Noname manuscript No.
(will be inserted by the editor)

General Approach for Extracting Road Vector Data from Raster Maps

Yao-Yi Chiang · Craig A. Knoblock

Received: date / Accepted: date

Abstract Raster maps are easily accessible and contain rich road information; however, converting the road information to vector format is challenging because of varying image quality, overlapping features, and typical lack of metadata (e.g., map geocoordinates). Previous road vectorization approaches for raster maps typically handle a specific map series and require significant user effort. In this paper, we present a general road vectorization approach that exploits common geometric properties of roads in maps for processing heterogeneous raster maps while requiring minimal user intervention. In our experiments, we compared our approach to a widely-used commercial product using 40 raster maps from 11 sources. We showed that overall our approach generated high quality results with low redundancy with considerably less user input compared to competing approaches.

Keywords GIS · raster maps · road vectorization · map processing

1 Introduction

For centuries, cartographers have been producing maps, which contain valuable geospatial information, such as road lines, text labels, building locations, and contour lines. Because of the availability of low cost and high-resolution scanners and the Internet, we can now obtain a huge number of maps in raster format from various sources. For instance, a digital raster graphic (DRG), which is a georeferenced scanned image of a United States Geological Survey (USGS) topographic map, can be purchased from the USGS website or accessed freely from TerraServer-USA.[1] Other map sources, such as online map repositories like the University of Texas Map Library,[2] also provide information-rich maps and historical maps for many countries. Websites such as OpenStreetMap[3] and MultiMap[4] provide high quality computer-generated maps produced directly from vector data with valuable geospatial information, such as business locations.

Because maps commonly contain road networks, raster maps are an important source of road information (e.g., road geometry), which is especially valuable for areas where road vector data are not readily available. Moreover, since the road networks exist across various geospatial data sources (e.g., satellite imagery), the road topology (e.g., road connectivity) and road geometry extracted from a raster map can be used as matching features to align the map and recognized map features to other geospatial data that contain roads [Chen et al., 2008; Wu et al., 2007].

Converting the roads in heterogeneous raster maps to vector format is challenging for a number of reasons: first, the access to metadata about the maps (e.g., map geocoordinates) is often not available, which makes it difficult to obtain prior knowledge about the region for processing the maps. Second, maps typically contain

Yao-Yi Chiang
University of Southern California, Information Sciences Institute and Spatial Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, USA
E-mail: yaoyichi@isi.edu

Craig A. Knoblock
University of Southern California, Department of Computer Science and Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, USA
E-mail: knoblock@isi.edu

[1] http://terraserver-usa.com/
[2] http://www.lib.utexas.edu/maps/
[3] http://www.openstreetmap.org/
[4] http://www.multimap.com/

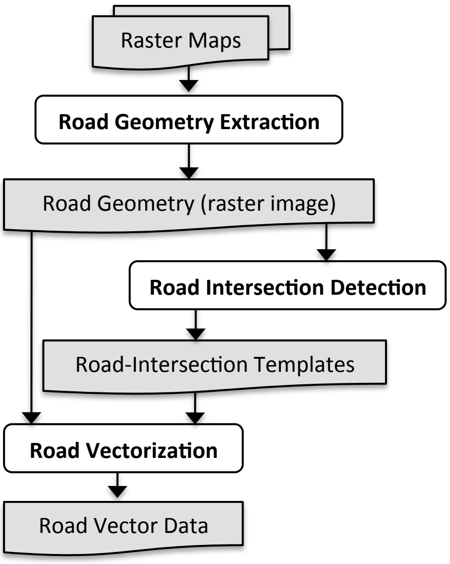

Raster Maps
Road Geometry Extraction
Road Geometry (raster image)
Road Intersection Detection
Road-Intersection Templates
Road Vectorization
Road Vector Data

Fig. 1 The overall approach for extracting road vector data from heterogeneous raster maps

overlapping layers of geographic features, such as roads, contour lines, and labels. Thus, the map content is multilayered and highly complex. Third, the image quality of raster maps is sometimes poor due to the scanning and/or image compression processes for creating the maps in raster format.

In this paper we present an end-to-end approach to extracting accurate road vector data from full-size raster maps with minimal user input. Figure 1 shows the three major steps of our approach: **(1) Road Geometry Extraction**, **(2) Road Intersection Detection**, and **(3) Road Vectorization**. These three steps build on the map processing work in our earlier papers, which solved specific subproblems (road intersection detection [Chiang et al., 2008], road-intersection-template extraction [Chiang and Knoblock, 2008], road layer extraction [Chiang and Knoblock, 2009a], and road vectorization [Chiang and Knoblock, 2009b]) of the overall approach.

Beyond the overall integrated approach, this paper makes a number of additional contributions. First, we present the complete algorithms for the road layer extraction [Chiang and Knoblock, 2009a], road-intersection-template extraction [Chiang and Knoblock, 2008], and road vectorization [Chiang and Knoblock, 2009b] (Sections 3, 4, and 5). Second, we present fast and scalable algorithms for generating road geometry from raster maps that have numerous colors (Section 3.1) and large image sizes (Section 3.3). Third, we present an approach to processing full-size maps in the integrated vectorization process (Section 5.2). Fourth, we evaluate our integrated approach to road vectorization and present experimental results on a variety of maps from diverse sources with varying image quality and compare our approach with a commercial product (Section 6).

The remainder of this paper is organized as follows. Section 2 discusses the related work. Sections 3 to 5 present the three major steps of our overall approach: road geometry extraction, road intersection detection, and road vectorization, respectively. Section 6 reports on our experimental results, and Section 7 presents the conclusion and future work.

2 Related Work

In this section, we first review the related work on segmenting color maps into layers of geographic features. Then we review the related research on road vectorization from raster maps and commercial products for raster-to-vector conversion.

2.1 Color Image Segmentation for Raster Maps

Leyk and Boesch [2010] present a color image segmentation technique that handles a series of scanned historical maps (Siegfried Maps) by considering the image plane, frequency domain, and color space. This color image segmentation technique has only been tested on maps with similar conditions (e.g., with the same set of map symbols) and may not work on heterogeneous raster maps. In our color segmentation process, our approach handles heterogeneous map types by including an interactive step for a user to select the quantized image that best represents the segmented map.

Lacroix [2009] presents the *median-shift* technique, which extracts the color palette (i.e., a small set of representative colors) from a raster map. This technique requires a preprocessing step based on the automatic edge detection, which is prone to color noise. In contrast, our approach does not rely on the edge detection and handles raster maps with poor scan quality.

Henderson et al. [2009] focus on USGS topographic maps to separate individual thematic layers from the maps. Their technique is based on the color key (i.e., the colors of individual feature layers in the map) that comes with a series of USGS topographic maps. Our color segmentation approach handles a variety of raster maps and does not require the knowledge of the map color-key, which is generally unavailable for scanned maps.

2.2 Research on Road Vectorization from Raster Maps

Much research work has been performed in the field of extracting graphic features from raster maps, such as separating lines from text [Cao and Tan, 2002; Li et al., 2000], detecting road intersections [Chiang et al., 2008; Habib et al., 1999], extracting road vector data [Bin and Cheong, 1998; Itonaga et al., 2003], and recognizing contour lines [Chen et al., 2006; Khotanzad and Zink, 2003] from raster maps.

One line of research on graphic extraction techniques uses simple techniques to extract the foreground pixels 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 extract 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. They do not process the road pixels further to generate the road geometry and extract the road vector data. 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 and to produce accurate geometry of road intersections.

Habib et al. [1999] focus on raster maps that contain only road lines to extract road intersections automatically. Their road intersection extraction technique detects the corner points on the extracted road edges and then groups the corner points and identifies 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.

Itonaga et al. [2003] focus on computer-generated raster maps that contain only road and background areas. They exploit the geometric properties of roads (e.g., elongated polygons) to first label each map area as either a road or background area. Then they apply the thinning operator to extract a 1-pixel width road network from the identified road areas. The distortion at a road intersection caused by the thinning operator is corrected by merging the intersecting lines with a similar orientation, which requires user-specified constraints, such as the maximum deviation between two intersecting lines and the maximum intersecting angle. Their approach cannot handle scanned maps and they do not report evaluation results on road vector data.

In our earlier work on road geometry extraction [Chiang et al., 2008], we developed an automatic technique that utilizes a grayscale-histogram-analysis method to automatically separate the foreground pixels from raster maps and then identify the road intersections. The histogram analysis method does not handle scanned maps well since the noise introduced in the scanning process is sometimes difficult to remove automatically. In addition, since we use the thinning operator in our previous work [Chiang et al., 2008], the extracted road intersections are not accurate when the roads are wide.

In comparison to the previous work that handles specific types of maps [Bin and Cheong, 1998; Cao and Tan, 2002; Habib et al., 1999; Itonaga et al., 2003; Li et al., 2000] and our previous approach [Chiang et al., 2008], this paper presents a semi-automatic approach, which includes user training and is capable of handling diverse types of maps, especially scanned maps. Moreover, we automatically generate accurate road geometry by detecting and correcting the distorted lines around road intersections caused by the thinning operator to handle roads that are wide.

Previous work has developed 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 USGS topographic maps. Chen et al. [2006] exploit the color segmentation method of Khotanzad and Zink [2003] to handle common topographic maps (i.e., not limited to 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 processes are 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, our semi-automatic approach for extracting road geometry requires the user to provide a few labels for road areas, which is simpler and more straightforward.

2.3 Commercial Products for Raster-to-Vector Conversion

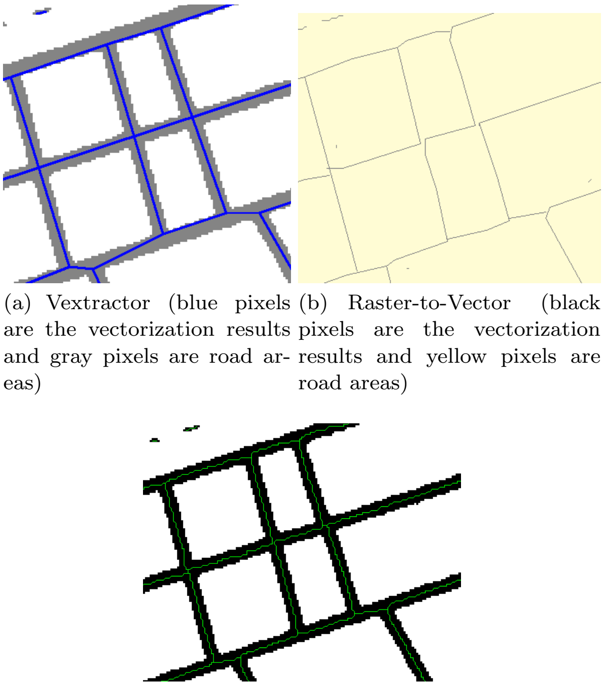
Many commercial products offer the functionality for raster-to-vector conversion, such as Adobe Illustrator,[5]

[5] http://www.adobe.com/products/illustrator.html

CorelDraw Graphics Suite,[6] and VectorMagic.[7] These commercial products are designed to generate vectorized boundaries of homogenous color areas from input images. For road vectorization from raster maps, we want to extract the centerlines of road areas and these products do not support this capability.

There are commercial products that offer centerline vectorization functionality. Vextractor[8] and Raster-to-Vector[9] use line approximation algorithms that generate vector lines based on the locations of the pixels inside road areas. The line approximation algorithms do not take into consideration the geometry and topology of the road network and hence the resulting road vector data are off-center, such as the Vextractor results shown in Figure 2(a), or have inaccurate road topology (i.e., two lines that are connected at an intersection in the road layer are not necessarily connected in the road vector data), such as the results shown in Figure 2(b). Another commercial product called R2V from Able Software[10] is an automated raster-to-vector conversion software package specialized in digitizing raster maps. R2V is specifically designed for raster-to-vector conversion from maps and can handle a variety of map specific linear features, such as curved roads and contour lines. Figure 2(c) shows more accurate results in terms of road geometry and topology compared to the results of Vectractor and Raster-to-Vector.[11]

To vectorize roads in raster maps using R2V, the user needs to first provide labels of road colors or select one set of color thresholds to identify the road pixels. The manual work of providing labels of only road pixels can be laborious in R2V, especially for scanned maps with numerous colors, and the color thresholding function does not work if one set of thresholds cannot separate all of the road pixels from the other pixels. In comparison, we 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 and produces small branches from a straight line if the line is wide. We detect the road width automatically and use the detected road information to generate parameters for identifying accurate road centerlines. In our experiments, we tested R2V using our test maps and show that our approach generates better results.



(a) Vextractor (blue pixels are the vectorization results and gray pixels are road areas)

(b) Raster-to-Vector (black pixels are the vectorization results and yellow pixels are road areas)

(c) R2V (green pixels are the vectorization results and black pixels are road areas)

Fig. 2 Sample screenshots of the road vectorization results from commercial products

3 First Step: Road Geometry Extraction

In this section, we present our technique for generating the road geometry from raster maps. This technique can handle raster maps with poor image quality, such as scanned maps. We first present a supervised approach for extracting road pixels from raster maps. Next, we describe how we exploit our previous work [Chiang et al., 2008] to identify the road centerlines from the extracted road pixels automatically. Finally, since scanned maps are usually large images (a typical 350 dot-per-inch (DPI) scanned map can be larger than 6000x6000 pixels), we present an fast algorithm for detecting the road width and road format of large maps efficiently. This algorithm is an enhanced version of the Parallel-Pattern-Tracing algorithm from our previous work [Chiang et al., 2008]

3.1 Supervised Extraction of Road Pixels

There are three major steps for the supervised extraction of road pixels from raster maps. The first step is

[6] http://www.corel.com/

[7] http://vectormagic.com/home

[8] http://www.vextrasoft.com/vextractor.htm

[9] http://www.raster-vector.com/

[10] http://www.ablesw.com/r2v/

[11] In these examples, we gave the three commercial products the same input image and we used the automatic vectorization function of each product to generate the sample results.

to quantize the color space of the input image. Then, a user labels road areas of every road color in the quantized image. Since the quantized image has a limited number of colors, we can reduce the manual effort in this user-labeling step. Finally, we automatically identify a set of road colors from the user labels and generate a color filter to extract the road pixels from the raster map. For the non-road map features drawn using the same color as the roads, we will remove them in the final step to generate the road geometry (Section 3.2). We 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(a) shows a 200x200-pixel 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, we apply color quantization algorithms to group the colors of individual feature layers into clusters. Since the color variation within a feature layer is generally smaller than the variation between feature layers in a map, after applying the color quantization algorithms, we can extract individual feature layers by selecting specific color clusters.

Our color quantization step includes three algorithms: the Mean-shift [Comaniciu and Meer, 2002], the Median-cut [Heckbert, 1982], and the K-means [Lloyd, 1982] algorithms. The Mean-shift algorithm is first applied to preserve the edges of map features (e.g., road lines) while reducing noise. The Median-cut algorithm, which requires the least computation time among the three color quantization algorithms, is then applied to further quantize the image. The goal of the Median-cut algorithm is to keep image details in the quantized image, such as the image texture but the goal of our color quantization is to have a single color representing a single feature in the map (i.e., eliminating the image texture). Therefore, we apply the K-means algorithm to the result of the Median-cut algorithm to merge similar colors for removing image details. We explain each algorithm in the following paragraphs.

The Mean-shift algorithm considers the spatial relationships between colors in the image space and in the color space (i.e., the image texture) and works in a multi-dimensional space of the image coordinates, X and Y, and the HSL color space, hue, saturation, and luminous. We use the HSL color space because of the fact that hue, saturation, and luminous provide good representation of human perception [Cheng et al., 2001]. 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 a 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'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 (we use a spatial distance of 3 pixels and a color distance of 25 to define the local area). If the distance between M and N is larger than a small threshold (we use a small threshold to limit the running time for the Mean-shift algorithm to converge), the Mean-shift algorithm shifts N to 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 are used as $P(x, y)$'s color. In the example shown in Figure 3, the Mean-shift algorithm reduces the number of colors in Figure 3(a) by 72% as shown in Figure 3(b).

The results after the Mean-shift algorithm can still have many colors and we utilize the Median-cut after the Mean-shift algorithm to generate an image with at most 1,024 colors. The Median-cut algorithm first locates the minimum box which contains every color in the input image in the three dimensional HSL space. Then the algorithm sorts the colors using the color component that varies the most (i.e., the longest axis of the box) and divides the minimum box into two boxes at the median of the sorted colors. This sort-and-divide process continues to apply on the new divided boxes until the total number of boxes is smaller than the desired number of colors in the resulting quantized image. The colors in the same box of the sort-and-divide result are then represented by their median color to generate the quantized image.

To further merge similar colors in the raster maps, we apply the K-means algorithm 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 its map features merged with a small K. For example, Figure 3(c) shows the quantized map with K as 8 where 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(d) with K as 16.

The Median-cut algorithm helps to reduce the running time of the K-means algorithm by limiting the input number of colors to the K-means algorithm to 1024. The Median-cut algorithm cannot replace the K-means algorithm because the Median-cut algorithm keeps image details in the quantized image. For example, as shown in Figure 4, if we apply the K-means and Median-cut algorithms directly to the original image, the Median-cut results shows significant color variation within an image object, such as the yellow pixels on the orange roads.

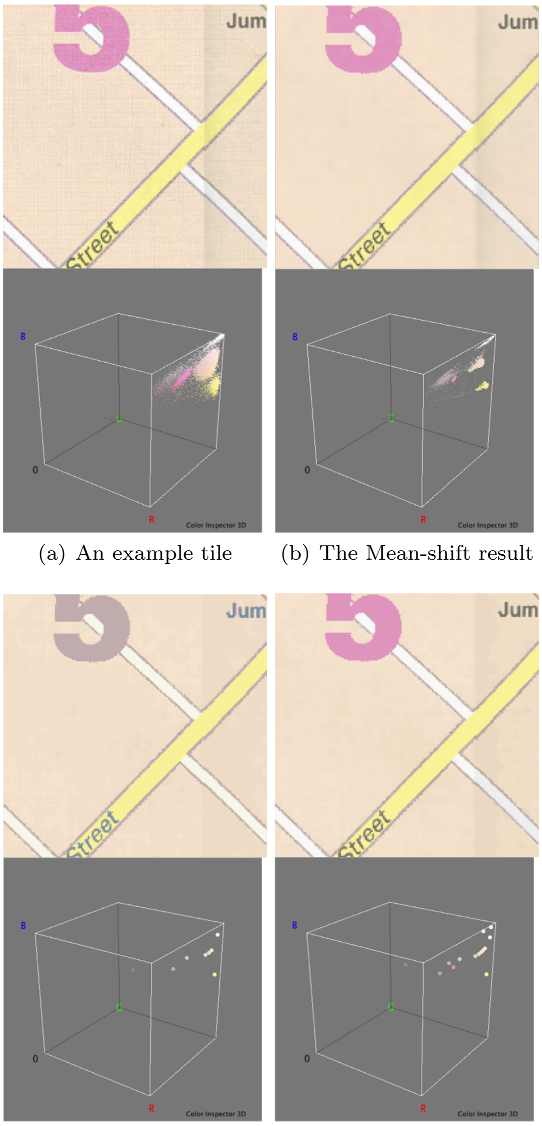
3.1.2 User Labeling

In the user-labeling step, we first generate a set of quantized maps in multiple 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. Next, 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 we exploit to identify the road colors in the next step. For example, Figure 5(a) shows an example map and Figure 5(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 and contain the two road colors in the quantized map (i.e., yellow and white) .

3.1.3 Automatic Identification of Road Colors and Extraction of Road Pixels

Each user label contains a set of colors, and some of the colors represent roads in the raster map. We exploit 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 Because the user labels are centered at a road line or a road intersection, the pixels of a road color are a portion of one or more road lines that pass through or nearby the image center. For example, we first separate pixels of individual colors from the user label shown in the top-right of Figure 5(b) and the result is a set of six images shown in Figure 6 (background is shown in black) and every decomposed image contains only one color from the user label. The pixels in the decomposed images, *Image 3*, *4*, and *5*, are portions of the road lines in the user label.



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

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

Fig. 3 An example map tile and the color quantization results with the color cubes

To exploit the centerline property, for each decomposed image, we first detect lines that are constituted from the connected objects in the image. This is achieved by applying the Hough transform [Duda and Hart, 1972] to identify a set of Hough lines from the skeletons of

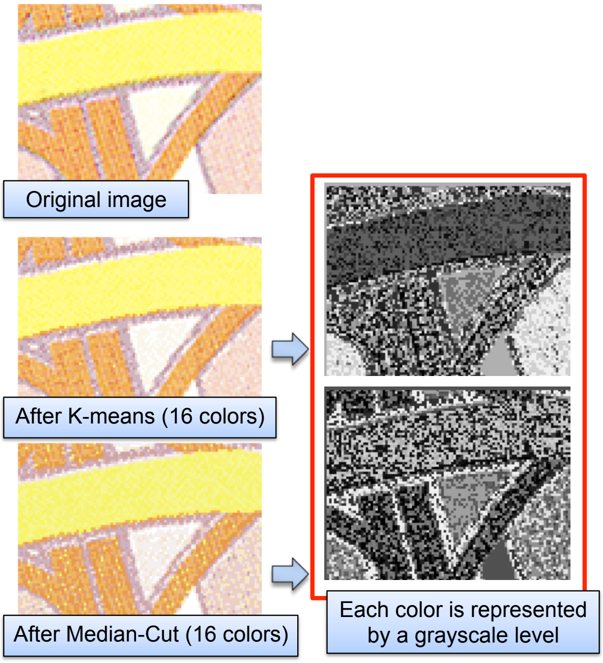
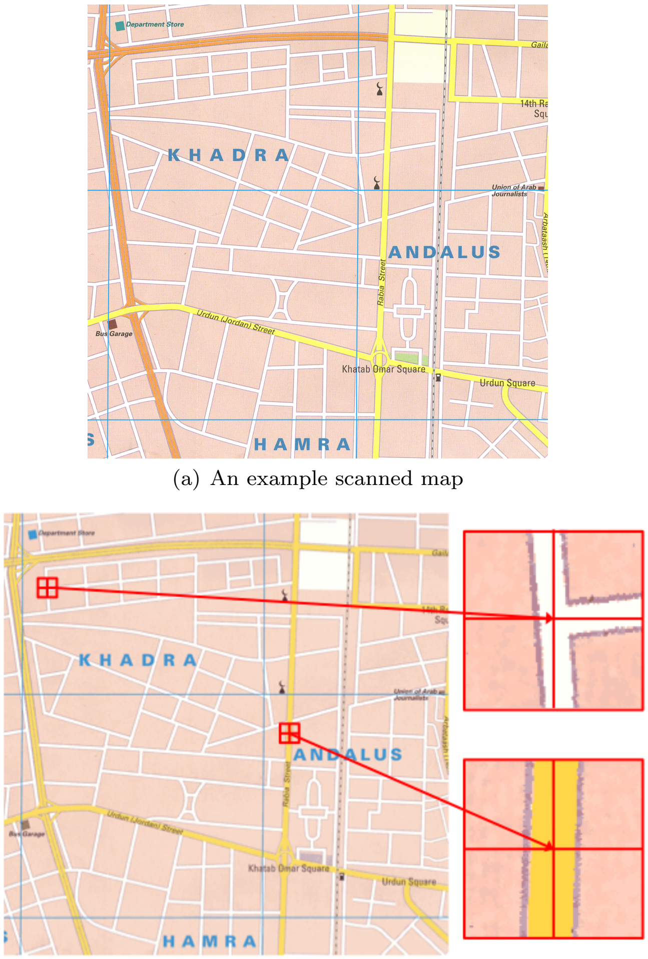

Original image
After K-means (16 colors)
After Median-Cut (16 colors)
Each color is represented by a grayscale level

Fig. 4 Median-cut algorithm result contains more image details while K-means result contains more homogenous regions

the connected objects. The Hough transform is a feature extraction technique that can identify lines (i.e., the Hough lines) from pixels that do not necessary represent every part of the lines. Since the image center of a user label is the center of a road line or a road intersection, if a Hough line is detected near the image center, the Hough line is most likely to represent a portion of the road lines. Hence, we detect the Hough lines in each decomposed image and compute the average distance between the detected Hough lines to the image center to determine if the foreground pixels (non-black pixels) in a decomposed image represent roads in the raster map.

Figure 7 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 (this distance threshold is only used to help explain the idea). In Figure 7, 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 a 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 their average distances between 1 pixel to the smallest average distance are also classified as road-pixel images. This



(a) An example scanned map

(b) The quantized map and user labels (the red boxes and crosses show the original positions and image centers of the two user labels)

Fig. 5 An example of the supervised extraction of road pixels

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 we first classify *Image 5* as a road-pixel image. Then, since *Image 4* is the only image with its average distance within a 1-pixel distance to the smallest average distance, we also classify *Image 4* as a road-pixel image.

The Neighboring Property Because the road pixels are spatially near each other in a user label, the pixels of the road colors should be spatially near each other. For example, the majority of pixels in *Image 3* can find an immediate neighboring pixels 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*.

Exploiting the centerline property for the road-pixel image classification is based on the average distance be-

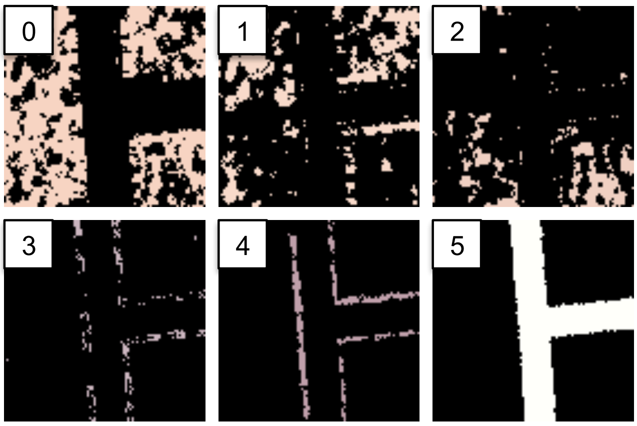


Fig. 6 The decomposed images (background shown in black), each contains one color in the user label shown in the top-right of Figure 5(b)

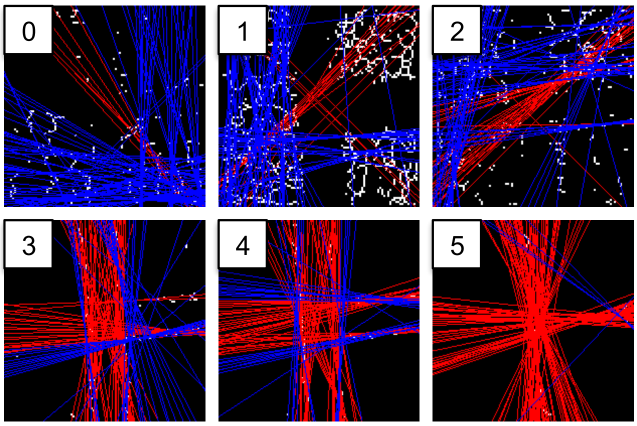
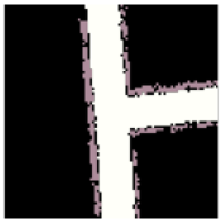


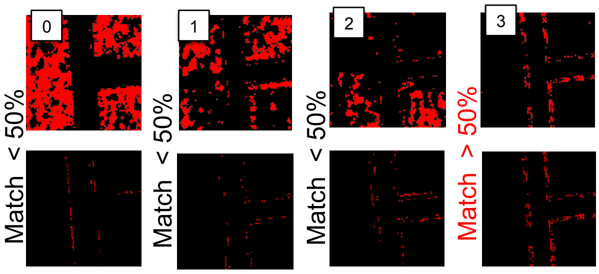
Fig. 7 The identified Hough lines using Figure 6 as the input (background shown in black)

tween the detected Hough lines to the image center. If the Hough transform detects only a few Hough lines, the Hough lines constituted from noise pixels can significantly bias the average distance and hence the image will not be classified correctly. Therefore, we present the Edge-Matching algorithm for exploiting 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 8(a) (background is shown in black). Next, we use 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, we search a 3x3-pixel neighborhood centered at (x,y) in the image of the road template to detect if there exists any road pixels. If one or more road pixels exist, we mark the pixel $C(x,y)$ as a road pixel since it is spatially near one or more road pixels. After we examine every foreground pixel in a given decomposed image, if more than 50% of the foreground pixels in that image are marked as road pixels, we classify the decomposed image as a road-pixel image.



(a) An example road template



(b) The Edge-Matching results

Fig. 8 Classifying the decomposed images using the road template

Figure 8(b) 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 we apply the Edge-Matching algorithm to 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 we classify *Image 3* as a road-pixel image and discard the others.

We process every user label and identify a set of road-pixel images from each label. We then scan the quantized map and extract the pixels that have their colors as one of the identified road colors as the road pixels. Figure 9 shows the extracted road pixels of Figure 5(a). Note that the map features that share the same color as the road lines are also extracted as the rectangular area shown in Figure 9. These features will be removed in the next step when we generate the road geometry.

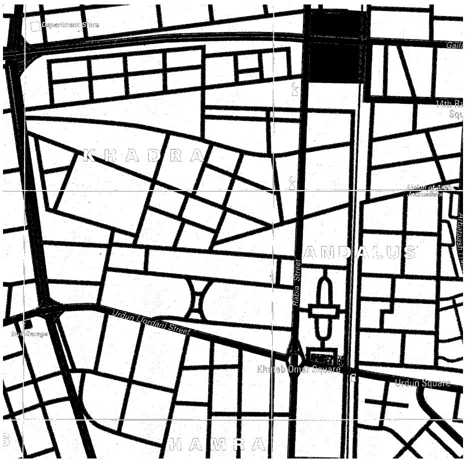


Fig. 9 The extracted road pixels of Figure 5(a)

3.2 Automatic Identification of Road Centerlines

The extracted road layer (i.e., the set of extracted road pixels) from the previous section can contain non-road map features, such as area features or text strings, if the map features are drawn using the same color as the roads. In this section, we briefly explain our previous work [Chiang et al., 2008] that we employ for removing these non-road features and automatically identify the road geometry.

Concerning the non-road map features of text strings, text/graphics separation techniques [Cao and Tan, 2002; Tombre et al., 2002] can be used to separate the text strings from the extracted road pixels (i.e., linear objects) . Our previous work uses the text/graphics separation technique from Cao and Tan [2002] since the technique was developed and tested for map processing.

To remove the non-road features other than text, reconnect broken road lines, and generate the road geometry, we first detect the road width and road format in the input map, and then we dynamically use three types of morphological operators: the erosion operator, the dilation operator, and the thinning operator (see [Pratt, 2001] for a detailed description of the morphological operators). The erosion operator is used to remove noise objects that are smaller than the road lines and to temper road areas for generating accurate road centerlines. The dilation operator is used to expand road areas for connecting broken road lines and for filling up holes in the road areas. The thinning operator is used to extract the centerlines of the roads (i.e., the skeleton of the road areas).

The numbers of iterations of these morphological operators (i.e., the number of times we apply each morphological operator) are decided dynamically based on the detected road width and road format. For example, to remove thick non-road features, if road width is 4-pixel wide, we first remove the road lines by applying the erosion operator twice with a 3-by-3 structuring element (or a 5-by-5 structuring element applied once): one iteration erodes a road line by 2 pixels; one at each side of the road. Then we apply the dilation operator to re-grow and obtain the non-road features that are thicker than road lines. By subtracting the resulting non-road features from the road layer, we remove the non-linear features such as the rectangular area in the upper-right of Figure 9.

Once the non-road features are removed, we use the dilation operator to expand the road areas and reconnect the road lines automatically. Since the expansion of the road areas should not connect two nearby roads, the number of iterations of the dilation operator with a 3-by-3 structuring element is half of the detected road width for single-line format roads. This is because we assume that two road lines should be at least a road-width apart in a map.

Concerning double-line format roads, the dilation operator not only reconnects the broken lines, but also merges parallel road lines into thick lines in single-line format. For example, by applying the dilation operator twice with a 3-by-3 structuring element, the parallel road lines that are 4-pixel apart (i.e., the detected road width is 4-pixel wide) are merged. This dilation technique fills up the areas in-between the parallel road lines if the distance between the road lines and every pixel in-between the parallel road lines is less than half of the detected road width. However, for road intersection areas, depending on how the intersections are drawn in the map, the distance between the road lines and an intersection center can be larger than the half of the detected road width. In this case, the number of iterations of the dilation operator needs to be increased for filling up the areas within road intersections of the parallel lines.

After the dilation operator, we apply the erosion operator to erode the thickened road lines. Finally, we use the thinning operator to generate the 1-pixel width road centerlines (i.e., the road geometry) from the erosion results. Figure 10 shows the extracted road geometry of Figure 9.

In this process of extracting the road geometry from raster maps, the morphological operators used for generating the road geometry cause distorted lines around road intersections. As shown in Figure 11, if we apply the thinning operator directly on the thick lines after the dilation operator shown in Figure 11(a), the lines that are near intersections are significantly distorted as shown in Figure 11(b). Our approach to reduce the extent of the line distortion is to erode the lines using the

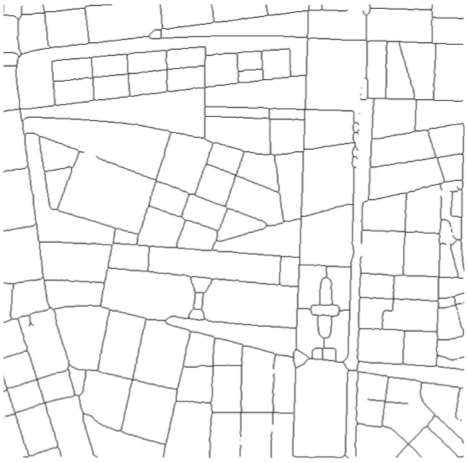
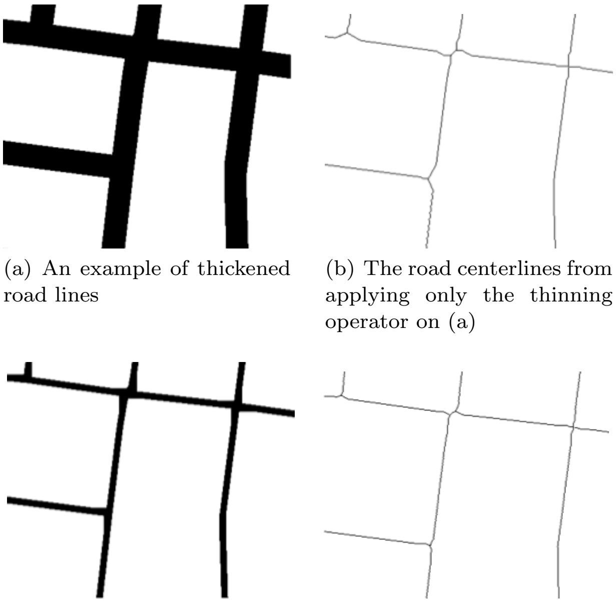


Fig. 10 The extracted road geometry of Figure 9



(a) An example of thickened road lines

(b) The road centerlines from applying only the thinning operator on (a)

(c) The eroded road lines from applying the erosion operator on (a)

(d) The road centerlines from applying both the erosion and thinning operators on (a)

Fig. 11 Distorted road lines near road intersections caused by the thinning operator

erosion operator and then apply the thinning operator. Figure 11(c) and Figure 11(d) show that the extent of the line distortion is smaller after we apply the erosion operator; however, the distortion is still not completely eliminated and will be handled in the next step by extracting accurate road geometry around intersections.

3.3 Single-Pass Parallel-Pattern Tracing Algorithm

In our previous work [Chiang et al., 2008], we developed the Parallel-Pattern-Tracing algorithm (PPT) for identifying the road format (i.e., single-line or double-line format) and road width of the dominant road type in the raster map. For example, if 80% of the roads in the map have the road width as 10 pixels and the other 20% have the road width as 3 pixels, the resulting road width after the PPT is 10 pixels.

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 we find a corresponding pixel in each direction, we classify the pixel as a parallel-pattern pixel of the given road width. By applying the PPT iteratively from 1-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.

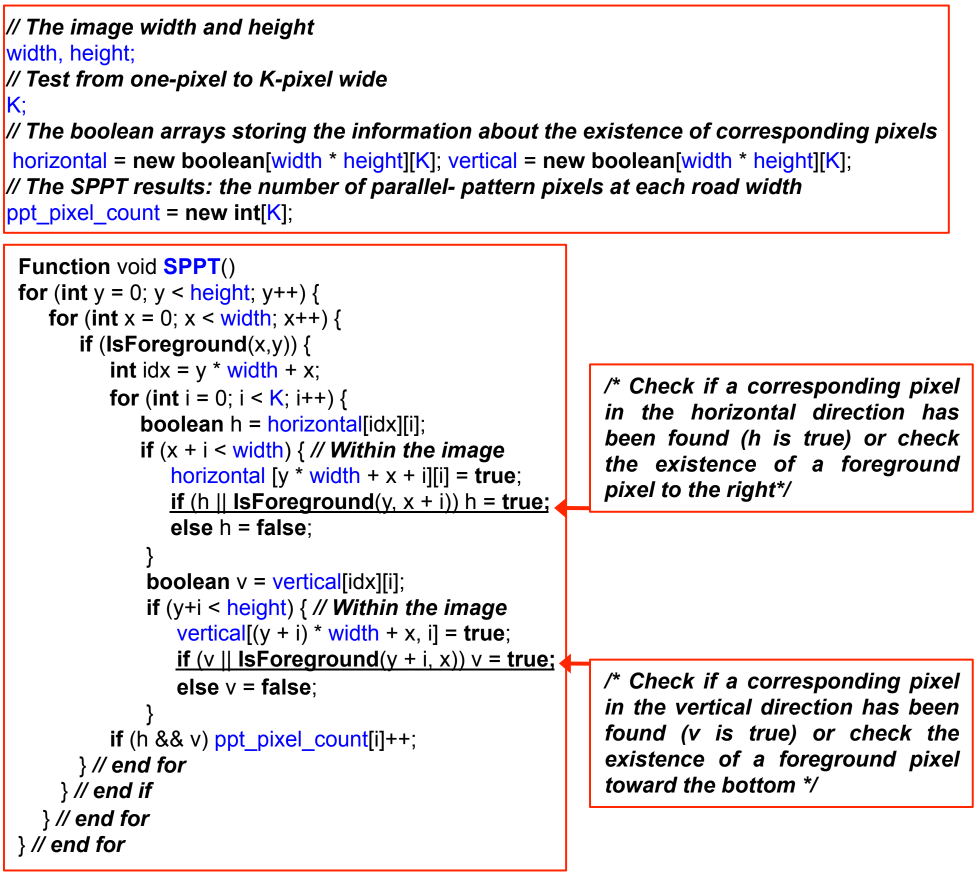
In our previous work, we implemented the PPT using two convolution masks. One convolution mask works in the horizontal direction and the other one works in the vertical direction to find the corresponding pixels. The sizes of the convolution masks are designed to cover the road areas in-between two parallel road lines: if the road width is X pixels, the size of the convolutions mask is YxY pixels and Y is $X \times 2 + 1$. For example, if the road width is 2 pixels, the size of the convolution masks is 5x5 pixels, which means by applying the convolution masks to a pixel, we check one pixel at 2-pixel-distance to left (towards the top) of this pixels, one pixel at 2-pixel-distance to right (towards the bottom) of this pixels. Note that these convolution masks can misidentify parallel-pattern pixels if there are non-road pixels in the image, which is fine because the PPT does not need every parallel-pattern pixel to be identified for detecting the road width [Chiang et al., 2008]

For N foreground pixels, the number of computing steps to iteratively apply the PPT on the road width from 1 pixel to K pixel is:

$$PPT(N, K) = N \times \sum_{r=1}^{K} (2r+1)^2 \qquad (1)$$

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, we developed the Single-Pass Parallel-Pattern-Tracing algorithm (SPPT), which does not rely on the image convolution and only requires a single-pass scan on the image. Moreover, the SPPT keeps a record of previously identified parallel patterns to further reduce the time complexity, which works as follows: in the previous work, to check whether there is a parallel pattern for a foreground pixel, P, at



```
// 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 right*/

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

Fig. 12 The pseudo-code of the SPPT

a distance D, the PPT has to check 4 pixels, 1 pixel towards the top of P at the distance D, 1 pixel towards the bottom of P at the distance D, 1 pixel to the left of P at the distance D, and 1 pixel to the right of P at the distance D. In the SPPT, to check whether there is a parallel pattern for a foreground pixel, P, at a distance D, the algorithm checks the parallel-pattern records and 2 pixels, 1 pixel to the right of P at the distance D and 1 pixel towards the bottom of P at the distance D.

Figure 12 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. To check the parallel pattern from 1 to K pixels, for a foreground pixel, the SPPT first records the existence of this foreground pixel for the pixels at the distance from 1 to K pixels to the right and towards the bottom of this foreground pixel (i.e., set the *horizontal* and *vertical* arrays as true in the pseudo-code). Next, the SPPT checks if the foreground pixel has a previously found parallel-pattern pixel (i.e., check the horizontal and vertical arrays using the position of this foreground pixel) and then checks the pixels at the distance from 1 to K pixels to the right and towards the bottom of this foreground pixel (i.e., the *IsForeground* function in the pseudo-code). The parallel pattern record (i.e., the *horizontal* and *vertical* arrays) eliminates searching in the pixel's left/top direction. Therefore, for N foreground pixels, to iteratively apply the SPPT on the road width from 1 to K pixels wide, the number of steps is:

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

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

The PPT keeps one record of the number of parallel-pattern pixels for each road width for the entire image; and the overall space complexity for the PPT in addition to the space for storing the image is:

$$PPT(N,K) = K \tag{3}$$

The SPPT keeps tracking whether a foreground pixel is a parallel-pattern pixel during the process. The tracking record for each foreground pixel includes an array of size K for the horizontal direction and an array of size K for the vertical direction. The overall space complexity for the SPPT in addition to the space for storing the image is:

$$SPPT(N,K) = 2 \times N \times K \tag{4}$$

The SPPT trades the space complexity for less computational steps. The PPT is an exponential time algorithm with linear space complexity while the SPPT is a linear time algorithm with linear space complexity.

Once we have the SPPT result, we build a parallel-pattern histogram using the number of parallel-pattern pixels as the X-axis and the road width as the Y-axis. We then identify the road width by analyzing the histogram to detect peaks in the histogram [Zack et al., 1977]. The detailed algorithm can be found in [Chiang et al., 2008].

4 Step Two: Road Intersection Detection

In this section, we describe our techniques for extracting accurate road geometry around road intersections (i.e., road-intersection templates) to then generate accurate road vector data in a later step. A road-intersection template represents the road geometry around a road intersection, which is the position of a road intersection, the orientations of the roads intersecting at the intersection, and the connectivity of the road intersection. Figure 13 shows the overall approach to automatically extract accurate road-intersection templates. This paper builds on our previous work [Chiang et al., 2008], which focuses on extracting positions of road intersections.

4.1 Generating Road-Intersection Blobs to Label Distorted Lines

Since the thinning operator produces distorted road geometry near the road intersections and the road width determines the extent of the distortion, we can utilize the extracted road intersections and the road width to label the locations of the potential distorted lines. We first generate a blob image with the detected road-intersection points labeled as individual foreground pixels. Then, we apply 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 14(a) shows an example map and Figure 14(b) shows the blob image after we apply 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, we overlap the blob image with the thinned-line image shown in Figure 14(c) to label the extent of the potential distorted lines as show in Figure 14(d).

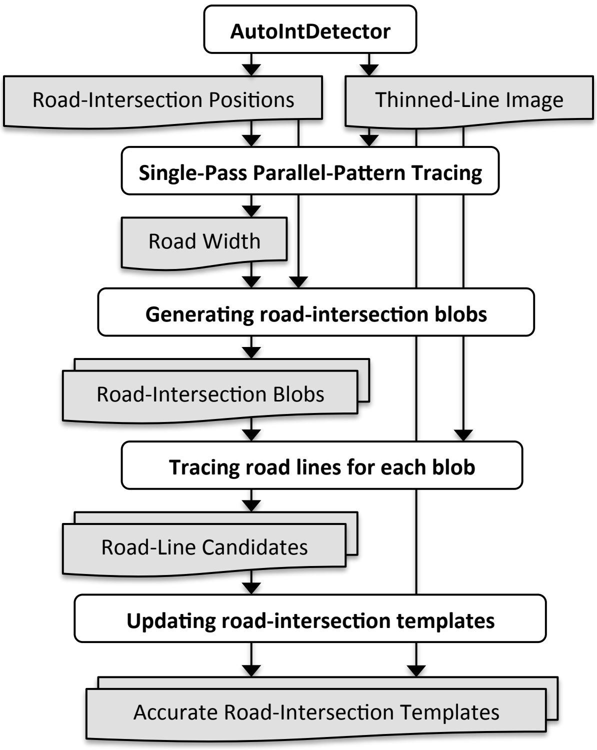
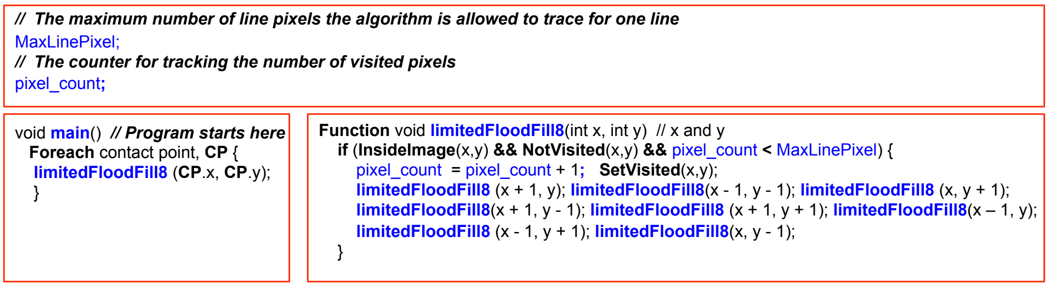

AutoIntDetector
Road-Intersection Positions
Thinned-Line Image
Single-Pass Parallel-Pattern Tracing
Road Width
Generating road-intersection blobs
Road-Intersection Blobs
Tracing road lines for each blob
Road-Line Candidates
Updating road-intersection templates
Accurate Road-Intersection Templates

Fig. 13 The overall approach to extract the road-intersection templates from raster maps

4.2 Identifying and Tracing Road-Line Candidates

To extract accurate road vector data around the intersections, we use the labeled image shown in Figure 14(d) to detect possible road lines intersecting at each road intersection (i.e., road-line candidates) and trace the thinned-line pixels to compute the line orientations. We first identify 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 14(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.

Once we have the contact points, to detect the road-line candidates, we present the Limited Flood-Fill algorithm to trace the thinned-lines from their contact points. Figure 15 shows the pseudo-code for the Limited Flood-Fill algorithm. The Limited Flood-Fill algorithm 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 is labeled as a potential distorted line pixel, the neighboring pixel is

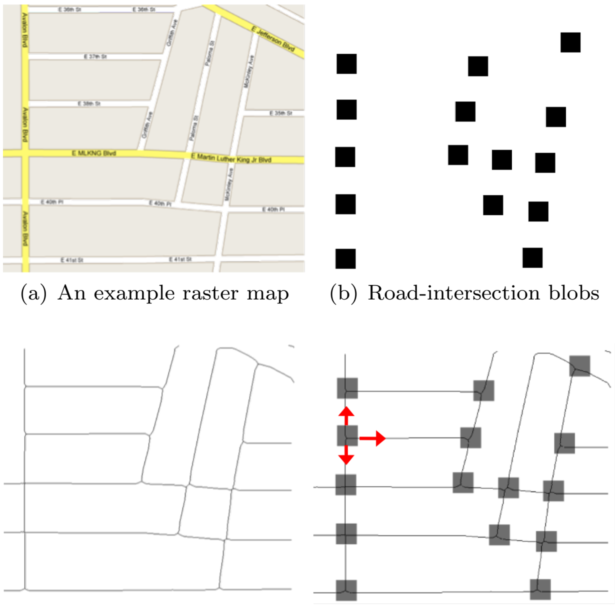


```
// The maximum number of line pixels the algorithm is allowed to trace for one line
MaxLinePixel;
// The counter for tracking the number of visited pixels
pixel_count;
```

```
void main() // Program starts here
  Foreach contact point, CP {
   limitedFloodFill8 (CP.x, CP.y);
  }
```

```
Function void limitedFloodFill8(int x, int y) // x and 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);
  }
```

Fig. 15 The pseudo-code for the Limited Flood-Fill algorithm



(a) An example raster map (b) Road-intersection blobs

(c) Thinned road lines (d) Labeling distortion areas

Fig. 14 Generating a blob image to label the distorted lines

set as the next visit point for the Limited Flood-Fill algorithm to process.

When the Limited Flood-Fill algorithm processes a new pixel, it records the position of the pixel to later compute the road orientation. Since it is very unlikely the road lines near an intersection are significantly curved, to trace only straight lines, we limit 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.

A smaller value for the *MaxLinePixel* variable prevents the Limited Flood-Fill algorithm from tracing curved lines. However, a very small value for the *MaxLinePixel* variable does not provide enough pixels for the Limited Flood-Fill algorithm to compute the road orientations. For example, if we set the *MaxLinePixel* variable to 1 pixel, there will be only eight possible orientations of the traced lines, which is not practical. In our approach, we empirically use 5 pixels for the *MaxLinePixel* variable to reduce the chance of tracing curved lines while still having enough pixels to generate the road orientations. As shown in Figure 16, 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), we utilize the *MaxLinePixel* to ensure that the Limited Flood-Fill algorithm traces only a small portion of the thinned-lines near the contact points.

After the Limited Flood-Fill algorithm processes every line from each contact point and records the positions of the line pixels, we utilize 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 computed line functions are then used in the next step of updating road-intersection templates to identify actual intersecting road lines and refine the positions of the road intersections.

4.3 Updating Road-Intersection Templates

There are three possible intersecting cases for the road-line candidates of one intersection as shown in Figure 17, where: the left images of 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 17 shows Case One where all the road-line candidates intersect at one point. The middle row shows Case Two 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

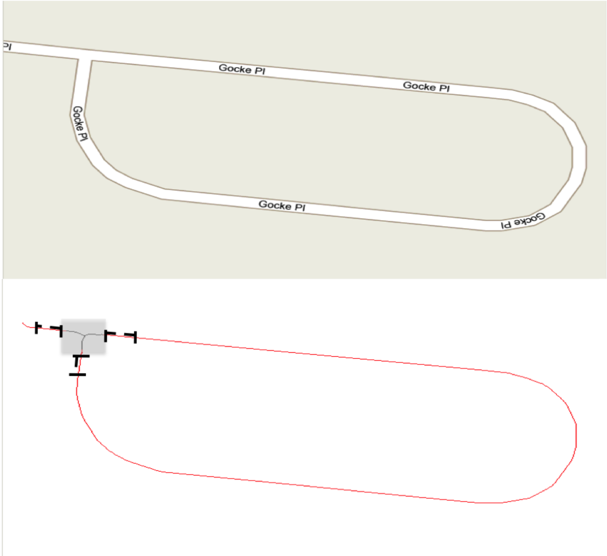


Fig. 16 Tracing only a small portion of the road lines near the contact points

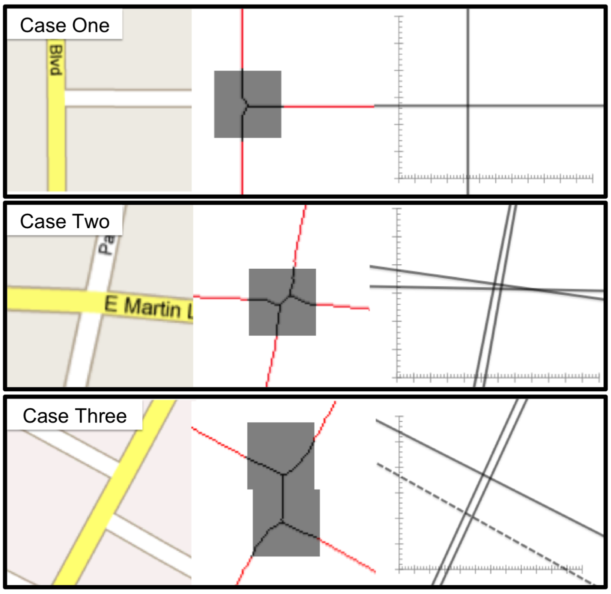


Fig. 17 The three intersecting cases for updating road-intersection templates

row shows Case Three 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 Case One, we adjust the position of the road intersection to the intersecting point of the road-line candidates. We keep 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.

For Case Two, Figure 18 shows a detailed view where: the solid red dot is the initially detected road-intersection

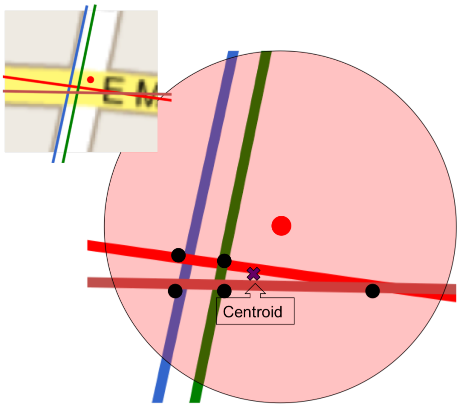
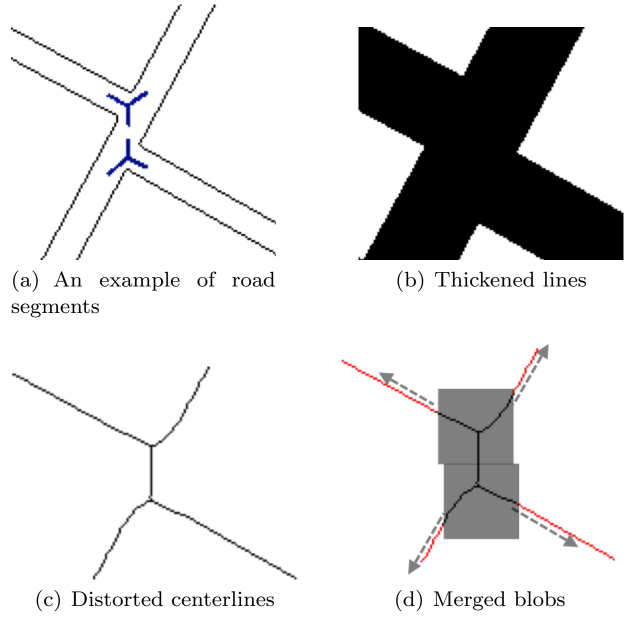


Fig. 18 Case Two: adjusting the road-intersection position without outliers

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 are in the semi-transparent red circle, we adjust the position of the road-intersection template to the centroid of the intersecting points of all road-line candidates. We keep 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.

For Case Three, when two road intersections are very close to each other, the road geometry between them is totally distorted as shown in Figure 19. In this case, the blobs of the two road intersections merge into one big blob as shown in Figure 19(d), and we associate both road intersections with the four thinned-lines linked to this blob. Figure 20 shows a detailed view of Case Three. 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, we discard the dashed road-line candidate. We use the centroid of the remaining two intersecting points as the position of the road-intersection template. Since we discard 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. Case Three shows how the blob image helps to extract correct road



(a) An example of road segments

(b) Thickened lines

(c) Distorted centerlines

(d) Merged blobs

Fig. 19 Merged nearby blobs

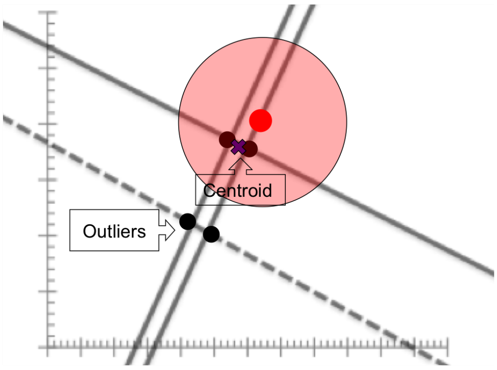
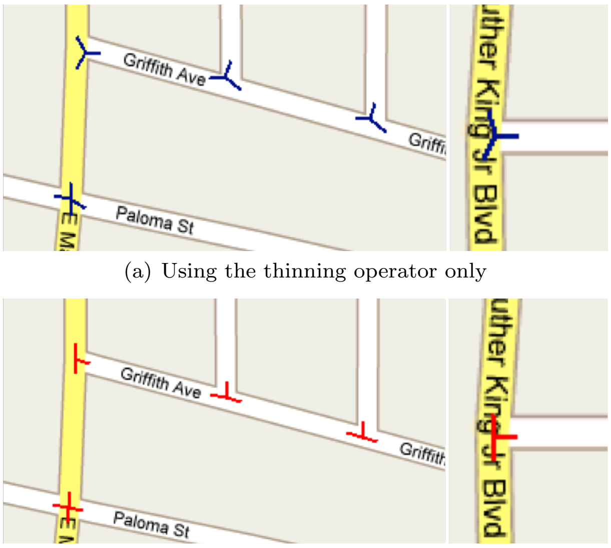


Fig. 20 Case Three: adjusting the road-intersection position with outliers

orientations even when an intersecting road line is totally distorted by the thinning operator. This case is not limited to three intersecting roads. Our approach holds when the distorted road line has the same orientation as the lines outside the distortion area. For example, in Case Three shown in Figure 17, the distorted line is part of a straight line that goes throughout the intersection so it has the same orientation as the 240-degree line. In addition, the intersecting road lines need to have a similar road width because the road width is used to determine the outlier of the road-line candidates.

Figure 21 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, we automatically detect and correct the distorted lines around road intersections caused by the thinning operator and generate accurate road-intersection templates. Since this approach is based on the heuristic that the road lines near an intersection are straight within a short distance smaller than the *MaxLinePixel* parameter, for significantly curved road lines around the road intersections (i.e., roads that are curved and shorter than the *MaxLinePixel* parameter), the traced line functions would not be accurate.



(a) Using the thinning operator only

(b) Accurate road-intersection templates

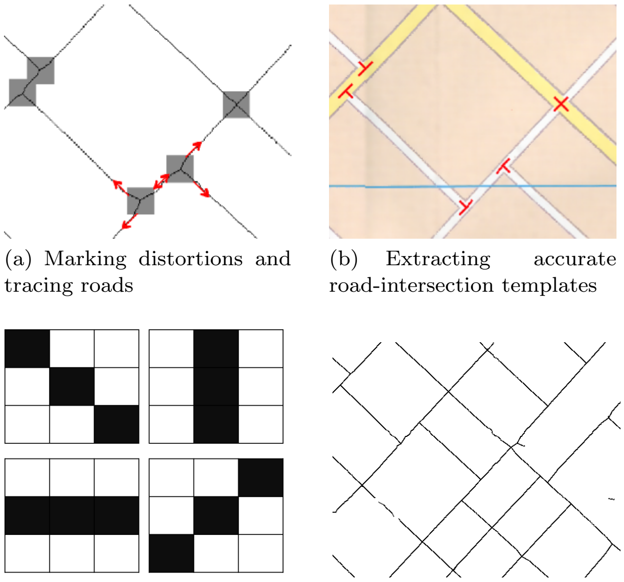
Fig. 21 Example results compared to using the thinning operator only

5 Step Three: Road Vectorization

In this section, we describe our techniques for vectorizing the extracted road geometry using the road-intersection templates. For large raster maps, instead of processing the entire map at once, we first divide the map into 2000x2000-pixel tiles with overlapping areas on their connected borders and extract the road vector data for each tile. Then we combine the road vector data from each tile and generate the road vector data for the entire map.

5.1 Road Vectorization Using Road-Intersection Templates

With the knowledge of potential distorted areas and the accurate positions of the road intersections as shown in Figures 22(a) and 22(b), we start 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 22(a) shows the three types of pixels, which are the black pixels not covered



(a) Marking distortions and tracing roads

(b) Extracting accurate road-intersection templates

(c) Straight-line patterns

(d) Extracted road vector data

Fig. 22 Extracting road vector data from an example map

by the gray boxes, black pixels in the gray boxes, and white pixels, respectively. We create a list of *connecting nodes* (CNs) of the road vector data. A CN is a point where two lines meet at different angles. We first add the detected road intersections into the CN list. Then, we identify 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 22(c). We add the pixel to the CN list if we *do not* detect a straight-line pattern since the road pixel is not on a straight line.

To determine the connectivity between the CNs, we developed an eight-connectivity flood-fill algorithm called the Road-Tracer to trace the road pixels. Figure 23 shows the pseudo-code of the Road-Tracer. Note that the Road-Tracer algorithm and the Limited Flood-Fill algorithm in Figure 15 are both derived from the traditional image processing flood-fill algorithm, which is used to determine the pixel connectivity or to fill up connected areas. The differences between the the Road-Tracer algorithm and the Limited Flood-Fill algorithm in this paper are their stopping criteria.

The Road-Tracer 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, we use the previously updated road intersection positions as the CNs' positions. The CN list and their connectivity are the results of our extracted road vector data. Figure 22(d) shows the extracted road vector data. The road vector data around the road intersections are accurate since we do not generate any CN using the distorted lines except the road intersections (i.e., our algorithm does not record the geometry of the distorted lines) and the intersection positions are updated using the accurate road orientations.

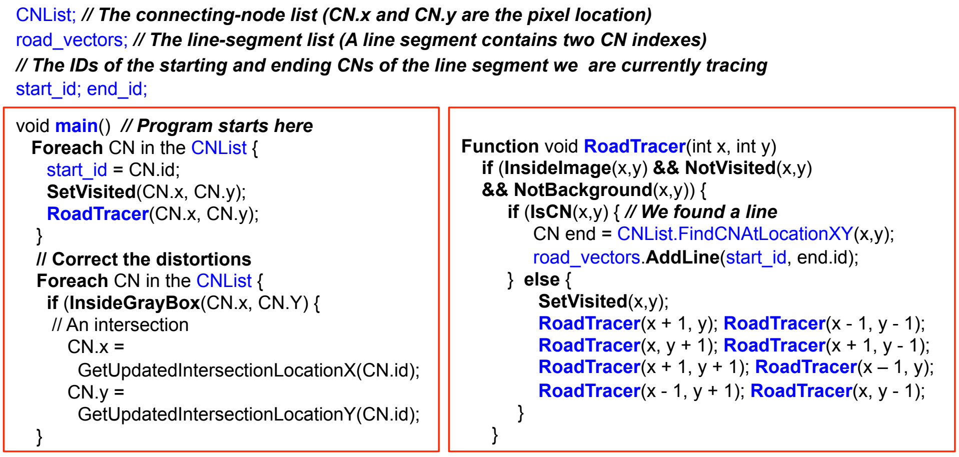
5.2 Divide-and-Conquer Extraction of Road Vector Data

We divide a raster map into overlapping tiles and process each tile for extracting its road vector data. Figure 24 shows an input scanned map and we divide the desired map region into four overlapping tiles. After we process all the tiles, we combine the extracted road vector data from each tile as one set of road vector data for the entire raster map.

For the extracted road vector data of each tile, we cut the vector data at the center of the overlapping areas as the dashed lines shown in Figure 24, and we discard 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).

We merge 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 24(b) shows two sets road vector data from two neighboring tiles. We first generate 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, we generate the set of cuttings points for the right set of road vector data. Finally, we merge the two sets of road vector data by connecting two road lines in the two tiles ending at the cutting points of the same location as shown in Figure 24(c) where each of the horizontal arrows point to a matched pair of cutting points shown as the cross marks. Figure 24(d) shows the merged road vector data.

We first merge the road vector data of tiles on the same row from left to right and then we merge the integrated vector data of each row into the final results from top to bottom. Note that the divide-and-conquer approach does not reduce the overall computational complexity, but introduces additional computational overhead (i.e., more pixels need to be processed). However, by dividing the input image into tiles that can be processed independently, our road vectorization process can scale to arbitrarily large images as long as we can divide the input image into smaller regions. Moreover, since each tile is processed independently, the road



```
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);
    RoadTracer(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 RoadTracer(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);
          RoadTracer(x + 1, y); RoadTracer(x - 1, y - 1);
          RoadTracer(x, y + 1); RoadTracer(x + 1, y - 1);
          RoadTracer(x + 1, y + 1); RoadTracer(x – 1, y);
          RoadTracer(x - 1, y + 1); RoadTracer(x, y - 1);
      }
   }
```

Fig. 23 The pseudo-code of the Road-Tracer algorithm

vectorization process can take the advantage of multi-core or multi-processor computers to process the tiles in parallel.

6 Experiments

In this section, we report on our experiments on the extraction of road vector data from heterogeneous raster maps using the techniques described in this paper. We have implemented our overall approach as two components in a system called Strabo. The first component, called Road Layer Extraction, is the supervised road-pixel-extraction technique in Section 3 (the first step, road geometry extraction). This component takes a raster map as input and extracts the road layer. The second component, called Road Layer Vectorization, includes the remaining techniques in Section 3 and the road intersection detection and road vectorization techniques in Sections 4 and 5. This component takes the extracted road layer as input and generates the road vector data.

We tested Strabo on 40 maps from 11 sources. Table 1 shows the information of the test maps and their abbreviations used in this section.[12] The ITM, GECKO, GIZI maps cover the city of Baghdad, Iraq and were scanned in 350 DPI. The UNIraq map covers the city of Samawah, Iraq, which is from the United Nations Assistance Mission for Iraq website[13] and provides no information of scan resolution. The UNAf map covers Afghanistan and is from the United Nations Assistance Mission in Afghanistan website.[14] The Afghanistan map shows the main and secondary roads, cities, political boundaries, airports, and railroads of the nation. The RM map covers the city of St. Louis, Missouri and is from Rand McNally.[15] The Google, Live, OSM, MapQuest, Yahoo maps cover one area in Los Angeles, California and one area in St. Louis, Missouri, which are from Google Maps, Microsoft Live Maps, OpenStreetMap, MapQuest Maps, and Yahoo Maps, respectively. Figure 25 shows examples of the test maps, where the scanned maps show poor image quality, especially the Gecko and Gizi maps with the shadows caused by the fold lines.

Map Source (map count, abbr.)	Map Type	Dimension (pixels)
International Travel Maps (6, ITM)	Scanned	4000x3636
Gecko Maps (3, GECKO)	Scanned	5264x1923
Gizi Map (4, GIZI)	Scanned	3344x3608
UN Iraq (9, UNIraq)	Scanned	4000x3636
Rand McNally (4, RM)	Computer	2084x2756
UN Afghanistan (4, UNAfg)	Computer	3300x2550
Google Maps (2, Google)	Computer	800x550
Live Maps (2, Live)	Computer	800x550
OpenStreetMap (2, OSM)	Computer	800x550
MapQuest Maps (2, MapQuest)	Computer	800x550
Yahoo Maps (2, Yahoo)	Computer	800x550

Table 1 Test maps

6.1 Experimental Setup

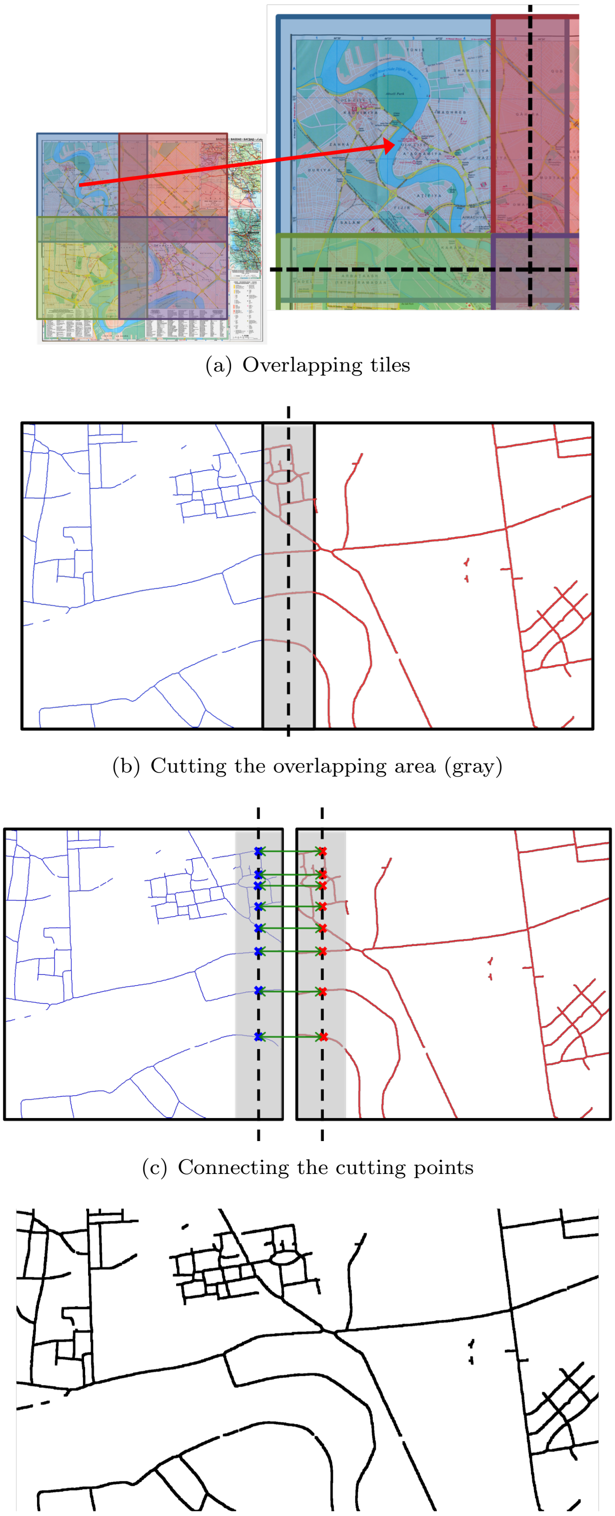
This section presents the experimental setup for the road layer extraction and vectorization components in Strabo.

[12] The detailed information for obtaining the test maps and the ground truth can be found on: `http://www.isi.edu/integration/data/maps/prj_map_extract_data.html`

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

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

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



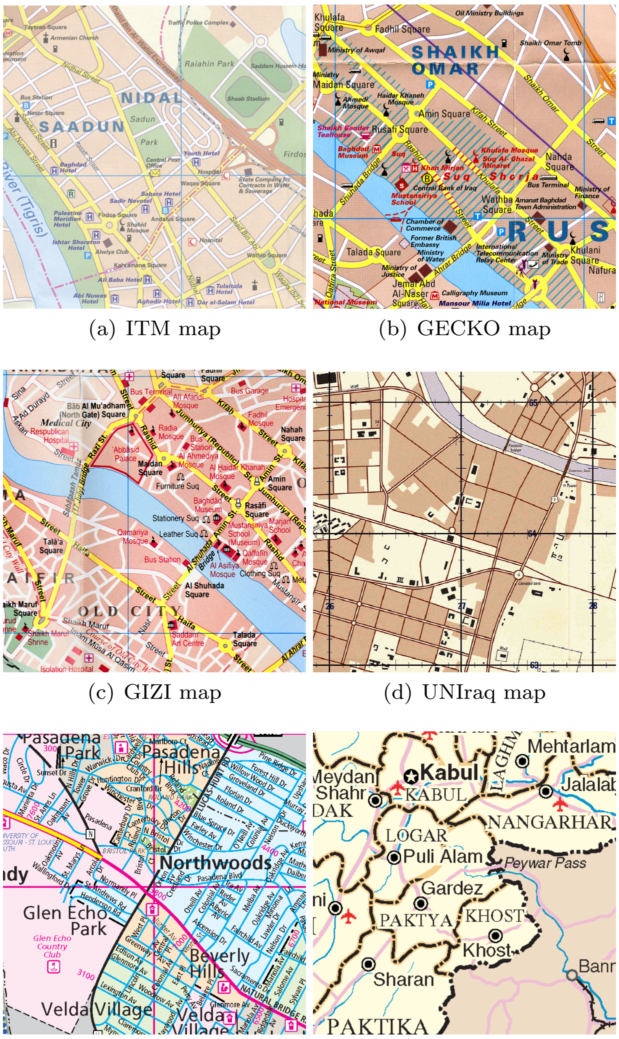
(a) Overlapping tiles

(b) Cutting the overlapping area (gray)

(c) Connecting the cutting points

(d) The merged road vector data

Fig. 24 Merging two sets of road vector data from neighboring tiles



(a) ITM map

(b) GECKO map

(c) GIZI map

(d) UNIraq map

(e) RM map

(f) UNAfg map

Fig. 25 Examples of the test maps

Road Layer Extraction We first used Strabo to generate a road layer for each test map automatically. This automatic technique of Strabo is implemented based on our previous grayscale-histogram analysis technique [Chiang et al., 2008]. Then we manually checked the road layer to determine if user intervention was required.

Strabo successfully extracted the road layer from the last five sources shown in Table 1 and did not extract correct road layers for the other six sources. Among the six sources (ITM, GECKO, GIZI, UNIraq, RM, and UNAfg), four sources correspond to scanned maps (ITM, GECKO, GIZI, and UNIraq) since the grayscale-histogram analysis technique could not separate the foreground pixels from the background. The other two map sources contain non-road linear features, which are drawn using the same single-line format as the roads, and hence the automatically extracted road layers contain these linear features. To achieve the best results for the six sources, we utilized the supervised technique presented in Section 3.1 to extract the road layers.

The supervised technique first quantized the maps from the four scanned-map sources (ITM, GECKO, GIZI, and UNIraq) to generate quantized images in various quantization levels (the quantization levels have the number of colors as 32, 64, 128, and 256, respectively). We did not apply the color segmentation algorithms on computer-generated maps (test maps from 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 5 colors representing roads. The user starts using Strabo for the user-labeling task with the quantized image of the highest quantization level (i.e., the quantized image that has the smallest number of colors). If the user cannot distinguish the road pixels from other map features (e.g., background) in the quantized image, the user selects a quantized image containing more colors (a lower quantization level) for user labeling.

For comparison, we tested the 40 test maps using R2V from Able Software. R2V allows the user to use one set of color thresholds to extract the road pixels from the map for vectorization. For raster maps that require more than one set of color thresholds (all of our test maps 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 colors, which requires significant effort. Therefore, for the raster maps that require more than one set of color thresholds to extract their road pixels, we used the road layers extracted from Strabo (which are the same set of road layers used to test Strabo's road vectorization function) without using R2V's manual pre-processing and post-processing. The UNAfg map requires only one set of color thresholds to extract the road pixels using R2V and both R2V and Strabo generated the same road layer for the UNAfg map.

Road Layer Vectorization Once the road layers were extracted, the second component of Strabo then processed the road layers and automatically generated the road vector data. For comparison, we utilized "Auto Vectorize" function in R2V to process the same set of road layers for generating the road vector data automatically.

6.2 Evaluation Criteria

This section describes the evaluation criteria for the road layer extraction and vectorization components in Strabo.

Road Layer Extraction To evaluate the road-layer-extraction component, we report the number of user labels that were required for extracting road pixels from each map source using Strabo.

Road Layer Vectorization For evaluating the extracted road vector data from the road-layer-vectorization component, we 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. We had a third-party to manually draw the centerline of every road line in the maps as the ground truth.

The completeness is the length of true positives divided by the sum 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 sum 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 is the length of matched extraction minus the length of matched reference. 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 redundancy does not depend on the number of line segments in the matched extraction line or the matched reference. 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 identify the length of the true positives, false negatives, and matched ground truth, Heipke et al. [1997] suggest using a buffer width of half of the road width in the test data so that a correctly extracted road segment is in between the road edges as shown in Figure 26. In our test maps, the roads range from 5 to 12 pixels wide. We used a buffer width of 3 pixels, which means a correctly extracted line is no farther than 3 pixels from the road centerlines. For example, to calculate the length of the true positives, we first drew the extracted road vector data using 1-pixel-width lines as the blue lines shown in the top of Figure 26 (i.e., the length of a road segment is approximately the number of pixels of the drawn road line). Then we drew the ground truth

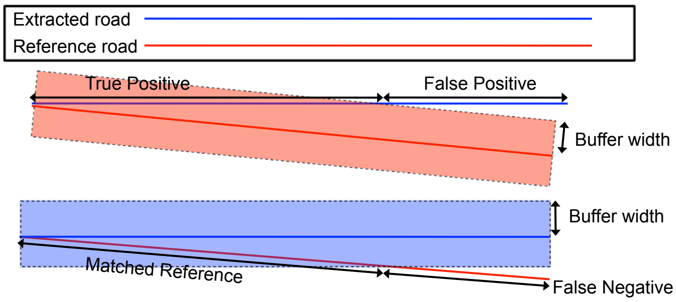


Fig. 26 Calculating road extraction metrics

lines (the reference roads) using the width as the buffer width on top of the blue lines as the red box shown in in Figure 26. The number of blue pixels outside the red box is the length of false positives and the total number of blue pixels minus the length of false positives is the length of true positives. More details on calculating the metrics can be found in [Heipke et al., 1997].

6.3 Experimental Results

This section presents the experimental results of the road layer extraction and vectorization components in Strabo.

Road Layer Extraction Table 2 shows the numbers of colors in the images used for user labeling and the numbers of user labels used for extracting the road pixels. Strabo did not generate quantized maps for the RM and UNAfg maps (i.e., no values for their quantized map colors in Table 2) because they contain only a small number of colors.

For all the scanned maps, only 1 to 9 labels were needed for Strabo to extract the road pixels. The number of user labels varied from 1 to 9 because of the varying complexity of map content and the number of road colors of the tested map sources. Strabo's user labeling function is an interactive process. During the road-layer-extraction experiments, a user first selected one or more labels for a map source and then instructed Strabo to show the layer extraction results using the selected labels. If not all of the road lines were extracted, the user would provide more labels and then re-examine the results. This interactive labeling process continued until all of the road lines were extracted. Since the user label does not have to contain only road pixels, the user does not have to carefully avoid including non-road pixels in the label and hence does not take a long time to decide on a label (i.e., usually less than 30 seconds).

In contrast to R2V, which requires manually providing samples of each road color, Strabo's interactive strategy provides a much easier-to-use approach for extracting road layers from raster maps. Note that if the color quantization failed to group the road colors into groups because of poor map quality, as found in some historical maps, this interactive strategy would not work well. In this case, a more advanced color segmentation technique could be used to produce a quantized map for user labeling [Chiang et al., 2011].

Road Layer Vectorization Table 3 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 redundancy of Strabo and R2V are shown in the bottom rows of Table 3. Strabo produced more accurate road vector data with a smaller RMS. We emphasize the numbers where R2V generated a better result than Strabo. R2V produced better completeness numbers for five test maps, but this was because R2V generated highly redundant lines, while Strabo eliminated small branches, such as the highway ramps shown in Figure 27.

R2V could achieve better results if we tuned R2V with manually specified pre-processing and post-processing functions. To demonstrate the pre-processing and post-processing functions in R2V for improving the vectorization result on the road layer shown in Figure 27(b), we first manually resized the image to a quarter of the original size for reducing the thickness of the line areas. Next, we applied the de-speckle function in R2V to remove noise objects. We then used the image editing function in R2V to manually draw lines to fill up the gaps between broken lines and holes; in this particular example, we manually drew 14 areas and the black pixels in Figure 28 shows the edited result. Finally, we applied the automatic vectorization function with spline smoothing to generate the results as the road lines shown in Figure 28. Note that the results are significantly improved after manual processing, but the road geometry near the intersections is not accurate compared to our results shown in Figure 27(c) due to the fact that Strabo automatically detects and corrects distorted lines near road intersections.

One limitation of Strabo's automatic vectorization process is that the process to generate road geometry relies on the width of the majority of roads. This can be seen in Figure 27 where Strabo eliminated small branches because Strabo detected the road width as the width of the majority of roads (in Figure 27, the majority roads are the white roads) and used the detected road width to set up the parameters for generating the road geometry automatically. As a result, in the examples in Figure 27, the number of iterations of the erosion operator was larger than the width of the small branches so that the branches were eliminated after applying the erosion operator.

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
UNAfg	90	N/A	1

Table 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

Map Source	Completeness	Correctness	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 (Strabo)	99.62%	99.87%	99.49%	0.00%	0.81
Google (R2V)	83.45%	81.93%	70.41%	18.78%	19.16
Live (Strabo)	99.47%	98.31%	97.79%	0.00%	8.08
Live (R2V)	83.42%	71.36%	62.69%	29.25%	23.85
OSM (Strabo)	99.81%	100.00%	99.81%	0.00%	0.76
OSM (R2V)	90.47%	93.71%	85.79%	6.82%	10.84
MapQuest (Strabo)	99.85%	100.00%	100.00%	0.00%	0.73
MapQuest (R2V)	92.01%	93.41%	87.149%	7.52%	6.72
Yahoo (Strabo)	99.97%	99.97%	99.94%	0.00%	0.69
Yahoo (R2V)	86.10%	77.17%	68.62%	103.29%	26.49
Avg. (Strabo)	95.07%	96.70%	92.15%	0.28%	3.05
Avg. (R2V)	90.66%	82.10%	75.64%	39.98%	13.22

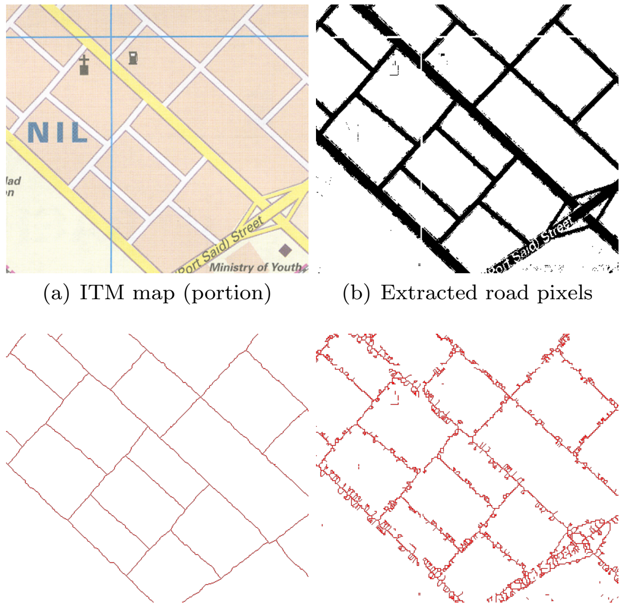
Table 3 Numeric results of the extracted road vector data (3-pixel-wide buffer) using Strabo and R2V

Figures 29 to 35 show some example results. Note that the geometry of the extracted road vector data is very close to the road centerlines for both straight and curved roads, especially the computer-generated maps from the web-mapping service providers.

For the lower than 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 UNAfg, GECKO, and GIZI maps shown in Figures 29, 30, and 31. 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. One post-processing example could be connecting extreme points that are within the distance of a user-specified threshold in the road vector data. This post-processing step could potentially increase the completeness of the road vector data results. For the lower than average completeness numbers in the scanned UNIraq map, some of the road lines as shown in Figure 32 are dashed lines and the ground truth were drawn as solid lines.

For the text labels in the test maps, except the RM maps, the maps contained text labels drawn in a different color than the roads so that the text pixels were removed during the extraction of road layers (i.e., Strabo identified the colors that represent roads and used the road color to extract the road layers). The overlapping text was also removed and hence the extracted road layers were broken as shown in the examples of the ITM, UNAfg, Gecko, and GIZI maps in Figure 29 to Figure 31.

For the RM maps, Figure 33(b) shows the extracted road pixels using the supervised function of Strabo. Many characters were extracted since they share the same color as the black lines. Although we removed



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

(c) Strabo results (d) R2V results

Fig. 27 Example results using Strabo and R2V of a cropped area from the ITM map

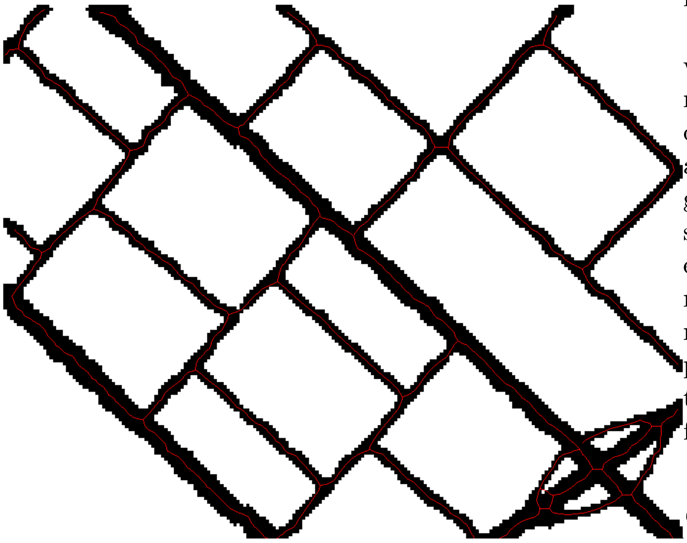


Fig. 28 R2V results with manual image editing (red lines are the extracted road vector data and black pixels are the edited road area

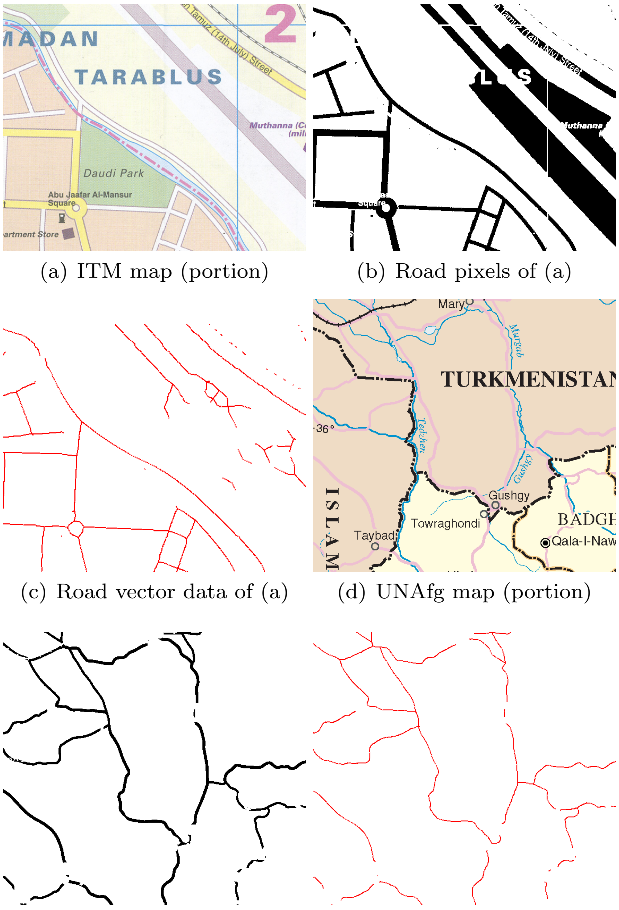
the majority of the characters automatically using the text/graphics separation and connected component analysis techniques [Cao and Tan, 2002; Chiang et al., 2008] as described in Section 3.3, some of the characters were miss-identified as road lines since they touch the road lines and the connected-component analysis approach we used could not remove this type of false positive, which resulted in lower completeness, correctness, and quality. For the Google, Live, OSM, MapQuest, and Yahoo maps, because of the good image quality, Strabo automatically separated the foreground pixels from the background and the foreground pixels contain both the text and road pixels as the example shown in Figure 35. Since the text labels in these maps were placed on top of the road lines, the extracted road layers show accurately connected road lines.

The ITM, GECKO, GIZI, UNIraq, and RM maps had lower than average 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 29(b) shows 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 32 shows a portion of the UNIraq map where some of the building pixels were extracted since they share the same colors as the road shadows. Figure 33 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. Including a user validation step after the road pixels were extracted could further reduce this type of false positive resulting in higher correctness numbers.

Strabo's redundancy numbers are generally low since we correctly identify the centerlines for extracting the road vector data. The average RMS differences are under 3 pixels, which shows that the thinning operator and our approach to correct the distortion result in good quality road geometry. For example, in the results shown in Figure 34 and Figure 35, 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.

6.3.1 Computation Time

We 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 of vectorizing the road pixels for a 800x550-pixel map was 5 seconds, for a 2084x2756-pixel map was 2 minutes, and for a 4000x3636-pixel 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 up the processes, such as multi-threading on processing map tiles of an input map.



(a) ITM map (portion) (b) Road pixels of (a)

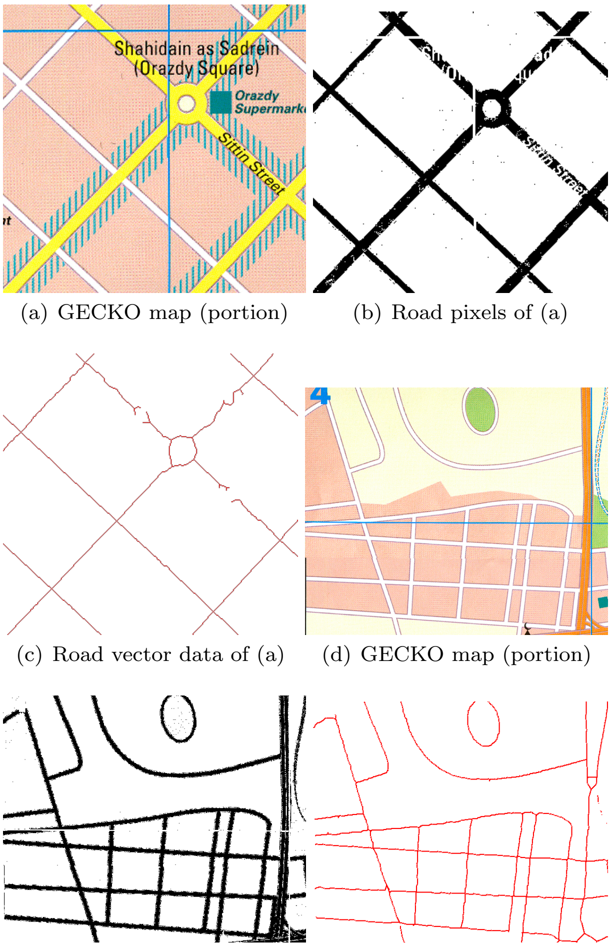
(c) Road vector data of (a) (d) UNAfg map (portion)

(e) Road pixels of (d) (f) Road vector data of (d)

Fig. 29 Examples of the road vectorization results on the ITM and UNAfg maps

7 Conclusion and Future Work

We present a general approach to extract accurate road vector data from heterogeneous raster maps with minimal user input. This approach handles raster maps with poor image quality using a semi-automatic technique. We show that our approach extracts accurate road vector data from 40 raster maps from 11 sources with varying color usages and image quality. In the future, we plan to extend our approach to include automatic post-processing on the road vector data. For example, without knowing the real-world lengths of the extracted road lines, we cannot apply heuristics for post-processing on the extracted road vector data, such as removing road lines that are shorter than 1 meter. With the extracted road vector data, we plan to utilize map conflation techniques, such as the one from Chen et al. [2008], to identify the geocoordinates of the road vector data and then discover the actual lengths of the extracted road lines. We can then utilize real-world heuristics, such as thresholds on the road length and road turning angles, to apply automatic post-processing on the extracted road vector data to improve the results.



(a) GECKO 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)

Fig. 30 Examples of the road vectorization results on the GECKO map

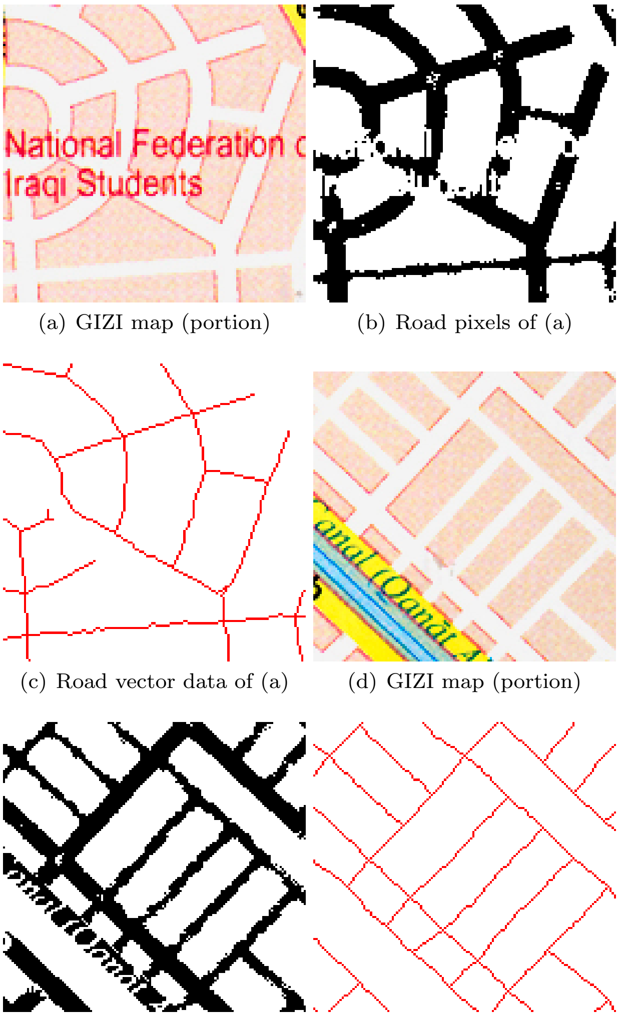
Acknowledgements

We thank Phyllis O'Neil for her excellent proofreading and editing work. We also thank Weili Chiang for manually digitizing the test maps to generate the ground truth.

This research is based upon work supported in part by the University of Southern California under the Viterbi School of Engineering Doctoral Fellowship.

References

Ballard, D. H. (1981). Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122.



(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)

Fig. 31 Examples of the road vectorization results on the GIZI map

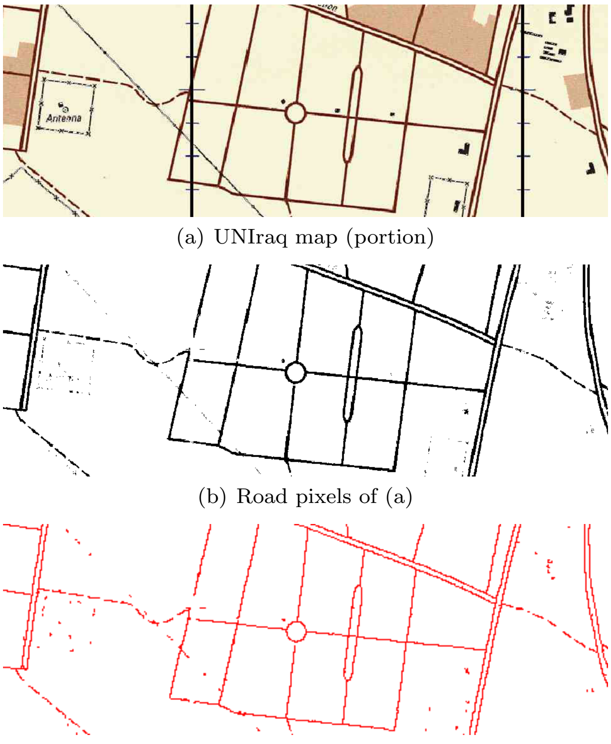
D. Bin and W. K. Cheong. A system for automatic extraction of road network from maps. In *Proceedings of the IEEE International Joint Symposia on Intelligence and Systems*, pages 359–366, 1998.

R. Cao and C. L. Tan. Text/graphics separation in maps. In *Proceedings of the Fourth GREC*, pages 167–177, 2002.

Cheng, H., Jiang, X., Sun, Y., and Wang, J. (2001). Color image segmentation: advances and prospects. *Pattern Recognition*, 34(12):2259 – 2281.

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

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

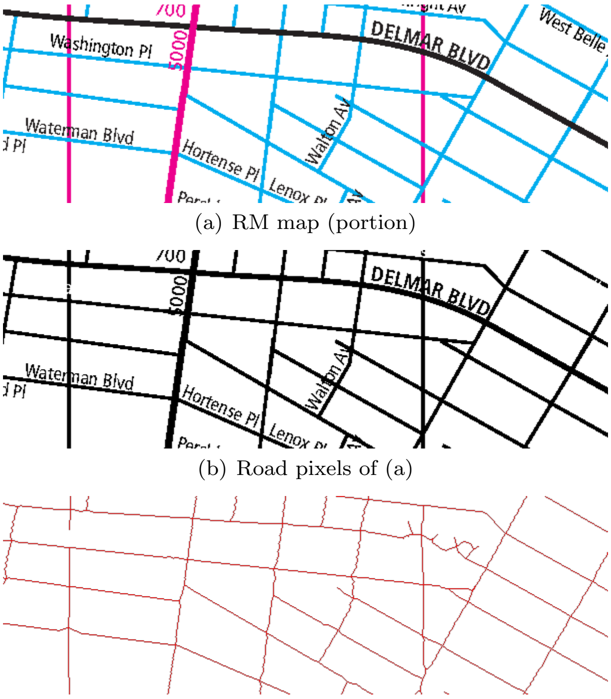


(a) UNIraq map (portion)

(b) Road pixels of (a)

(c) Road vector data of (a)

Fig. 32 Examples of the road vectorization results on the UNIraq map

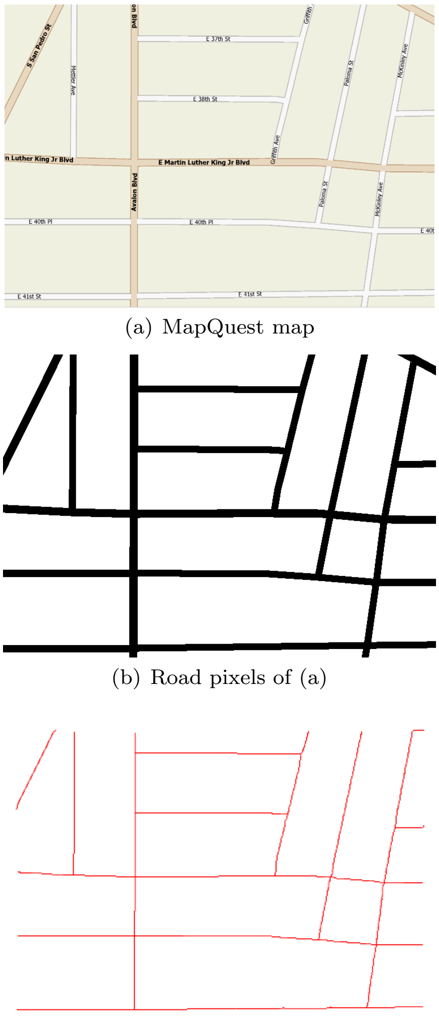


(a) RM map (portion)

(b) Road pixels of (a)

(c) Road vector data of (a)

Fig. 33 Examples of the road vectorization results on the RM map

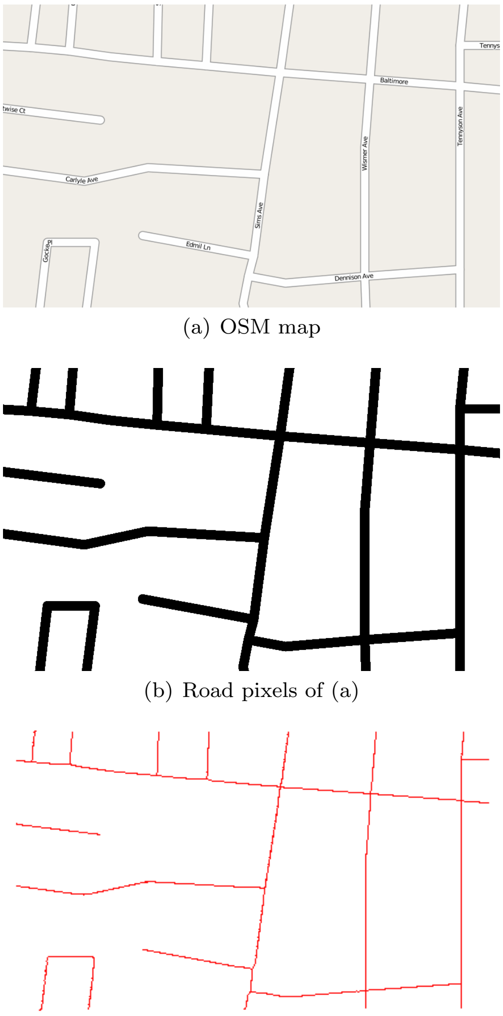


(a) MapQuest map

(b) Road pixels of (a)

(c) Road vector data of (a)

Fig. 34 Examples of the road vectorization results on a MapQuest map



(a) OSM map

(b) Road pixels of (a)

(c) Road vector data of (a)

Fig. 35 Examples of the road vectorization results on a OSM map

Y.-Y. Chiang and C. A. Knoblock. Automatic extraction of road intersection position, connectivity, and orientations from raster maps. In *Proceedings of the 16th ACM GIS*, pages 1–10, 2008.

Y.-Y. Chiang and C. A. Knoblock. A method for automatically extracting road layers from raster maps. In *Proceedings of the Tenth ICDAR*, pages 838–842, 2009a.

Y.-Y. Chiang and C. A. Knoblock. Extracting road vector data from raster maps. *Selected Papers of the Eighth GREC, LNCS*, 6020:93–105, 2009b.

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

Y.-Y. Chiang, S. Leyl, and C. A. Knoblock. Integrating Color Image Segmentation and User Labeling for Efficient and Robust Graphics Recognition from Historical Maps. *The Ninth IAPR International Workshop on Graphics RECognition*, 2011b.

D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE TPAMI*, 24(5):603–619, 2002.

Duda, R. O. and Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15:11–15.

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

P. Heckbert. Color image quantization for frame buffer display. *SIGGRAPH*, 16(3):297–307, 1982.

C. Heipke, H. Mayer, C. Wiedemann, and O. Jamet. Evaluation of automatic road extraction. In *Interna-*

tional Archives of Photogrammetry and Remote Sensing, pages 47–56, 1997.

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

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

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

V. Lacroix. 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.

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

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

Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137.

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

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

J. Shi and C. Tomasi. Good features to track. In *Proceedings of the IEEE CVPR*, pages 593–600, 1994.

Tombre, K., Tabbone, S., Pélissier, L., Lamiroy, B., and Dosch, P. (2002). Text/graphics separation revisited. In Lopresti, D., Hu, J., and Kashi, R., editors, *Document Analysis Systems V*, volume 2423 of *Lecture Notes in Computer Science*, pages 615–620. Springer Berlin / Heidelberg.

X. Wu, R. Carceroni, H. Fang, S. Zelinka, and A. Kirmse. Automatic alignment of large-scale aerial rasters to road-maps. In *Proceedings of the 15th ACM GIS*, pages 1–8, 2007.

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