, namely the example strings. The TextLayerSeparator computes the average size of the connected component in the example strings and then filters out the connected component in the text layer, which is smaller than half or larger than twice of the average size. This filtering rule keeps most of the connected component in the text layer and filters out only extreme cases, such as a long linear objects or single-pixel connected component. Figure 4.9(h) to Figure 4.9(j) shows the results after post-processing. The characters that overlap a large connected component, such as the 'S' shown in Figure 4.9(g), could also be removed. In this case, text separation algorithms [Cao and Tan, 2002] can be used to keep the overlapping characters.

4.2 Automatic Recognition of Text Labels

In this section, I present my AutoTextRecognizer technique that recognizes text labels in the separated text layers automatically. The first step of the AutoTextRecognizer is my

conditional dilation algorithm for locating individual strings in the extracted text layers, which contain multi-oriented text strings of various sized characters. The conditional dilation algorithm exploits the heuristic that the characters in a text string should have similar size and be spatially closer than the characters in two separate strings to locate the individual text strings. Similar to the AutoRoadVectorizer described in Chapter 3, instead of processing the entire map at once, the AutoTextRecognizer divides the map into overlapping tiles, utilizes the conditional dilation algorithm to locate individual text strings in each tile, and merges the identified text strings. Finally, with the identified individual strings, the AutoTextRecognizer detects their orientations and utilizes a conventional OCR product to recognize the characters.

4.2.1 Conditional Dilation Algorithm (CDA)

The conditional dilation algorithm (CDA) is an image filter that expands the foreground area of the connected components in an input image when certain conditions are met. The purpose of the expansion is to determine the connectivity between the connected components in the input image. Figure 4.10 shows the pseudo-code for the CDA. An iteration of the CDA includes two passes on the input image. Given a set of conditions, the first pass identifies the pixels that meet the conditions as the expansion candidates. Then, the second pass, which is a verification pass, confirms that the existence of an expansion candidate does not generate a connection between two connected components that violate the conditions using the candidate's neighboring foreground pixels and expansion candidates. The CDA runs from one iteration until the first pass cannot identify any more expansion candidates.

```
// The list for storing the identified expansion candidates
ExpansionCandidateList;

// The current number of iterations of the conditional dilation process
IterationCounter = 0;

// The current number of expandable connected components
Expandiable_CC_Counter;
```

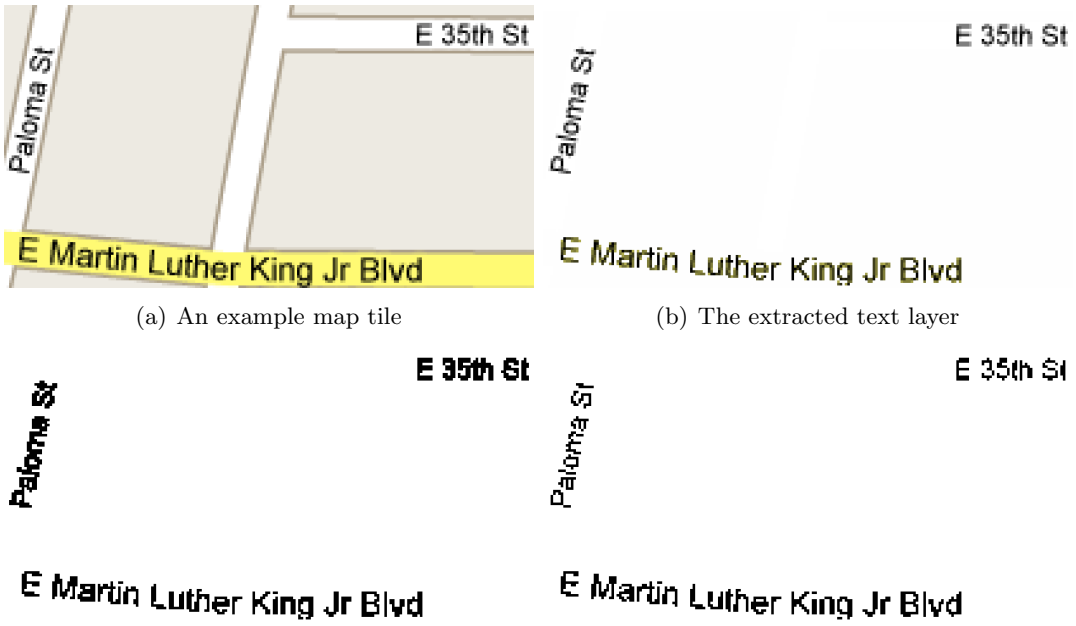
```
Function void ConditionalDilation(int[,] image, double size_ratio)
    For each connected component CC in image {
        CC.expandable = true;
        Do {
            FirstPass(image, size_ratio);
            VerificationPass(image, size_ratio);
            CountExpandibleCC(image);
            IterationCounter = IterationCounter +1;
        } While(Expandiable_CC_Counter > 0)
    }

Function void FirstPass(int[,] image)
    For each background pixel BG in image {
        if (PassConnectivityTest(BG) && PassSizeTest(BG) &&
            PassExpandibilityTest(BG) && PassStraightStringTest(BG) )
            ExpansionCandidateList.Add(BG);
    }

Function void VerificationPass(int[,] image)
    For each expansion candidate EC in ExpansionCandidateList {
        if (PassConnectivityTest(EC) && PassSizeTest(EC) &&
            PassExpandibilityTest(EC) && PassStraightStringTest(EC) )
            image.SetAsForeground(EC);
    }

Function void CountExpandibleCC (int[,] image)
    For each connected component CC in image {
        if (HasConnectedToTwoCCs(CC) ||
            IterationCounter > DynamicDistanceThreshold(CC.size))
            CC.expandable = false;
        else
            Expandiable_CC_Counter  = Expandiable_CC_Counter +1;
    }
```

Figure 4.10: The pseudo-code for the conditional dilation algorithm (CDA)

4.2.1.1 Input

The input of the CDA is a binary text layer, which contains characters as individual connected components (i.e., character components). From an extracted text layer, I generate a binary text layer automatically based on the character types (i.e., solid or non-solid characters) of the text labels used to extract the text layer. If the text labels contain only solid characters, I use every foreground pixel in the text layer as the foreground pixels of the binary text layer and everything else is the background.

For the text labels containing non-solid characters, the AutoTextRecognizer uses the foreground pixels in the text layer that are closer to the foreground (black) than the background (white) as the foreground pixels in the binary text layer. This is because the text layers of non-solid characters are pixilated and very often contain foreground pixels that represent the shadow around characters to emphasize the visualization of the characters. If we use all foreground pixels in the text layer containing non-solid characters as foreground pixels in the binary text layer, the shadow pixels can connect two characters as a connected component and hence we do not have characters as individual connected components. Figure 4.11(a) shows a map tile from Google Maps and Figure 4.11(b) shows the extracted text layer containing non-solid characters. Figure 4.11(c) shows the binary text layer if every foreground pixel in Figure 4.11(b) is a foreground pixel in the binary text layer, where many of the characters are connected, especially the text strings of "Paloma St" and "E 35th ST". Figure 4.11(d) shows the binary text layer after we remove the shadow pixels and most of the characters are individual connected components.



(a) An example map tile

(b) The extracted text layer

(c) The binary text layer with character shadow

(d) The binary text layer without character shadow

Figure 4.11: Generating a binary text layer from a text layer with non-solid characters

4.2.1.2 The First Pass

In the first pass, the CDA checks every background pixel in the input image. If a background pixel meets all of the following conditions, the CDA sets the background pixel as an expansion candidate. These conditions do not have to be checked in a fixed order.

First, *the connectivity condition*. An expansion candidate needs to connect at least one and at most two connected components. Since the maximum neighboring characters that any character in a text string can have is two, an expansion candidate cannot connect to more than two connected components.

Second, *the size condition*. If an expansion candidate connects to two connected components, the sizes of the two connected characters must be similar. The size of a connected component is defined in Equation 4.2. For any connected character components,

their size ratio must be smaller than two. For two connected components, A and B, their bounding boxes are Abx and Bbx, their size ratio is given by:

$$SizeRatio = \frac{Max(Size(A), Size(B))}{Min(Size(A), Size(B))} \tag{4.3}$$

The size limitation guarantees that every character in an identified text string has a similar size. The limitation of size ratio equal to two is because some letters, such as the letter 'l' and 'e', do not necessarily have the exact same size, even when the same font size and type is used.

Third, *the expansibility condition*. An expansion candidate needs to connect to at least one expandable character component. Before the first CDA iteration, every character is expandable. After each iteration, the CDA checks the connectivity of each character and if the character has already connected to two other characters, the character is not expandable. Then, for the remaining character components with connectivity numbers less than two, the CDA compares the number of iterations that have been done and the size of the character component to determine the expandability of the character components. This is to control the longest distance between two characters that the CDA can connect. Since one iteration expands the area of a character by one pixel, the CDA connects two characters that are within two pixels after one iteration if the two character components satisfy other conditions. The CDA uses a dynamic distance threshold as 1/5 of the character size. For example, for a character of size equal to 20 pixels, it will not be expandable after four iterations, which means the character can only

find a connecting neighbor within the distance of four pixels plus 1/5 of the size of a neighboring character component.

Using the iteration of the CDA to limit the distance between two characters in a string is more reliable than using the distance between the bounding-box centers of the character components. This is because distance between the bounding-box centers is based on the locations of the bounding-box centers, which can vary depending on the sizes and orientations of the character components, and the shortest distance between two characters does not have this dependency. For example, as shown in Figure 4.12, the lengths of the green lines that represent shortest distances between two neighboring characters in a text string are similar, and do not depend on the characters' sizes and orientations.

Figure 4.12: The shortest distance between two character components does not depend on the characters' sizes and orientations

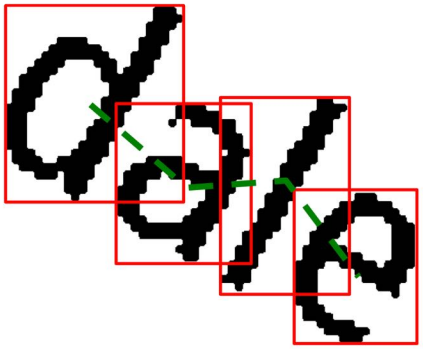
Fourth, *the straight-string condition.* If an expansion candidate connects to two character components and at least one of the two character components has a connected neighbor (i.e., a string with at least three characters), the connected character-components should constitute a straight string. This limitation guarantees that the CDA does not

connect the text strings of different orientations. However, to determine a straight string using only the character components without knowing how the characters are aligned is difficult because the connecting angle between three characters in a straight string is not necessarily 180 degrees. Considering the text string "Wellington", we can calculate the connecting angles between any three neighboring connected characters in the string by connecting the mass centers or bounding-box centers of the character components. Since the characters have various heights, the connecting angle between "Wel" is very different than the one between "ell". Therefore, using the connecting angles to determine a straight string is unreliable.

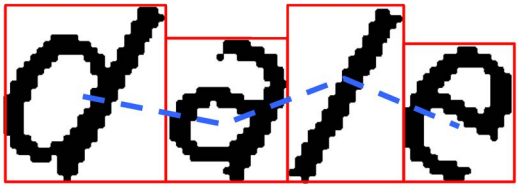
Instead of using only the connecting angles, the CDA first establishes a connecting-angle baseline for a text string, which represents the connecting angles between any three neighboring connected characters as if the string is in the horizontal direction. Then, the CDA compares the connecting-angle baseline to the actual connecting angles in the string to determine a straight string. To calculate the baseline, The CDA first aligns each of the characters in the string vertically and rearranges their positions in the horizontal direction. Then, the CDA calculates the connecting angles between the characters using the connecting lines between the bounding-box centers of the characters. This connecting angle is called the *normalized connecting angle* and the connecting-angle baseline is a set of the *normalized connecting angles*.

Figure 4.13 shows two examples. The green dashed line in Figure 4.13(a) shows the lines connecting the bounding-boxes centers for calculating the actual connecting angles. The blue dashed line in Figure 4.13(b) shows the lines connecting the rearranged bounding-boxes centers for calculating the connecting-angles baseline. Figure 4.13(c)

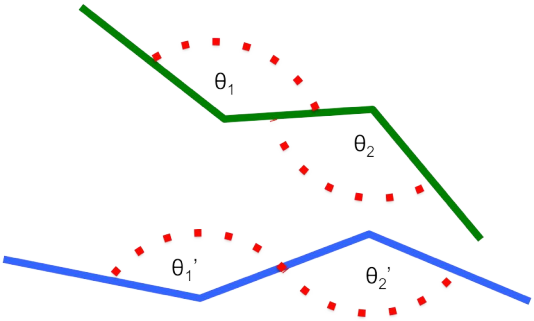
shows that the θ_1 is similar to θ_1' and θ_2 is similar to θ_2' and hence the string "dale" is a straight string. Figure 4.13(d) to Figure 4.13(f) shows an example of a non-straight string where θ_1 and θ_1' are very different.



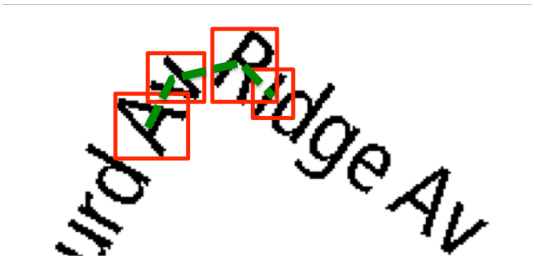
(a) The actual connecting angle of "dale"



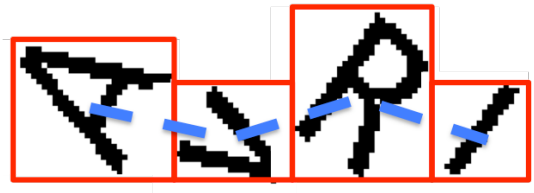
(b) The connecting-angle baseline of "dale"



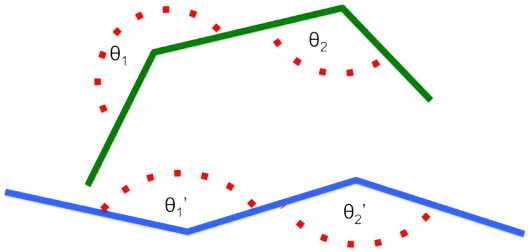
(c) The actual connecting angle is similar to the baseline



(d) The actual connecting angle of "AvRi"



(e) The connecting-angle baseline of "AvRi"



(f) The actual connecting angle is very different than the baseline

Figure 4.13: Comparing the angle baseline with the connecting angle to test the straight string condition

With the connecting-angle baseline, the CDA calculates the actual connecting angle between the characters using their bounding boxes and if the differences between the

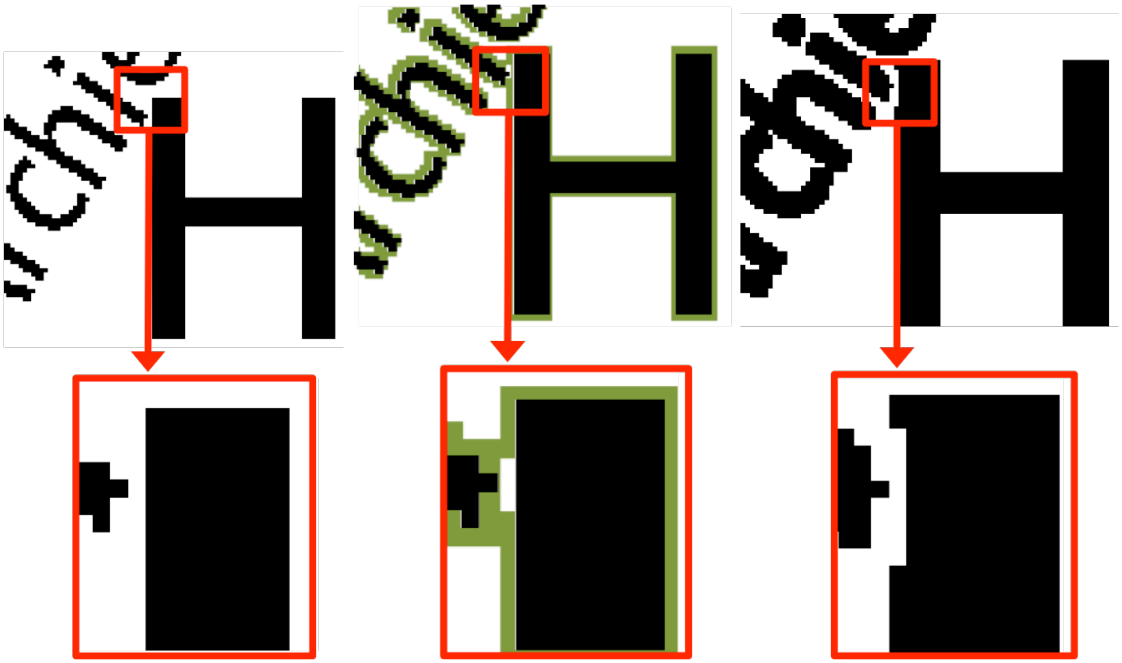
actual connecting angle and the *normalized connecting angle* are smaller than 30%, the string satisfies the straight-string condition. I use the 30% threshold to prevent breaking text strings that are slightly curved.

4.2.1.3 The Verification Pass

The verification pass checks each expansion candidate using the same conditions in the first pass. When the CDA marks an expansion candidate in the first pass, the CDA checks only the foreground pixels connecting to the expansion candidate. If the CDA further marks one of the expansion candidate's connecting background pixels as an expansion candidate after the first pass, we need to verify if the connectivity between the two expansion candidates is valid. For example, Figure 4.14(a) shows two characters that should not be connected because of the size limitation. Figure 4.14(b) shows the results after the first pass, where the green pixels are the identified expansion candidates. After the first pass, the CDA marks all background pixels that directly connect to the two characters as expansion candidates. If the distance between the two characters is two pixels, such as the areas in the red rectangle shown in Figure 4.14(a), after the first pass, the CDA fills up the two-pixel area with expansion candidates and the two characters are then connected. Therefore, we need the verification pass to verify the expansion candidates. Figure 4.14(c) shows the results after the verification pass.

4.2.1.4 Output

The output of the CDA is a binary image and the individual connected components in the output image represent a text string. Figure 4.15(a) shows an example text layer



(a) An example map area (b) After the first pass (expansion candidates shown in green) (c) After the verification pass

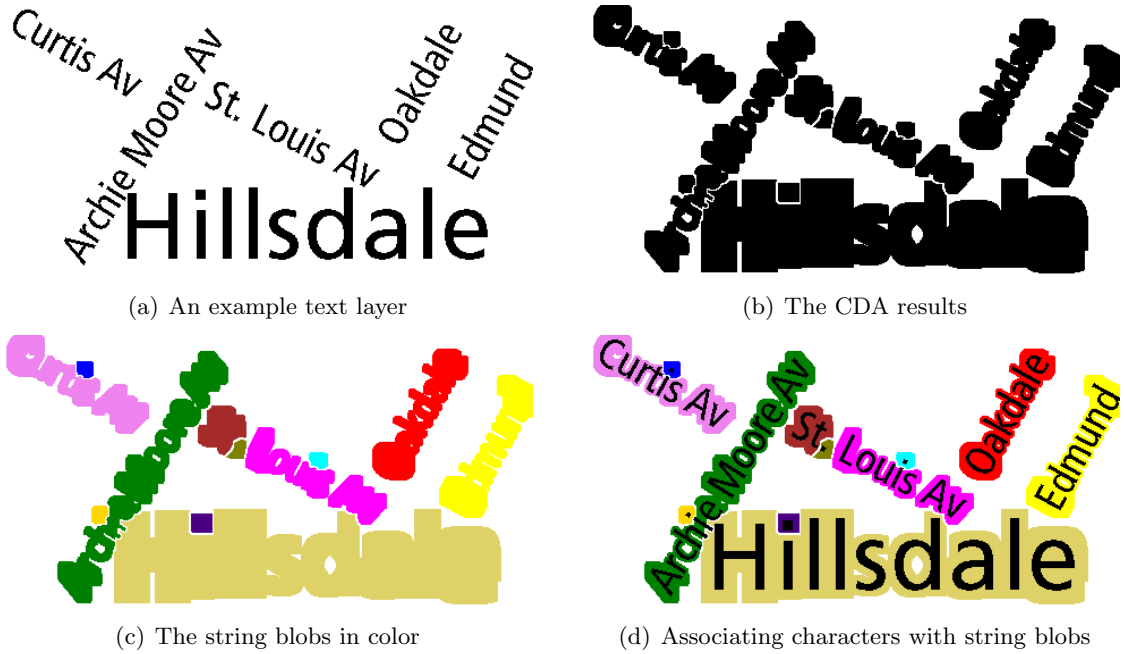
Figure 4.14: Using the verification pass to determine actual expansion pixels (background is shown in white)

and Figure 4.15(a) shows the CDA results. Figure 4.15(c) shows the string blobs; each is shown in a different color. Once the CDA has identified the string blobs, it overlaps the text layer with the string blobs to group the characters into text strings as shown in Figure 4.15(d).

The CDA does not group the small dots in the text layer correctly to string blobs, such as the dot on top of the character 'i' as shown in Figure 4.15. This is because the sizes of the dots are too small compared to their neighboring character components (i.e., violating the size condition). The OCR system will recover the missing small parts of characters in the character recognition step, which is simpler than adopting special rules for connecting the small parts. This is because grouping a small part to a correct string blob requires additional rules on handling the size differences between the connected components in the binary text layer, which can lead to unreliable results. For example, if the grouping between the dot of "St." and the character 't' is allowed, it is very likely that the character 'd' in "Hillsdale" will connect to the character 'A' in "Av" before 'd' can connect to 's' and 'a' in "Hillsdale".

4.2.2 Divide-and-Conquer Identification of Text Labels

The AutoTextRecognizer utilizes a divide-and-conquer (DAC) technique that divides a raster map into overlapping tiles and processes each tile to recognize their text labels. Figure 4.16(a) shows an example text layer with two overlapping tiles. After the DAC processes all the tiles, it merges the recognized text from each tile as one set of text strings for the entire raster map. Before the processing of each tile, the DAC first removes the connected components that touch each tile's borders since the touching connected



(a) An example text layer

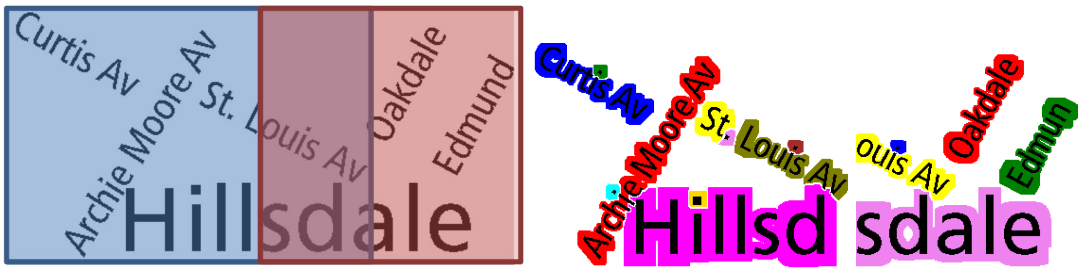
(b) The CDA results

(c) The string blobs in color

(d) Associating characters with string blobs

Figure 4.15: The CDA output

components might only be a portion of a character. The overlapping area should be larger than any of the characters in the text layer so that the characters near the tile borders always exist in one of the tiles. For example, Figure 4.16(b) and Figure 4.16(c) shows the text identification results of the two overlapping tiles using the conditional dilation algorithm. In the left tile, the character 'a' of the text string "Hillsdale" in the text layer is on the tile border and hence the DAC removes the 'a' before applying the conditional dilation algorithm. For the identified strings in two neighboring tiles, the DAC merges the stings that contain one or more characters that are the same. For example, Figure 4.16(b) and Figure 4.16(c) show that the conditional dilation algorithm identifies the text string, "Hillsd", from the left tile and the text string, "sdale", from the right tile. Since the two characters "sd" in the original text layer exist in both text strings, the DAC merges the two strings into one as "Hillsdale".



(a) An example text layer (b) The left tile (c) The right tile

Figure 4.16: The divide-and-conquer processing

As I described in Chapter 3, this divide-and-conquer approach to process raster maps scales to large images and can take advantage of multi-core or multi-processor computers to process the tiles in parallel.

4.2.3 Automatically Detecting String Orientation

Skew correction is well developed in modern OCR techniques; however, classic skew correction can only apply to multi-line and multi-word documents since the line spacing and the word spacing is exploited to detect the tilt angle, such as the morphological-operator-based RLSA method [Najman, 2004].

The morphological-operator-based RLSA method first uses the closing operator to generate a set of string blobs and then uses the erosion operator to reduce the areas of the string blobs. The closing operator is used to merge characters in a string so the structure-element size of the closing operator should be larger than the width of a character. The erosion operator is used to eliminate the string blobs that are shorter horizontally than the width of the erosion operator's structure elements. For example, in a document of 500 words with 50% of the words longer than 300 pixels, if we apply RLSA with the erosion operator using a structure element of width 300 pixels, we have

250 words remaining after the RLSA if the document is in the horizontal direction. If the document is tilted by 45 degrees, the lengths of the words in the horizontal direction is the original length multiplied by cos(45°) and hence fewer than 250 words will remain after the RLSA. Therefore, by detecting the number or the areas of the remaining words, we can determine the tilt angle in a multi-word document.

For the AutoTextRecognizer to detect the orientation of a single string, I present a single-string orientation detection algorithm (SSOD) that is based on the morphological-operator-based RLSA and uses dynamically generated structure elements for the morphological operators (i.e., the closing and erosion operators) [Chiang and Knoblock, 2010]. Figure 4.17 shows the pseudo-code of the SSOD.

```
// The list for storing the rotated string images
RotatedImageList;

Function int SSOD (Image string_image)
   AvgSize = FindAvgCCSize(string_image);
   For angle = MinRotation to MaxRotation {
      rotated_image = Rotate(string_image, angle);
      RotatedImageList.Add(rotated_image);
   }
   MaxWidth = FindMaxStringWidth();
   MaxForegroundPixelCount =
                     RLSA(AvgSize , MaxWidth);
   For each rotated image RI in RotatedImageList {
      if(RI. ForegroundPixelCount ==
                           MaxForegroundPixelCount )
         return RI.RotatedAngle;
   }
```

```
Function double FindAvgCCSize(Image string_image)
   double AvgSize;
   For each connected component CC in string_image {
      AvgSize = AvgSize + CC.Size;
   }
   return AvgSize/string_image .TotalNumberOfCCs;

Function int FindMaxStringWidth()
   StringWidthList;
   For each rotated image RI in RotatedImageList {
      Pixel left =
         FindTheLeftMostForegroundPixel(RI);
      Pixel right =
         FindTheRightMostForegroundPixel(RI);
      StringWidthList.Add(right.X – left.X);
   }
   return StringWidthList.MaxValue;

Function int RLSA(AvgSize , MaxWidth )
   RemainingForgroundPixelList;
   For each rotated image RI in RotatedImageList {
      RI = Closing(RI, AvgSize, MaxWidth);
      RI = Erosion(RI, MaxWidth);
      int FGC = RI.ForegroundPixelCount;
      RemainingForgroundPixelList.Add(FGC);
   }
return RemainingForgroundPixelList.MaxValue;
```

Figure 4.17: The pseudo-code for the single-string orientation detection algorithm

The SSOD first calculates the average size of the connected components in the text string, namely $AvgSize$. Then, the SSOD rotates the string image by 0 degrees to 179 degrees and calculates the maximum width of the bounding boxes of the rotated strings, namely $MaxWidth$. Considering a text string in the horizontal direction, we should find the $MaxWidth$ when the string is rotated to the 45 degrees. Therefore, we can use the length of $MaxWidth$ multiplied by $\cos(45°)$ to approximate the length of the string when the string is in the horizontal direction. I do not use the rotated string that has the $MaxWidth$ as the string of 45 degrees orientation to calculate the real orientation of the text string because the characters in the string can have various shapes and hence the $MaxWidth$ can be a few degrees off from the 45 degrees.

With the identified $AvgSize$ and $MaxWidth$, for each rotated string, the SSOD applies the closing operator using a structure element of height equal to one and width equal to $AvgSize$ to grow the string blobs, as the examples shown in the middle row in Figure 4.18. Then, the SSOD applies the erosion operator using a structure element of height equal to one and width equal to $MaxWidth$ multiplied by $\cos(45°)$ to shrink the area of the string blob. If the string blob is not in the horizontal direction, the erosion operator eliminates most of the blobs since their horizontal width is shorter than the approximate length of the horizontal string.

The SSOD identifies the horizontal string among the rotated strings using the number of remaining pixels after applying the erosion operator. The bottom row in Figure 4.18 shows example results after applying the erosion operator where the horizontal string has more remaining pixels than the tilted string.

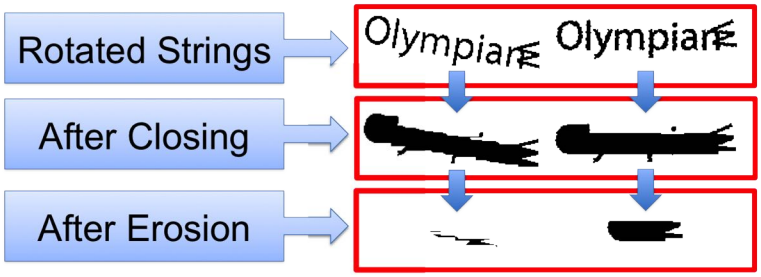


Figure 4.18: Detecting string orientation using the morphological-operators-based RLSA

The SSOD only applies on the strings having more than three connected components. This is because the detected orientation of a short string can be dominated by the heights of the short strings' connected components using the RLSA. Since short strings in a raster map are usually part of a longer string, the AutoTextRecognizer searches from the centroid of a short string for nearby strings and uses the orientations of the nearby strings as the short string's possible orientations. For example, the most common short strings in my test maps are "Av" as avenue, "Dr" as drive, and "Cir" as circle, which are all part of the road names. The AutoTextRecognizer uses a dynamic distance-threshold derived from the size of the bounding box of each short string to limit the search space.

4.2.4 Automatically Recognizing Characters Using OCR Software

Once the SSOD identifies the string orientations, the AutoTextRecognizer first rotates each string clockwise and counterclockwise to the horizontal direction according to its possible orientations (short strings might have more than one detected orientation depending on the number of its neighboring strings) to generate a set of rotated strings. Then, to identify the correctly oriented horizontal string (i.e., not the upside-down one), the AutoTextRecognizer sends all rotated strings to a commercial OCR product called

ABBYY FineReader 10 and automatically selects the rotated string with the highest returned recognition confidence.

The AutoTextRecognizer uses the number of connected component in the string, the number of recognized characters, and the number of suspicious characters to calculate the recognition confidence, which is defined as follows: NRC is the number of recognized characters, NSC is the number of suspicious recognized characters, and NCC is the number of connected components of a text string, the recognition confidence is given by:

$$RecognitionConfidence = \frac{NRC - NSC}{NCC} \tag{4.4}$$

If two rotated strings have the same highest recognition confidence, the AutoTextRecognizer uses the two recognition results in the final results. For short strings with fewer than three characters, if the recognition confidence is less than 50%, the AutoTextRecognizer discards the results since the short strings are very likely to be non-text objects that are mis-identified. For longer strings, if the recognition confidence is less than 50%, the AutoTextRecognizer sets the number of suspicious characters to zero and then recalculates the recognition confidence. This is because it is very likely that the quality of the original map is poor so the OCR software marks most of the recognized characters as suspicious.

4.3 Experiments

I have implemented the supervised text-layer-separation and automatic text-recognition approaches (the TextLayerSeparator and TextRecognizer) described in this chapter as the text recognition component in a map processing system called Strabo. In this section,

I report my experiments on the recognition of text label in heterogeneous raster maps using Strabo. I tested Strabo on 15 maps from 10 sources. The set of test maps includes three scanned maps and 12 computer generated maps directly generated from vector data, which are the same set of test maps used in the experiments of Chapter 3 for extracting road vector data, except one scanned map that has very poor image quality and is not suitable for the character recognition task.

For the scanned maps, we purchased three paper maps covering the city of Baghdad, Iraq from the International Travel Maps, Gecko Maps, and Gizi Map. I scanned the Bagdad paper maps in 350 DPI to produce the raster maps. For the computer generated maps, we purchased a map in PDF format covering the city of St. Louis, Missouri from the Rand McNally website.[5] I downloaded a map covering Afghanistan in PDF format from the United Nations Assistance Mission in Afghanistan website.[6] I cropped 10 computer generated maps from five web-mapping-service providers, Google Maps, Microsoft Live Maps, OpenStreetMap, MapQuest Maps, and Yahoo Maps. The 10 computer generated maps cover two areas in the U.S. One area is in Los Angeles, California, and the other one is in St. Louis, Missouri.

Table 4.1 shows the information of the test map and their abbreviations that we use in this section. For the ITM, GECKO, GIZI, RM, and UNAfg maps, I crop a center area for the text recognition experiments. The total numbers of characters and words in the testing area are shown in Table 4.1.

[5]http://www.randmcnally.com/

[6]http://unama.unmissions.org/

Map Source (abbr.)	Map Count	Map Type	No. of Chars./Words
International Travel Maps (ITM)	1	Scanned	1358/242
Gecko Maps (GECKO)	1	Scanned	874/153
Gizi Map (GIZI)	1	Scanned	831/165
Rand McNally (RM)	1	Computer Generated	1154/266
UN Afghanistan (UNAfg)	1	Computer Generated	1607/309
Google Maps (Google)	2	Computer Generated	401/106
Live Maps (Live)	2	Computer Generated	233/64
OpenStreetMap (OSM)	2	Computer Generated	162/42
MapQuest Maps (MapQuest)	2	Computer Generated	238/62
Yahoo Maps (Yahoo)	2	Computer Generated	214/54

Table 4.1: Test maps

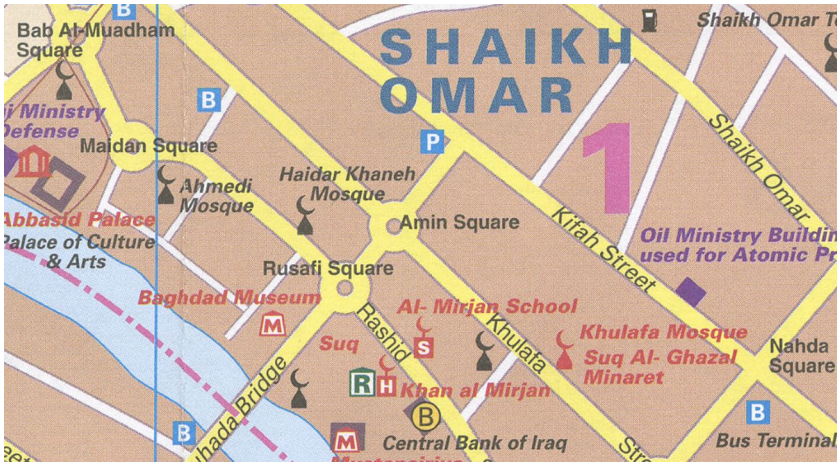
Figure 4.19 to Figure 4.22 shows the example areas and text of each test map, where the scanned maps show poor image quality compared to the computer generated maps. In addition, the text labels in the computer generated maps of Google, Live, OSM, MapQuest, and Yahoo contain pixilated non-solid characters, which is especially difficult for an OCR system to recognize even if the text labels are in the horizontal direction.

4.3.1 Experimental Setup

I utilized Strabo and a commercial OCR product called ABBYY FineReader 10 to recognize the text labels in the test maps. For comparison, I tested the 15 test maps using only ABBYY FineReader 10 by directly loading each of the test maps into the OCR software for the automatic character recognition.

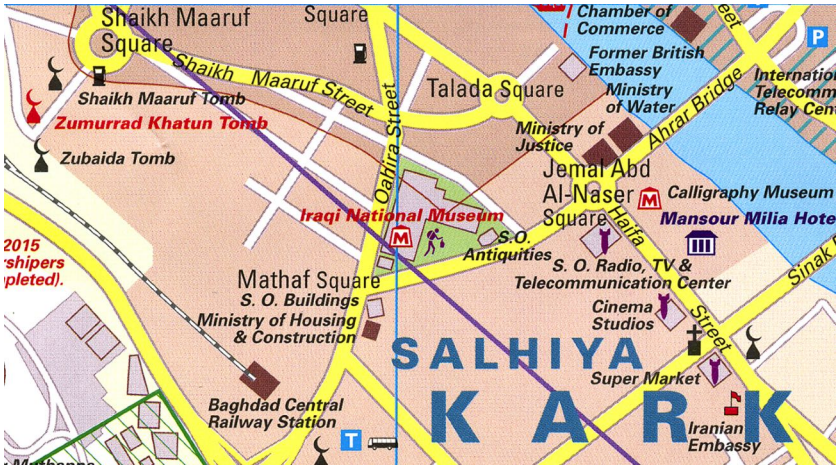
4.3.2 Evaluation Methodology

To evaluate my supervised text-layer-extraction approach, I report the number of user labels that were required for extracting text pixels from a test map using Strabo. For



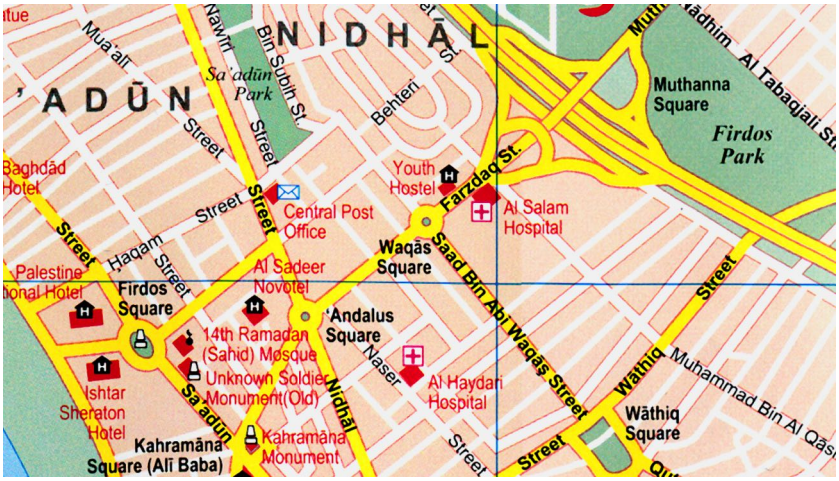
(a) ITM map

(b) ITM text



(c) GECKO map

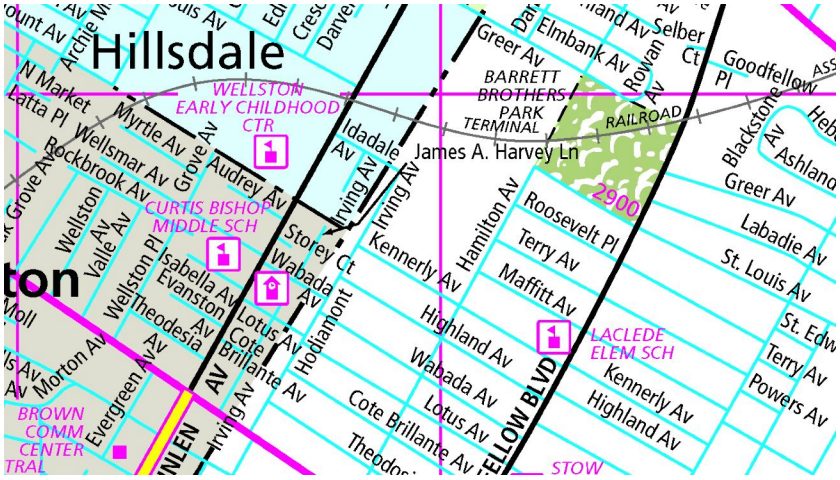
(d) GECKO text



(e) GIZI map

(f) GIZI text

Figure 4.19: Examples of the ITM, GECKO, GIZI maps

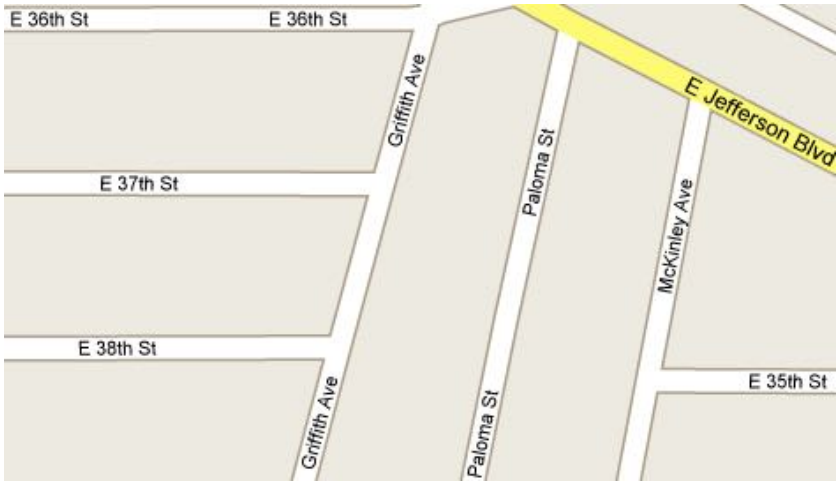


(a) RM map

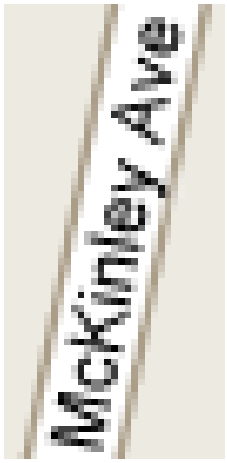
(b) RM text

(c) UNAfg map

(d) UNAfg text

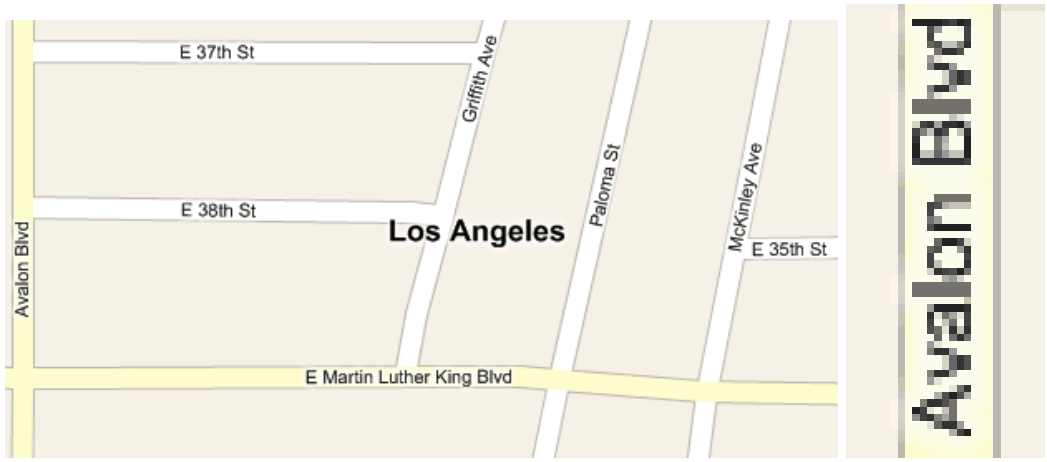


(e) Google map



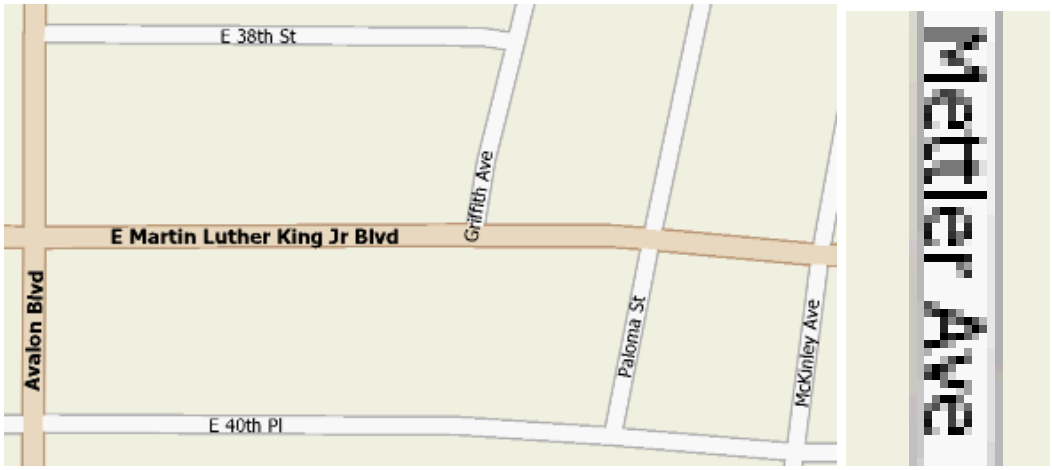
(f) Google text

Figure 4.20: Examples of the RM, UNAfg, Google maps



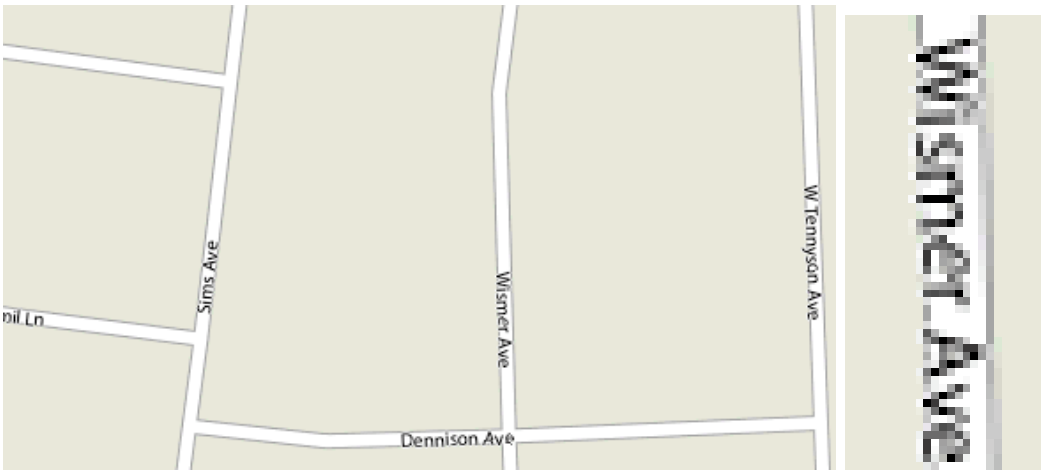
(a) Live map (b) Live text

(c) OSM map (d) OSM text



(e) MapQuest map (f) MapQuest text

Figure 4.21: Examples of the Live, OSM, MapQuest maps



(a) Yahoo map (b) Yahoo text

Figure 4.22: Examples of the Yahoo map

the 15 test maps, three maps are scanned maps, which contain numerous colors (ITM, GECKO, and GIZI). Strabo automatically quantized the three scanned maps using the Mean-shift, Median-cut, and K-means algorithms to generate four quantized images in different quantization levels for each map (the four quantization levels have the number of colors as 8, 16, 32, and 64, respectively). Strabo did not apply the color segmentation algorithms on the computer generated maps before user labeling. This is because the computer generated maps contain a smaller number of colors. The user started using Strabo for the user-labeling task from the quantized image of the highest quantization level (i.e., the quantized image that has the smallest number of colors). If the user could not distinguish the road pixels from other map features (e.g., background) in the quantized image, the user would need to select a quantized image containing more colors (a lower quantization level) for user labeling.

For evaluating the recognized text labels, I report the precision and recall on the both of the character and word levels. The character precision is given by:

$$Precision_{character} = \frac{Number of Correctly Recognized Characters}{Number of Recognized Characters} \tag{4.5}$$

The character recall is given by:

$$Recall_{character} = \frac{Number of Correctly Recognized Characters}{Number of Ground Truth Characters} \tag{4.6}$$

Similarly, the word precision is given by:

$$Precision_{word} = \frac{Number of Correctly Recognized Words}{Number of Recognized Words} \tag{4.7}$$

The word recall is given by:

$$Recall_{word} = \frac{Number of Correctly Recognized Words}{Number of Ground Truth Words} \tag{4.8}$$

4.3.3 Experimental Results

Table 4.2 shows the number of extracted text layers and the number of user labels for extracting the text layers. The ITM, GECKO, GIZI, RM, and UNAfg maps contain solid characters and hence fewer user labels were needed for extracting one text layer. For the other maps that have pixilated non-solid characters, the higher numbers of user labels were due to the fact that more non-text labels were used.

Map Source	No. of Extracted Text Layers	No. of User Labels (Text/Non-Text)	Total User Labels
ITM	3	6/3	9
GECKO	3	3/2	5
GIZI	2	2/2	4
RM	2	2/2	4
UNAfg	2	2/2	4
Google	2	2/4	6
Live	2	6/12	18
OSM	2	3/10	13
MapQuest	2	3/6	9
Yahoo	2	1/1	2

Table 4.2: The number of extracted text layers and the number of user labels for extracting the text layers

For the maps of Google, OSM, MapQuest, and Yahoo, I labeled one map from each source and used the labels to extract text layers from every map of the source. For example, for the two Yahoo maps, I labeled a text label and a non-text label to extract a text layer from one map and then I used the two user labels again to extract the text layer from a second Yahoo map.

Table 4.3 shows the numeric results from using Strabo and using ABBYY FineReader 10 itself to recognize text string in the 15 test maps. Strabo extracted 6,708 characters and 1,383 words from the test maps and ABBYY FineReader 10 extracted 2,956 characters and 655 words. The average character precision, character recall, word precision, and word recall of Strabo are 92.77%, 87.99%, 82.07%, and 77.58% compared to ABBYY's 71.99%, 30.09%, 46.11%, and 20.64%. Strabo produced higher numbers compared to only using ABBYY FineReader 10 in all metrics, especially the recall numbers. ABBYY FineReader 10 did not do well on identifying text regions because of the multi-oriented

text strings in the test maps. Without Strabo, ABBYY FineReader 10 could only recognize text stings that were in the horizontal direction. Moreover, my supervised text-layer extraction method separated text layers that had different colors so that there were only a few text strings overlapping with non-text objects in the extracted text layer. ABBYY FineReader 10 processed the test maps as a whole so that some overlapping text strings could not be recognized even when they were in the horizontal direction.

Map Source	**Char. P.**	**Char. R.**	**Char. F.**	**Word P.**	**Word R.**	**Word F.**
ITM (Strabo)	93.65%	93.37%	93.51	83.33%	82.64%	82.99%
ITM (ABBYY)	86.47%	45.66%	59.76	57.55%	33.06%	41.99%
GECKO (Strabo)	93.44%	86.38%	89.77%	83.1%	77.12%	80%
GECKO (ABBYY)	77.87%	41.08%	53.78%	66.28%	37.25%	47.7%
GIZI (Strabo)	95.12%	77.38%	85.34%	82.03%	63.64%	71.67%
GIZI (ABBYY)	71.35%	16.49%	26.78%	51.43%	10.91%	18%
RM (Strabo)	93.42%	94.8%	94.11%	87.94%	84.96%	86.42%
RM (ABBYY)	71.86%	10.4%	18.17%	23.53%	3.01%	5.33%
UNAfg (Strabo)	91.59%	88.05%	89.78%	82.39%	80.26%	81.31%
UNAfg (ABBYY)	65.67%	56.07%	60.49%	34.88%	36.57%	35.7%
Google (Strabo)	97.35%	91.77%	94.48%	89.22%	85.85%	87.5%
Google (ABBYY)	0%	0%	0%	0%	0%	0%
Live (Strabo)	94.78%	93.56%	94.17%	75.38%	76.56%	75.97%
Live (ABBYY)	51.87%	47.64%	49.66%	47.89%	53.13%	50.37%
OSM (Strabo)	95.45%	77.78%	85.71%	74.36%	69.05%	71.6%
OSM (ABBYY)	0%	0%	0%	0%	0%	0%
MapQuest (Strabo)	91.32%	84.03%	87.53%	81.03%	75.81%	78.33%
MapQuest (ABBYY)	0%	0%	0%	0%	0%	0%
Yahoo (Strabo)	69.74%	63.55%	66.50%	43.14%	40.71%	41.9%
Yahoo (ABBYY)	0%	0%	0%	0%	0%	0%
Avg. (Strabo)	92.77%	87.99%	90.32%	82.07%	77.58%	79.76%
Avg. (ABBYY)	71.99%	30.09%	42.44%	46.11%	20.64%	28.52%

Table 4.3: Numeric results of the text recognition from test maps using Strabo and ABBYY (P. is precision, R. is recall, and F. is the F-Measure)

For the test maps that contain pixilated non-solid characters (i.e., Google, Live, MapQuest, OSM, and Yahoo), ABBYY FineReader 10 only recognized some characters from the Live maps and reported that the other maps did not contain text. This was because automatically segmenting the pixilated non-solid characters for identifying text regions is difficult.

Overall Strabo achieved accurate text recognition results on both the character and word levels. The errors in Strabo's results came from several aspects: First, the poor image quality of the test maps could result in poor quality of text layers, such as broken characters or the existence of non-text objects in the text layer. In my experiments, the GIZI map had the worst image quality among the scanned maps and hence the result numbers of the GIZI map were the lowest among the maps with solid characters.

Second, the similarity between symbols led to false positives. There were many short strings of "Pl" for place in our test maps, and most of them were mis-identified as "PI" (a capital 'p' and a capital 'i'). This is because some font types, the capital 'i' is printed as 'l' and the OCR software mis-matched the two letters. Moreover, a string could be mis-identified as a totally different string when the string was upside down, especially short strings, such as "Pl" and "ld", "99" and "66". Figure 4.23 shows another example that the string "Zubaida" could be mis-identified as "epieqnz". This type of false positive resulting from similar symbols is very difficult to remove if the actual orientation of the string is unknown. One possible solution is to introduce additional knowledge of the map area to filter out unlikely results, such as "ld" and "PI" (a capital 'p' and a capital 'i').

Third, Strabo could not detect correct orientations for significantly curved text strings and the OCR software could not recognize all characters in curved strings. Figure 4.24

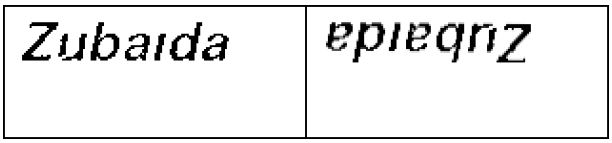
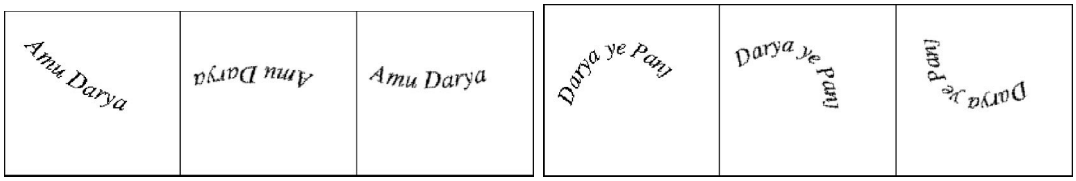


Figure 4.23: The string "Zubaida" can be mis-identified as "epieqnz"

shows two examples of curved strings and the rotated horizontal strings according to their detected orientations. If the string was slightly curved, such as the one in Figure 4.24(a), Strabo could detect correct orientation so that one of the rotated horizontal strings is correctly oriented (the third string in Figure 4.24(a)). For the curved string in Figure 4.24(b), Strabo could not detect the correct orientation since the RLSA could only work on slightly curved labels. However, even if Strabo identified the correct orientation, since the string was curved, the OCR software could not recognize all characters of the string; for example, the recognition results of the string in Figure 4.24(a) was "Darya" and the sub-string "Amu" was not identified. To overcome this problem, Strabo could use a lower threshold on comparing the connecting angle to the baseline for breaking any curved strings. Then, post-processing to merge the recognition results of the pieces of the curved strings should be used to recover the broken strings.



(a) A slightly curved string with correct detected orientation

(b) A curved string with incorrect detected orientation

Figure 4.24: Examples of the curved strings and their rotated images using the detected orientations

Fourth, the OCR software could not recognize many of the pixilated non-solid characters. For the pixilated non-solid characters, a character is not necessarily an individual

connected component, and the CDA might generate incorrect string blobs. The Yahoo maps had the most pixilated characters and hence the result numbers were the worst. In addition to the incorrect string blobs, the pixilated characters are difficult for a machine to process, although humans can recognize the pixilated characters from a distance, which is a similar problem to the CAPTCHA [von Ahn et al., 2003].

Fifth, the CDA might not group strings with wide character-spacing. The characters in a string that had wide character spacing were not correctly grouped since the CDA used a distance threshold depending on the sizes of the characters. For example, Figure 4.25 the string "Hindu Kush" in the UNAfg map were not identified correctly.

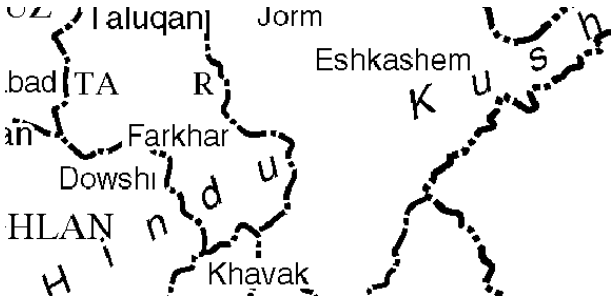


Figure 4.25: The string "Hindu Kush" has a wide character spacing

Sixth, the CDA might mis-group characters with non-text objects. The extracted text layers had only a few non-text objects and very often the non-text objects had very different sizes than the characters and were farther from a character than the character's neighbor. Therefore, the CDA did not group the non-text objects with characters. However, in some cases, the non-text objects were close to the ends of a string and hence when the characters in the strings expanded, the characters connected to a non-text object. For example, the first image from the left in Figure 4.26 shows a string "Qeysar"

grouped with a dashed line and the second and third images are the rotated images using the detected orientation. The non-text object unbalanced the string and hence the RLSA did not detect the correct orientation. A stricter connected-component filter can be used to post-process the extracted text layer for removing this type of errors. However, the connected-component filter might also remove characters and lower the recall of the results.

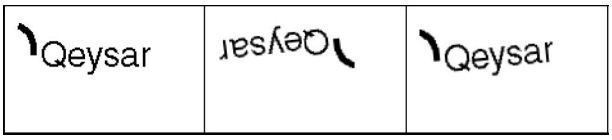


Figure 4.26: An example string with a non-text object and the rotated image using the detected orientation

Last, there was insufficient information from the OCR software for calculating the recognition confidence. Since ABBYY FineReader 10 is consumer grade software, its results did not provide much information for Strabo to select the correctly oriented rotated string. This could be solved by adopting a OCR SDK into Strabo, such as OCRopus.

4.3.4 Computation Time

I built Strabo using Microsoft Visual Studio 2008 running on a Microsoft Windows 2003 Server powered by a 3.2 GHz Intel Pentium 4 CPU with 4GB RAM. The average processing time for the CDA on a 1688x1804-pixels text layer of 626 characters was 37 seconds, for a 2905x2384-pixels text layer of 1092 characters was 39 seconds, and for a 850x550-pixels text layer of 78 characters was 2.6 seconds. Dominant factors of the computation time are the image size, the number of characters, and the shortest distance between two

characters in a string (a longer distance requires more iterations for the CDA to converge). The average processing time for detecting the orientation on a string longer than three characters was 2.2 seconds (a total of 922 strings), and the dominant factor on the computation time was the length of a string. Although I used multi-threading in several places in the system for parallel processing, the implementation was not fully optimized and improvements could still be made to speed the processing.

4.4 Conclusion

This chapter presents the TextLayerSeparator and AutoTextReconizers techniques. The TextLayerSeparator separates text layers from raster maps with poor image quality or complex maps, and the AutoTextReconizers automatically recognizes text labels from the separated text layer. My experiments show that the techniques separated text layers from raster maps of 10 sources with varying color usages and image quality using minimal user input and achieved accurate results recognizing text labels at both the character and word levels.

Chapter 5

Related Work

Separating and recognizing geographic features from raster maps is an active research area that spans many research fields, such as geography, pattern recognition, computer vision, image processing, and document analysis. Therefore, it is necessary to define the terminology used to review the related work in this chapter:

- *Digitizing* a map is the process of converting a printed map (e.g., a paper map) into electronic format, which is a raster image of the printed map (i.e., a raster map).

- *Computerizing* a map is the process of converting a printed map into a machine-editable format. For example, the results from map computerizing can be a map in Portable Document Format (PDF), where linked nodes (i.e., road vector data) represent the road lines.

- A *geographic feature* can be a set of linear objects, such as the road lines and contour lines, a set of character objects, a set of homogeneous-region objects, or a set of symbolic objects.

- A *geographic-feature layer* (feature layer) is a raster image that contains a set of pixels that represent an individual geographic feature in a map.

- *Separating* a geographic-feature layer or a geographic feature is the process of extracting the set of pixels that represent an individual geographic feature in a map.

- *Recognizing* a geographic feature is the process of converting the geographic feature into a machine-editable format, such as the road vectorization process. Therefore, computerizing a map is a process that recognizes every feature layer in a map. Recognizing a geographic feature does not necessarily require first separating the feature layer from the raster map.

In the following sections, I first review the related work on separating and recognizing geographic features from raster maps in general and then present the related work that focuses on separating and recognizing the geographic features of roads and text labels from raster maps

5.1 Separation of Feature Layers

Podlasov and Ageenko [2005] utilize user-specified colors to separate individual feature layers from the raster maps and then process the map using the morphological operators for removing the artifacts and restoring the feature layers, such as reconnecting broken contour lines resulting from the layer separation. To expand the foreground area of a target feature layer for restoring the layer, they first union all the feature layers excluding the target layer as a mask image. Then, for every iteration, to expand the foreground area of the target feature layer, they shrink the foreground area of the mask image. The

expansion is guaranteed to converge since the foreground area of the mask image eventually disappears. In comparison, I determine the number of iteration for the expansion to reconnect road lines in a road layer using the automatic detected road width and road format, which requires comparatively fewer iterations to restore the road layer.

For a more general approach, Leyk and Boesch [2010], Henderson et al. [2009], and Lacroix [2009] present automatic techniques to generate a set of color clusters for separating individual feature layers from raster maps. Their techniques are based on the assumptions that the color variation within a feature layer is smaller than the variation between feature layers and the pixels of the same layer are mutually close in the image.

Leyk and Boesch [2010] develop a color image segmentation (CIS) technique that considers the local image plane, frequency domain, and color space for clustering the pixels in raster maps into feature layers. The CIS technique handles raster maps with poor image quality, such as scanned historical maps with high degrees of mixed and false coloring and blurring. Henderson et al. [2009] utilize several classification techniques, the K-means, graphic theoretic, and expectation maximization, with a manually specified number of feature layers to classify the color space of the raster map for separating individual feature layers. Lacroix [2009] develops the median-shift classification technique that works in the image and color spaces for identifying a color palette that can be further used to separate individual feature layers from raster maps.

In comparison to the research in this thesis, the techniques presented in this section [Henderson et al., 2009; Lacroix, 2009; Leyk and Boesch, 2010; Podlasov and Ageenko, 2005] do not further investigate each separated feature layer and hence have no knowledge about the types of the features in the separated layers. The general techniques to separate

feature layers from raster maps [Henderson et al., 2009; Lacroix, 2009; Leyk and Boesch, 2010] can serve as a pre-processing step to generate the input image for my supervised techniques of separating the road and text layers, which may further reduce the number of required user labels.

5.2 Separation and Recognition of Feature Layers

In this section, I first present the related work on recognizing geographic features from raster maps that relies heavily on human operators. Then, I describe the related work that utilizes prior knowledge of the maps, such as map legends, to reduce the user interaction for recognizing geographic features from raster maps. In comparison, my research uses general heuristics of the road lines and text labels in a raster map for the layer separation, road vectorization, and text recognition instead of using any prior knowledge of the maps, which can be difficult to access. Moreover, my supervised techniques on separating road and text layers require only minimal user effort.

5.2.1 Separation and Recognition of Feature Layers With Intensive User Interaction

Leberl and Olson [1982] and Suzuki and Yamada [1990] present highly labor-intensive map processing systems that focus on hardware design and techniques to help human operators speed the map computerizing processes. For example, Leberl and Olson [1982] described techniques to aid the operator to move the cursor quickly, such as the line-following technique to automate the cursor movement.

The United Nations Statistics Division present a map computerizing system called MapScan [MapScan, 1998] for manually converting the linear features in raster maps into vector format and recognizing the text strings in raster maps. MapScan includes an extensive set of image processing tools (e.g., the morphological fiters) and labeling functions for the user to manually computerize the raster maps, which requires intensive user input. For example, to recognize text string using MapScan, the user needs to label the areas of each text string and the string has to be in the horizontal direction.

5.2.2 Separation and Recognition of Feature Layers With Prior Knowledge

Cofer and Tou [1972] present a Machine Automated Perception of Pictorial Symbology system (MAPPS) that recognizes feature layers from raster maps using a given set of geographic symbols. MAPPS includes hardware devices to digitize 35-mm transparencies to raster maps and a software program to identify lines and nodes from the raster maps. The feature recognition is achieved by matching the identified lines and nodes to a given set of symbols.

Samet and Soffer [1994, 1996] present a system named MARCO (denoting MAp Retrieval by COntent) that is used for the acquisition, storage, indexing, and retrieval of map images. MARCO exploits the map legend to recognize geographic symbols in raster maps, such as a fish-like symbol that represents recreational fishing spots in a map.

Myers et al. [1996] utilize a verification-based computer-vision approach to recognize geographic symbols in the raster maps using training samples of the symbols. To separate

linear structures and text areas from raster maps, they use prior knowledge of the semantic criteria (e.g., the lines that represent rivers are blue) and gazetteer information.

Dhar and Chanda [2006] separate a topographic map into layers using user-specified colors. For each layer, they identify the skeleton of each connected component and then recognize geographic symbols (e.g., a tree or a grass symbols) and characters using a set of user-specified rules applied on the geometric properties of the component skeletons, such as a tree feature as a connected component with one hole (e.g., a torus).

Kerle and de Leeuw [2009] separate and recognize the layer of population dots (i.e., a set of filled circles with varying sizes representing the numbers of population) from a historical scanned map of Kenya. Instead of working on the pixel level, they utilize object-oriented image processing techniques that first break the raster map into homogeneous square areas and then work on each square area for identifying the dots. The object-oriented image processing techniques work well on the dot feature; however, the features of irregular shapes, such as the text strings or elongated linear features, can cause a problem since the features cannot be broken down into homogeneous square regions.

5.3 Road Layer Separation and Road Vectorization

Much research has been performed in the field of extracting linear features from raster maps, such as separating lines from text [Cao and Tan, 2002; Li et al., 2000], detecting road intersections [Habib et al., 1999; Henderson et al., 2009], and vectorization of road lines [Bin and Cheong, 1998; Itonaga et al., 2003], generic linear objects [Ablameyko et al., 1993a, 1994, 1993b, 1998, 2002a,b, 2001; Ablameyko and Frantskevich, 1992; Bucha et al.,

2007], and contour lines [Arrighi and Soille, 1999; Chen et al., 2006b; Khotanzad and Zink, 2003].

One line of techniques uses simple methods to separate the foreground pixels or road layers from raster maps and hence can only handle specific types of maps. Cao and Tan [2002], Li et al. [2000], and Bin and Cheong [1998] utilize a preset grayscale threshold to remove the background pixels from raster maps and work on the foreground pixels to separate their desired features. The grayscale thresholding technique does not work on raster maps with poor image quality. In addition, the work of Cao and Tan [2002] and Li et al. [2000] focuses on recognizing text labels and they do not process the separated road layers further to reconnect the broken lines. Bin and Cheong [1998] extract the road vector data from raster maps by identifying the medial lines of parallel road lines and then linking the medial lines. The linking of the medial lines requires various manually-specified parameters for generating accurate results, such as the thresholds to group medial-line segments to produce accurate geometry of road intersections.

Habib et al. [1999] work on raster maps that contain only road lines for extracting road intersections automatically. They detect the corner points on the identified road edges and then group the corner points and identify the centroid of each group as the road intersection. False-positive corner-points or intersections of T-shape roads can significantly shift the centroid points away from the correct locations. Henderson et al. [2009] exploit the pre-defined color structure of the USGS topographic maps to first separate the road layer and then detect the road intersections using the tensor-voting framework [Mordohai and Medioni, 2006]. A manual step for separating the road layer is required if the raster map does not have a pre-defined color structure.

Itonaga et al. [2003] employ a stochastic-relaxation approach to first extract the road areas and then apply the thinning operator to extract one-pixel width road networks from raster maps with good image quality. The stochastic-relaxation approach does not work on scanned maps since the road areas do not have consistent color usage. Moreover, Itonaga et al. [2003] correct the distorted lines around the road intersections based on user-specified constraints, such as the road width.

Arrighi and Soille [1999] extract the pixels having a red hue as the layer of contour lines and then utilize the morphological operators to remove noise and connect small gaps. Then, they identify the extremities of lines and apply the skeletonization algorithm using the extremities as anchor points to extract the centerlines. Finally, the extremities of the lines are evaluated for connecting or disconnecting the lines.

For the techniques that are not limited to a specific type of map, Bucha et al. [2007] utilize the pixel-force field algorithm [Bucha et al., 2006] to extract the centerlines of linear objects from raster maps. The pixel-force field algorithm can apply directly on color maps without a binarization pre-processing step; however, the user needs to manually specify the start and end points of the linear feature.

In a series of publications [Ablameyko et al., 1993a, 1994, 1993b, 1998, 2002a,b, 2001; Ablameyko and Frantskevich, 1992], Ablameyko et al. present an interactive map interpretation system that first separates the feature layers from a raster map using user selected colors. Using user-selected colors to separate the feature layers can be laborious since the raster maps with poor image quality usually contain thousands of colors. For each feature layer, the map interpretation system produces the centerlines for linear features and the contours for region features. Then, the system links the centerlines or

contours to generate the final results. The line-linking step relies on user intervention, such as to confirm a merge between two line segments or to indicate the continuation of a line.

In comparison, my approach includes user training and is capable of handling diverse types of maps, especially scanned maps. Moreover, I automatically generate the road geometry and correct the distorted lines around road intersections using dynamically generated parameters.

For the techniques that include more sophisticated user training processes for handling raster maps with poor image quality, Salvatore and Guitton [2004] use a color extraction method as their first step to extract contour lines from topographic maps. Khotanzad and Zink [2003] utilize a color segmentation method with user annotations to extract the contour lines from the USGS topographic maps. Chen et al. [2006b] exploit the color segmentation method of Khotanzad and Zink [2003] to handle common topographic maps (i.e., not limited to the USGS topographic maps) using local segmentation techniques. These techniques with user training are generally able to handle maps that are more complex and/or have poor image quality. However, their user-training process is complicated and laborious, such as manually generating a set of color thresholds for every input map [Salvatore and Guitton, 2004] and labeling all combinations of line and background colors [Khotanzad and Zink, 2003]. In comparison, my supervised approach for separating the road layers requires the user to provide a few labels for road areas, which is simpler and more straightforward.

In addition to the research work, a commercial product called R2V from Able Software is an automated raster-to-vector conversion software package specialized for digitizing

raster maps. To vectorize roads in raster maps using R2V, the user needs to first provide labels of road colors or select one set of color threshold to identify the road pixels. The manual work of providing labels of only road pixels can be laborious, especially for scanned maps with numerous colors, and the color thresholding function does not work if one set of thresholds cannot separate all the road pixels from the other pixels. In comparison, I automatically identify road colors from a few user labels for extracting the road pixels. After the road pixels are extracted, R2V can automatically trace the centerlines of the extracted road pixels and generate the road vector data. However, R2V's centerline-tracing function is sensitive to the road width without manual pre-processing, which produces small branches on a straight line when the road is wide. I detect the road width automatically and use the detected road information to generate parameters for identifying accurate road centerlines. In my experiments, I tested R2V using a set of test maps and show that my road vectorization approach generated better results with considerably less user input.

5.4 Text Layer Separation and Text Recognition

Text recognition from raster maps is a difficult task since the text labels often overlap with other features in the maps, such as road lines, and the text labels do not follow a fixed orientation within a map [Nagy et al., 1997]. Raveaux et al. [2008] work on a set of color cadastral maps to separate the quarter (a set of parcels) and text layers from the maps. They first apply an anti-fading algorithm on the maps for restoring the faded color in the historical maps. Then, they use the expectation maximization algorithm to select a color

space from a set of color components (e.g., the red, green, and blur or hue, saturation, and lightness color components) for processing the map. Finally, they utilize an edge detector to detect the connected components and use a genetic algorithm [Raveaux et al., 2007] to classify the connected components into the text and quarter layers. They do not further group the characters in the text layer into text strings and recognize the strings.

For the techniques that work on a specific type of map or character, Fletcher and Kasturi [1988] and Bixler [2000] assume that the line and character pixels are not overlapping, and the assumption does not apply on raster maps in general. Chen and Wang [1997] utilize the Hough transformation to group nearby numeric letters for identifying straight numeric strings in raster maps. Then, they use a set of font and size independent features to recognize the numeric strings. The font and size independent features cannot apply on characters other than the numeric characters.

One line of research builds specific character recognition components for handling multi-oriented text strings [Adam et al., 2000; Deseilligny et al., 1995]. Deseilligny et al. [1995] use rotation-invariant features and Adam et al. [2000] use image features based on the Fourier-Mellin Transformation to compare the target characters with the trained character samples for recognizing text labels in maps. Pezeshk and Tutwiler [2010] produce artificial training data (i.e., a set of character images) from a set of initial training data using their Truncated Degradation Model and then recognize the map characters by using two Hidden Markov Models with 10 states trained for each character class. The Truncated Degradation Model helps to reduce the manual effort of producing the training data but for each font type, the user still has to generate a set of character images as the initial training data. These methods [Adam et al., 2000; Deseilligny et al.,

1995; Pezeshk and Tutwiler, 2010] require training for each input map, such as providing sample characters for maps using different fonts to generate distinct feature sets for the classification.

For a more general text recognition techniques, Li et al. [2000] identify the graphics layer and text labels using connected-component analysis and extrapolate the graphics layer to remove the lines that overlap with characters within each identified text label. Then, a template-matching based optical character recognition (OCR) component is used to recognize characters from the text labels. However, this extrapolation method does not apply to curved labels.

Cao and Tan [2002] analyze the geometric properties of the connected component to first separate text labels from graphics (i.e., linear objects). Then, they decompose the separated graphics into line segments and a size filter is used to recover the character strokes that touch the lines. Finally, a commercial OCR product from HP is used to recognize the text labels. Their method does not provide a solution for automatic processing non-horizontal text labels.

Velázquez and Levachkine [2004] utilize the V-Lines and the rectangle growing approaches to separate the text labels from graphics and then recognize the text labels with Artificial Neural Networks (ANN). Pouderoux et al. [2007] use component analysis and string analysis with dynamic parameters generated from the geometry of the connected components to identify strings in the raster maps, which does not consider overlapping labels that commonly exist in maps. They then render the identified strings horizontally for character recognition using the average angle connecting the centroid points of the components in a string, which can be inaccurate when the characters have very different

heights or widths. In my approach, I use a robust skew detection method derived from a morphological-operator-based skew detection technique [Najman, 2004] to identify the orientation of each string automatically.

Although these text recognition techniques [Cao and Tan, 2002; Li et al., 2000; Pouderoux et al., 2007; Velázquez and Levachkine, 2004] assume a more general type of raster map, they all require a manually prepared foreground image of the maps as the input of their text recognition algorithms. However, raster maps usually suffer from compression and scanning noise and hence contain numerous colors, which means the preprocessing step requires tedious manual work. In comparison, my supervised technique, which utilizes color segmentation methods to separate the text layers from raster maps, requires only minimal user input.

Other techniques include additional information for identifying areas of text labels in the raster map. Gelbukh et al. [2004] extend the algorithm from Velázquez and Levachkine [2004] by exploiting additional information from a toponym database, linguistic dictionaries, and the spatial distributions of various font sizes in a map to detect and correct the errors introduced in the processes of text recognition. Myers et al. [1996] generate hypotheses of the characters in the text labels and the locations of the text labels using a gazetteer and identify text zones in the raster maps by detecting and grouping short length patterns in the horizontal and vertical directions. The identified text zones are then verified with the hypotheses generated from the gazetteer using a keyword recognition technique that detects the presence of possible text labels based on whole text shape in the text zone. The auxiliary information (e.g., a gazetteer) used in these techniques is usually not easily accessible compared to the raster maps.

Chapter 6

Conclusion and Future Extensions

In this chapter, I first recapitulate the contributions of this thesis and then summarize the heuristics that this thesis exploits for processing the raster maps. The heuristics sum up the capability of handling diverse types of maps with the techniques described in this thesis. Finally, I identify potential future research directions and conclude.

6.1 Contributions

This thesis offers a general approach that requires minimal human effort for harvesting geographic features from heterogeneous raster maps. The approach extracts feature layers of road lines and text labels and detects road-intersection templates for map alignment, generates and vectorizes road geometry for road vector data, and recognizes text labels for map context.

In Chapter 2, 3, and 4, I presented a general approach for separating the road and text layers from heterogeneous raster maps with minimal user input. This approach automatically separates the feature layers from raster maps with good image quality

(Chapter 2) and handles raster maps with poor image quality or complex maps using supervised techniques that require minimal user input (Chapter 3 and 4).

In Chapter 3, I presented an automatic approach for extracting accurate road vector data from the separated road layers automatically. This approach exploits the geometric properties of road lines to automatically generate the road geometry and correct possible distortions in the road geometry. I showed that my approach extracts accurate road vector data from 16 raster maps of 11 sources with varying color usages and image quality.

In Chapter 4, I presented an automatic approach for recognizing text labels in the separated text layers automatically. My text recognition approach focuses on locating individual strings in the text layer and detecting the string orientations to then leverage the horizontal text recognition capability of commercial OCR software. By doing so, my approach requires no training on character recognition and benefits from future improvements on commercial OCR software. My experiments on 15 maps from 10 sources showed accurate text recognition results from heterogeneous raster maps.

To enable the processing of large raster maps, the automatic road vectorization and text recognition techniques both employ a divide-and-conquer approach that divides the input map into tiles, processes each tile, and merges the results from individual tiles. Moreover, since the processing of each divided tile is a independent task, the divide-and-conquer approach benefits from the multi-threading implementation and multi-processor platform for improving the overall efficiency.

6.2 Map Processing Heuristics

A major advantage of the work in this thesis over the related work is that my techniques are not limited to a specific type of raster map and do not rely on prior knowledge of the input map. Although maps contain unstructured features compared to classic document images, such as scanned book pages, the road lines and text labels across different raster maps share several visual and geometric properties.

In this thesis, I presented automatic techniques that detect the common properties of the feature layers of road lines and text labels and then exploit the properties for processing the raster maps automatically. In particular, I presented general automatic techniques to separate the road and text layers from the maps and then to automatically extract road-intersection templates, vectorize road geometry, and recognize text labels. I showed accurate feature recognition results testing the general techniques on a variety of maps from various sources. The tested maps include an array of scanned and computer generated maps with varying map complexity and image quality. The common properties of the feature layers, road lines, and text labels across raster maps are as follows:

- The background luminosity levels have a dominant number of pixels.
- The number of foreground pixels is significantly smaller than the number of background pixels.
- The foreground luminosity levels have high contrast against the background luminosity levels.

- The majority of the roads share the same road format: single-line or double-line roads.
- The majority of the roads have the same road width.
- In a double-line raster map, the linear structures (i.e., elongated lines) in single-line format are not roads, but can be other geographic feathers, such as contour lines.
- Road lines are straight within a small distance (i.e., several meters).
- Road lines are connected as the road network and hence a road layer usually has a small number of connected components or even only one large connected component when the entire road layer is connected.
- Characters are isolated, small connected components that are close to each other.
- Character strokes are generally shorter than lines that represent roads.
- Characters in a text string should have a similar size and are spatially closer than the characters in two separated strings.

The above map processing heuristics imply both the strength and limitation of the automatic techniques in this thesis. For example, if a raster map violates the assumption that the background luminosity levels have a dominant number of pixels, the automatic map decomposition technique cannot work and hence we would need to use the supervised layer separation techniques.

6.3 Future Extensions

The georeferenced and machine-editable geographic features extracted from raster maps offer many possibilities for future research. In this section, I point out possible future directions based on the research in this thesis.

With the extracted road vector data, we can utilize map conflation techniques [Chen et al., 2008; Wu et al., 2007] to identify the geospatial extent of the raster map. We can then include automatic post-processing based on the geospatial extent of the map to discover the real world lengths of the extracted road lines for improving the accuracy of the extracted road vector data (i.e., the completeness and correctness). For example, given the map extent, we can infer the resolution of the road vector data and then automatically remove unlikely road branches, such as a road segment that is smaller than one meter or road lines constituting a sharp turning angle.

For the recognized text labels, we can include additional knowledge of the map region to build a dictionary for improving the OCR accuracy. For example, we can include generic map terms for a specific region, such as "Pl" for place and "Av" for avenue in the U.S. to prevent the OCR from mis-recognizing the small strings. For a set of maps covering the same region, we can first recognize text labels in each of the map and use the mutual information to help improving the overall recognition accuracy. Moreover, a possible extension is to apply the technique to identify individual text strings in Chapter 4 to raster maps using languages other than English for recognizing the text labels in maps of foreign countries.

Given the location of the road vector data and the text labels, one interesting extension is to infer the relationship between the extracted geographic features, such as producing the road names for the roads in the extracted road vector data. This information helps to further utilize the extracted geographic features, such as planning the direction between two addresses using the road vector data with associated road names or creating a computer-generated map from the extracted geographic features.

6.4 Conclusion

This thesis presented a general approach for harvesting geographic features from raster maps, which makes raster maps a geospatial source that is understandable by machines. The general approach separates individual layers of geographic features from raster maps and recognizes the geographic features in the separated layers. In particular, the feature recognition techniques include detection of road-intersection templates, generation and vectorization of road geometry, and recognition of text labels. The general approach automatically handles raster maps with good image quality by exploiting common properties of road lines and text labels in raster maps. For complex maps or maps with poor image quality, the general approach employs a supervised technique that utilizes a few labels of road and text areas to separate the feature layers to then leverage the automatic approach for feature recognition.

The road intersections and road vector data can be used in a map conflation system to identify the geospatial extent of the raster map, georeference the map, separated feature layers, and recognized features, and align them to other geospatial data that contain roads.

We can then exploit the georefereced map, feature layers, and machine-editable map context to provide additional knowledge for viewing and understanding other geospatial data and the area covered by the map. For instance, we can create a keyword-search function for imagery by exploiting the recognized text labels from raster maps that are aligned to the imagery. Moreover, we can build gazetteers from the recognized text or generate road vector data for the areas where such data are unavailable.

Bibliography

Ablameyko, S., Beregov, B., and Kryuchkov, A. (1993a). Computer-aided cartographical system for map digitizing. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 115–118.

Ablameyko, S., Beregov, B., and Kryuchkov, A. (1994). Automatic map digitising: problems and solution. *Computing Control Engineering Journal*, 5(1):33 –39.

Ablameyko, S., Bereishik, V., Frantskevich, O., Homenko, M., Melnik, E., Okun, O., and Paramonova, N. (1993b). A system for automatic vectorization and interpretation of map-drawings. In *Proceedings of the Fourth International Conference on Computer Vision*, pages 456–460.

Ablameyko, S., Bereishik, V., Frantskevich, O., Homenko, M., and Paramonova, N. (1998). A system for automatic recognition of engineering drawing entities. In *Proceedings of the Fourteenth International Conference on Pattern Recognition*, volume 2, pages 1157–1159.

Ablameyko, S., Bereishik, V., Homenko, M., Lagunovsky, D., Paramonova, N., and Patsko, O. (2002a). A complete system for interpretation of color maps. *International Journal of Image and Graphics*, 2(3):453–479.

Ablameyko, S., Bereishik, V., Homenko, M., Paramonova, N., and Patsko, O. (2002b). Automatic/interactive interpretation of color map images. In *Proceedings of the 16th International Conference on Pattern Recognition*, volume 3, pages 69–72.

Ablameyko, S., Beveisbik, V., Homenko, M., Paramonova, N., and Patsko, O. (2001). Interpretation of colour maps. a combination of automatic and interactive techniques. *Computing Control Engineering Journal*, 12(4):188–196.

Ablameyko, S. and Frantskevich, O. (1992). Knowledge based technique for map-drawing interpretation. In *Proceedings of the International Conference on Image Processing and its Applications*, pages 550–554.

Adam, S., Ogier, J., Cariou, C., Mullot, R., Labiche, J., and Gardes, J. (2000). Symbol and character recognition: application to engineering drawings. *International Journal on Document Analysis and Recognition*, 3(2):89–101.

Agam, G. and Dinstein, I. (1996). Generalized morphological operators applied to map-analysis. In *Proceedings of the 6th International Workshop on Advances in Structural and Syntactical Pattern Recognition*, pages 60–69.

Arrighi, P. and Soille, P. (1999). From scanned topographic maps to digital elevation models. In *Proceedings of the International Symposium on Imaging Applications in Geology*.

Bin, D. and Cheong, W. K. (1998). A system for automatic extraction of road network from maps. In *Proceedings of the IEEE International Joint Symposia on Intelligence and Systems*, pages 359–366.

Bixler, J. P. (2000). Tracking text in mixed-mode documents. In *Proceedings of the ACM Conference on Document Processing Systems*, pages 177–185.

Bucha, V., Uchida, S., and Ablameyko, S. (2006). Interactive road extraction with pixel force fields. In *Proceeeding of the 18th International Conference on Pattern Recognition*, volume 4, pages 829 –832.

Bucha, V., Uchida, S., and Ablameyko, S. (2007). Image pixel force fields and their application for color map vectorisation. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition*, volume 2, pages 1228 –1242.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698.

Cao, R. and Tan, C. L. (2002). Text/graphics separation in maps. In *Proceedings of the Fourth IAPR International Workshop on Graphics Recognition Algorithms and Applications*, pages 167–177.

Chen, C.-C., Knoblock, C. A., and Shahabi, C. (2006a). Automatically conflating road vector data with orthoimagery. *GeoInformatica*, 10(4):495–530.

Chen, C.-C., Knoblock, C. A., and Shahabi, C. (2008). Automatically and accurately conflating raster maps with orthoimagery. *GeoInformatica*, 12(3):377–410.

Chen, C.-C., Knoblock, C. A., Shahabi, C., Chiang, Y.-Y., and Thakkar, S. (November, 2004a). Automatically and accurately conflating orthoimagery and street maps. In *The 12th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS'04)*, Washington, D.C., USA.

Chen, C.-C., Shahabi, C., and Knoblock, C. A. (2004b). Utilizing road network data for automatic identification of road intersections from high resolution color orthoimagery. In *The 2nd Workshop on Spatio-Temporal Database Management - STDBM'04*.

Chen, L.-H. and Wang, J.-Y. (1997). A system for extracting and recognizing numeral strings on maps. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 1, pages 337 –341.

Chen, Y., Wang, R., and Qian, J. (2006b). Extracting contour lines from common-conditioned topographic maps. *IEEE Transactions on Geoscience and Remote Sensing*, 44(4):1048–1057.

Chiang, Y.-Y. and Knoblock, C. A. (2008). Automatic extraction of road intersection position, connectivity, and orientations from raster maps. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10.

Chiang, Y.-Y. and Knoblock, C. A. (2009a). Automatic road vectorization of raster maps. In *Proceedings of the Eighth IAPR International Workshop on Graphics Recognition*, pages 27–28.

Chiang, Y.-Y. and Knoblock, C. A. (2009b). Extracting road vector data from raster maps. In *Graphics Recognition: Achievements, Challenges, and Evolution, Selected Papers of the 8th International Workshop on Graphics Recognition (GREC), Lecture Notes in Computer Science, 6020*, pages 93–105, New York. Springer.

Chiang, Y.-Y. and Knoblock, C. A. (2009c). A method for automatically extracting road layers from raster maps. In *Proceedings of the Tenth International Conference on Document Analysis and Recognition*.

Chiang, Y.-Y. and Knoblock, C. A. (2010). An approach for recognizing text labels in raster maps. In *Proceedings of the 20th International Conference on Pattern Recognition*.

Chiang, Y.-Y., Knoblock, C. A., and Chen, C.-C. (2005). Automatic extraction of road intersections from raster maps. In *Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems*, pages 267–276.

Chiang, Y.-Y., Knoblock, C. A., Shahabi, C., and Chen, C.-C. (2008). Automatic and accurate extraction of road intersections from raster maps. *GeoInformatica*, 13(2):121–157.

Cofer, R. H. and Tou, J. T. (1972). Automated map reading and analysis by computer. In *Proceedings of the AFIPS Joint Computer Conferences, Part I*, pages 135–145.

Comaniciu, D. and Meer, P. (2002). Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619.

Desai, S., Knoblock, C. A., Chiang, Y.-Y., Desai, K., and Chen, C.-C. (2005). Automatically identifying and georeferencing street maps on the web. In *Proceedings of the 2nd International Workshop on Geographic Information Retrieval*, pages 35–38.

Deseilligny, M. P., Mena, H. L., and Stamonb, G. (1995). Character string recognition on maps, a rotation-invariant recognition method character string recognition on maps, a rotation-invariant recognition method. *Pattern Recognition Letters*, 16(12):1297–1310.

Dhar, D. B. and Chanda, B. (2006). Extraction and recognition of geographical features from paper maps. *International Journal of Document Analysis and Recognition*, 8(4):232–245.

Fletcher, L. A. and Kasturi, R. (1988). A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):910–918.

Forsyth, D. A. and Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference.

Gelbukh, A., Levachkine, S., and Han, S.-Y. (2004). Resolving ambiguities in toponym recognition in cartographic maps. In *Proceedings of the 5th IAPR International Workshop on Graphics Recognition*, pages 104–112.

Goldberg, D. W., Wilson, J. P., and Knoblock, C. A. (2009). Extracting geographic features from the internet to automatically build detailed regional gazetteers. *International Journal of Geographic Information Science*, 23(1):92–128.

Habib, A., Uebbing, R., and Asmamaw, A. (1999). Automatic extraction of road intersections from raster maps. Technical report, Center for Mapping, The Ohio State University.

Heckbert, P. S. (1982). Color image quantization for frame buffer display. *Computer Graphics*, 16:297–307.

Heipke, C., Mayer, H., Wiedemann, C., and Jamet, O. (1997). Evaluation of automatic road extraction. In *International Archives of Photogrammetry and Remote Sensing*, pages 47–56.

Henderson, T. C., Linton, T., Potupchik, S., and Ostanin, A. (2009). Automatic segmentation of semantic classes in raster map images. In *Proceedings of the Eighth IAPR International Workshop on Graphics Recognition*, pages 253–262.

Itonaga, W., Matsuda, I., Yoneyama, N., and Ito, S. (2003). Automatic extraction of road networks from map images. *Electronics and Communications in Japan (Part II: Electronics)*, 86(4):62–72.

Kanai, J., Rice, S. V., Nartker, T. A., and Nagy, G. (1995). Automated evaluation of ocr zoning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):86–90.

Kerle, N. and de Leeuw, J. (2009). Reviving legacy population maps with object-oriented image processing techniques. *IEEE Transactions on Geoscience and Remote Sensing*, 47(7):2392–2402.

Khotanzad, A. and Zink, E. (2003). Contour line and geographic feature extraction from USGS color topographical paper maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1):18–31.

Koutaki, G. and Uchimura, K. (2004). Automatic road extraction based on cross detection in suburb. In *Proceedings of the SPIE Image processing: algorithms and systems III*, volume 5299, pages 337–344.

Lacroix, V. (2009). Automatic palette identification of colored graphics. In *Graphics Recognition: Achievements, Challenges, and Evolution, Selected Papers of the 8th International Workshop on Graphics Recognition (GREC), Lecture Notes in Computer Science, 6020*, pages 95–100. Springer, New York.

Leberl, F. W. and Olson, D. (1982). Raster scanning for operational digitizing of graphical data. *Photogrammetric Engineering and Remote Sensing*, 48(4):615–672.

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707.

Leyk, S. and Boesch, R. (2010). Colors of the past: color image segmentation in historical topographic maps based on homogeneity. *GeoInformatica*, 14(1):1–21.

Li, L., Nagy, G., Samal, A., Seth, S., and Xu, Y. (1999). Cooperative text and line-art extraction from a topographic map. *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, pages 467–470.

Li, L., Nagy, G., Samal, A., Seth, S. C., and Xu, Y. (2000). Integrated text and line-art extraction from a topographic map. *International Journal of Document Analysis and Recognition*, 2(4):177–185.

Mao, S., Rosenfeld, A., and Kanungo, T. (2003). Document structure analysis algorithms: A literature survey. *Document Recognition and Retrieval X*, 5010:197–207.

MapScan (1998). *MapScan for Windows Software Package for Automatic Map Data Entry, User's Guide and Reference Manual*. Computer Software and Support for Population Activities, INT/96/P74, United Nations Statistics Division, New York, NY 10017, USA.

Michelson, M., Goel, A., and Knoblock, C. A. (2008). Identifyinig maps on the world wide web. In *Proceedings of the 5th International Conference on GIScience, LNCS 5266*, pages 249–260.

Mordohai, P. and Medioni, G. (2006). *Tensor Voting: A Perceptual Organization Approach to Computer Vision and Machine Learning*. Morgan & Claypool.

Mori, S., Suen, C. Y., and Yamamoto, K. (1995). Historical review of ocr research and development. *Document image analysis*, pages 244–273.

Myers, G. K., Mulgaonkar, P. G., Chen, C.-H., DeCurtins, J. L., and Chen, E. (1996). Verification-based approach for automated text and feature extraction from raster-scanned maps. In *Lecture Notes in Computer Science*, volume 1072, pages 190–203. Springer.

Nagy, G., Samal, A., Seth, S., Fisher, T., Guthmann, E., Kalafala, K., Li, L., Sivasubramaniam, S., and Xu, Y. (1997). Reading street names from maps - technical challenges. In *GIS/LIS conference*, pages 89–97.

Nagy, G. L., Nartker, T. A., and Rice, S. V. (2000). Optical character recognition: An illustrated guide to the frontier. In *Proceedings of the SPIE International Symposium on Electronic Imaging Science and Technology*, volume 3967, pages 58–69.

Najman, L. (2004). Using mathematical morphology for document skew estimation. In *Proceedings of the SPIE Document Recognition and Retrieval IX*, pages 182–191.

Pezeshk, A. and Tutwiler, R. (2010). Extended character defect model for recognition of text from maps. In *Proceedings of the IEEE Southwest Symposium on Image Analysis Interpretation*, pages 85–88.

Podlasov, A. and Ageenko, E. (2005). Extraction and removal of layers from map imagery data. In Kalviainen, H., Parkkinen, J., and Kaarna, A., editors, *Image Analysis*, volume 3540 of *Lecture Notes in Computer Science*, pages 1107–1116. Springer.

Pouderoux, J., Gonzato, J. C., Pereira, A., and Guitton, P. (2007). Toponym recognition in scanned color topographic maps. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition*, volume 1, pages 531–535.

Pratt, W. K. (2001). *Digital Image Processing: PIKS Scientific Inside*. Wiley-Interscience, 3rd edition.

Raveaux, R., Barbu, E., Locteau, H., Adam, S., Héroux, P., and Trupin, E. (2007). A graph classification approach using a multi-objective genetic algorithm application to symbol recognition. In Escolano, F. and Vento, M., editors, *Graph-Based Representations in Pattern Recognition*, volume 4538 of *Lecture Notes in Computer Science*, pages 361–370. Springer.

Raveaux, R., Burie, J.-C., and Ogier, J.-M. (2008). Object extraction from colour cadastral maps. In *Proceedings of the IAPR International Workshop on Document Analysis Systems*, volume 0, pages 506–514.

Rice, S. V., Jenkins, F. R., and Nartker, T. A. (1996). The fifth annual test of ocr accuracy. Technical report, University of Nevada, Las Vegas.

Saalfeld, A. J. (1993). *Conflation: automated map compilation*. PhD thesis, University of Maryland at College Park, College Park, MD, USA.

Salvatore, S. and Guitton, P. (2004). Contour line recognition from scanned topographic maps. In *Proceedings of the Winter School of Computer Graphics*, pages 1–3.

Samet, H. and Soffer, A. (1994). A legend-driven geographic symbol recognition system. In *Proceedings of the 12th International Conference on Pattern Recognition*, volume 2, pages 350–355.

Samet, H. and Soffer, A. (1996). Maco: Map retrieval by content. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):783–798.

Sezgin, M. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–168.

Shi, J. and Tomasi, C. (1994). Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600.

Suzuki, S. and Yamada, T. (1990). Maris: map recognition input system. *Pattern Recognition*, 23(8):919–933.

Tang, Y. Y., Lee, S.-W., and Suen, C. Y. (1996). Automatic document processing: A survey. *Pattern Recognition*, 29(12):1931–1952.

Velázquez, A. and Levachkine, S. (2004). Text/graphics separation and recognition in raster-scanned color cartographic maps. In Lladós, J. and Kwon, Y.-B., editors, *Graphics Recognition*, volume 3088 of *Lecture Notes in Computer Science*, pages 63–74. Springer.

von Ahn, L., Blum, M., Hopper, N. J., and Langford, J. (2003). Captcha: Using hard ai problems for security. In *Advances in Cryptology, Eurocrypt*, pages 294–311.

Wong, K. Y. and Wahl, F. M. (1982). Document analysis system. *IBM Journal of Research and Development*, 26:647–656.

Wu, X., Carceroni, R., Fang, H., Zelinka, S., and Kirmse, A. (2007). Automatic alignment of large-scale aerial rasters to road-maps. In *Proceedings of the 15th ACM International Symposium on Advances in geographic information systems*, pages 1–8.

Zack, G., Rogers, W., and Latt, S. (1977). Automatic measurement of sister chromatid exchange frequency. *Journal of Histochemistry and Cytochemistry*, 25(7):741–753.

Ziou, D. and Tabbone, S. (1998). Edge detection techniques - an overview. *International Journal of Pattern Recognition and Image Analysis*, 8:537–559.