AUTOMATICALLY AND ACCURATELY CONFLATING ROAD VECTOR DATA, STREET MAPS AND ORTHOIMAGERY

by

Ching-Chien Chen

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

May 2005

Dedication

To my parents and my wife, Chia-Hsin Liu

Without their support and love, this would not have been achieved.

Acknowledgements

I would like to thank and show my appreciation to my thesis advisors Professor Cyrus Shahabi and Professor Craig Knoblock for their direction, support, assistance, and guidance. Their suggestions and recommendations have been invaluable for my Ph.D. quest and my dissertation. Through their guidance and advice, I have learned how to do research, how to read research papers, how to present papers, how to give demonstrations, and how to organize my ideas and my experiences to write research papers and my dissertation. Moreover, I want to thank them for the freedom they gave me to explore my own paths towards solving problems.

I am also grateful to the other members of my dissertation committee: Professor John Wilson, Professor Michael Arbib and Professor Ramakant Nevatia for sharing with me their time and giving me valuable comments about this research. I would like to especially show my appreciation to Professor John Wilson for his helpful insights and comments in various aspects on the final draft of my dissertation and defense.

Moreover, I would like to thank all my friends and labmates for their advice and help. I enjoyed my graduate study with the members of the InfoLab. Thanks to Farnaz Azmoodeh, Farnoush Banaei-Kashani, Mehrdad Jahangiri, Farid Parvini, Mehdi Sharifzadeh, Kiyoung Yang, Hyunjin Yoon, Sahar Mastoureshgh, Bahareh Navai, Minyoung Mun, Dr. Mohammad Reza Kolahdouzan, Dr. Yi-Shin Chen and my friend Shou-De Lin. Especially, I appreciate Farnoush, Mehdi, Mohammad,

Kiyoung, Farid, Hyunjin and Mehrdad, for attending my qualifying and defense dry-runs to give me feedback.

I also want to thank the members of our geo-integration group, including Dr. Jose Luis Ambite, Snehal Thakkar, Martin Michalowski, Yao-Yi Chiang and Dan Goldberg, for their discussions in my research. In particular, I thoroughly enjoyed my collaboration with Snehal in the beginning of my research. He gave me valuable feedback in various aspects in this project. I also appreciate Yao-Yi's assistance to implement map-imagery rubber-sheeting techniques and to improve the map intersection detector.

Also, I would like to thank to my family for their encouragement and support. They gave me the courage to pursue my dream. Finally, to my dear wife Chia-Hsin and my son Chih-Yang. My wife has come to Los Angels to accompany with me and to inspire me for years. Her love, encouragement, support and advice over the years mean more to me than I know how to express.

This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0238560 (PECASE), and IIS-0324955 (ITR), in part by the Air Force Office of Scientific Research under grant numbers F49620-01-1-0053 and FA9550-04-1-0105, in part by a gift from the Microsoft Corporation, and in part by a grant from the US Geological Survey (USGS) under the grant number 03CRSA0631.

Contents

List of Tables

List of Figures

Abstract

Recent growth of the geospatial information on the web has made it possible to access various spatial data. By integrating diverse spatial datasets, one can support the queries that could have not been answered given any of these sets in isolation. However, accurately integrating different geospatial data remains a challenging task because diverse geospatial data may have different projections and different accuracy levels. Most of the existing conflation algorithms only handle vector-vector data integration or require human intervention to accomplish vector-raster or raster-raster data integration. In this dissertation, I propose an approach, named AMS-Conflation, that achieves automatic geospatial data integration by exploiting multiple sources of geospatial information. In particular, I focus on vector-imagery and map-imagery conflation. For vector-imagery conflation, I describe techniques to automatically generate control points by exploiting the information from the road vectors to perform localized image processing on the imagery. I also evaluate various filtering algorithms to eliminate inaccurate control point pairs. Based on the experimental results, these techniques automatically align the roads to orthoimagery, such that in one of my experiments, 85% of the conflated roads are within 4.5 m from the real road axes compared to 55% for the original roads for partial areas in St. Louis, MO. For map-imagery conflation, my approach can take a map of unknown coordinates and automatically align it with an image. My approach first aligns road

vectors with imagery using vector-imagery conflation techniques to generate control points on the imagery. For the maps, my approach utilizes image processing techniques to detect intersections. Furthermore, I present an algorithm (called GeoPPM) to compute the matched point pattern from the two point sets. The experimental results show that GeoPPM only misidentified one point pattern from the fifty tested maps. The experimental results also show that my approach can align a set of TIGER maps with imagery for an area in St. Louis, MO, such that 85.2% of the conflated map roads are within 10.8 m from the real road axes compared to 51.7% for the original and geo-referenced TIGER map roads.

.

Chapter 1

Introduction

1.1 Motivation and Problem Statement

With the rapid improvement of geospatial data collection techniques, the growth of Internet and the implementation of Open GIS, a large amount of geospatial data are now readily available on the web. The examples of well-known vector datasets are US Census TIGER/Line files[1] (covering most roads over the United States), NAVSTREETS from NAVTEQ[2] and the GDT data from Geographic Data Technology.[3] The National Map,[4] ESRI's Geography Network,[5] MapQuest,[6] Yahoo Map Service,[7] Google Map Service,[8] Microsoft TerraService[9] [4] and Space Imaging[10] are good examples of map or satellite imagery repositories. The users of these data products often want these geospatial data and other related data to be displayed in some integrated fashion for knowledge discovery. Instead of simply being able to display all of the related data in a single framework, we need to

[1] http://www.census.gov/geo/www/tiger/
[2] http://www.navteq.com/
[3] http://www.geographic.com/
[4] http://seamless.usgs.gov
[5] http://www.geographynetwork.com/
[6] http://www.mapquest.com
[7] http://maps.yahoo.com/
[8] http://maps.google.com
[9] http://terraserver-usa.com/
[10] http://www.spaceimaging.com/

actually fuse the data to provide additional inferences that cannot be gathered from any single information source.

In fact, geospatial data fusion has been one of the central issues in GIS[11] [42]. Geospatial data fusion requires that the system integrates various datasets, and then creates a single composite dataset from the integrated elements. Towards geospatial data fusion, a vital step is reducing the spatial inconsistencies among multiple datasets. Figure 1.1 shows an example of combining a road network (NAVTEQ NAVSTREETS) and an image (geo-referenced USGS color imagery with 0.3 m per pixel resolution). By simply superimposing the roads on top of the imagery, certain geospatial inconsistencies become noticeable (as shown in Figure 1.1(a)). An alignment of the imagery with the road network of the area (as Figure 1.1(b)) is useful to annotate streets in the imagery with detailed attribution information often contained in vector datasets. Moreover, once the road network is aligned to higher resolution imagery (e.g., 0.3 m/pixel), its relatively poorer positional accuracy can be improved. Another example demonstrating the advantages of map and imagery integration is shown in Figure 1.2. The user can view the imagery of an unknown area nearby and can identify a park in the imagery (as in Figure 1.2(a)). However, the imagery does not provide street names. An integrated view of the imagery with the map of the area (see Figure 1.2(c)) results in an intelligent image that combines the visual appeal and accuracy of imagery with the detailed attribution information contained in the map.

[11] http://www.cobblestoneconcepts.com/ucgis2summer2002/researchagendafinal.htm

a) Without alignment
(black lines: NAVTEQ NAVSTREETS roads)

b) With alignment
(roads are annotated with vector information)

Figure 1.1: An example of the integration of road vector data and imagery

a) Imagery with the area of interest highlighted

b) ESRI street map

c) Imagery with aligned map

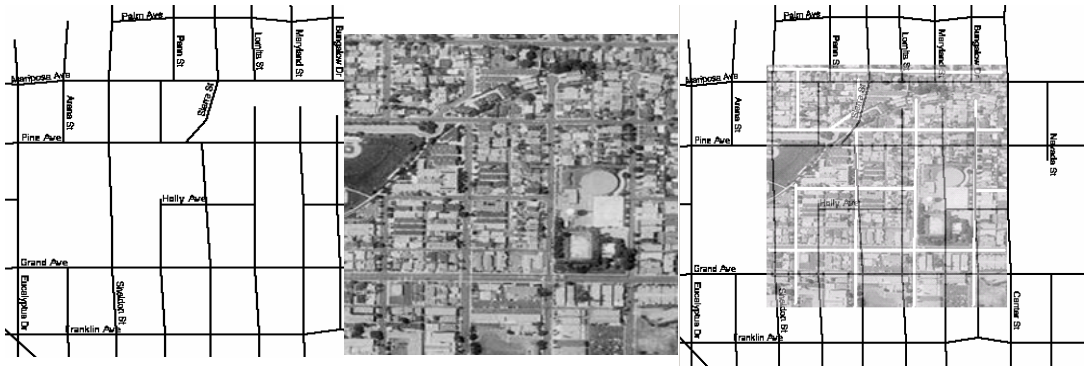
Figure 1.2: An example of the integration of street maps and imagery

One cannot rely on a manual approach to align diverse geospatial datasets, as the area of interest may be anywhere in the world and manually aligning a large region, such as the continental United States, is very time consuming and error-prone. Moreover, performing alignment offline on two geospatial datasets is also not a viable option in online GIS-related applications as both datasets may be obtained by

querying different information sources at run-time (i.e., the data of "area of interest" are often needed to be processed and integrated on-demand). However, automatically and accurately aligning geospatial datasets, such as road vector data, street maps and imagery, is a difficult task. The major challenges are:

- Different data products may not align. This is because they may have been collected at different resolutions; they may use different spheroids, projections or coordinate systems; they may have been collected in different ways or collected with different precision or accuracy, etc. If the geographic projections of both datasets are known, then both datasets can be converted to the same geographic projections. However, the geographic projection for a wide variety of geospatial data available on the web is not known. Furthermore, converting datasets into the same projection does not address the issue of different inaccuracies between two spatial datasets.
- There are few online maps with known geo-coordinates. For example, many online street maps, such as Yahoo Map Service and MapQuest, are not geo-referenced. Hence, their geo-coordinates are not known in advance, although the general region is provided. Even though some map services, such as TIGER maps,[12] provide the geo-coordinates, they often do not align with imagery, due to the reasons mentioned above (as shown in the example in Figure 1.3).

[12] http://tiger.census.gov/cgi-bin/mapsurfer



a) Georeferenced TIGER street map

b) Georeferenced imagery

c) Imagery with superimposed map (the roads on imagery are represented as white lines)

Figure 1.3: The map-imagery integration without alignment

Conflation is often the process used to integrate or align different geospatial datasets[13]. The conflation process can be divided into the following subtasks: (1) feature matching: find a set of conjugate point pairs, termed control point pairs, in two datasets, (2) match checking: detect inaccurate control point pairs from the set of control point pairs for quality control, and (3) alignment: use the accurate control points to align the rest of the geospatial objects (such as points or lines) in both datasets by using the triangulation and rubber-sheeting techniques. In particular, finding accurate control point pairs is a very important step in this kind of feature-based conflation process as all the other points in both datasets are aligned based on the control point pairs.

Essentially, the related work for geospatial data conflation techniques can be categorized to three groups:

[13] In this thesis, I use the terms conflation, integration and alignment interchangeably.

- Vector to vector data conflation (e.g., road vector to road vector conflation)
- Vector to raster data conflation (e.g., road vector to satellite imagery conflation)
- Raster to raster data conflation (e.g., map to satellite imagery conflation)

There have been a number of efforts to automatically or semi-automatically accomplish vector to vector conflation. These approaches utilize different methods to locate the counterpart elements. However, due to the complexity that characterizes natural scenes in imagery, there has been relatively little work on automatically conflating vector with imagery or maps with imagery. These days, advances in satellite imaging technology are making it possible to capture imagery with ever increasing precision. Remotely sensed images from space can offer a resolution of 0.3 m or better. Utilizing vector to imagery conflation, we can use this accurate imagery to update the vector datasets with poorer positional accuracy but richer attribution. Moreover, utilizing map to imagery conflation, we can create intelligent images that combine the visual appeal and accuracy of imagery with the detailed attribution information often contained in diverse maps.

Traditionally, the problems of vector-imagery and map-imagery conflation have been in the domain of image processing and GIS. The focus of the image processing techniques has been on automatic identification of objects in the image in order to resolve vector-image or map-image inconsistencies. However, these techniques require significant CPU time to process an image in its entirety and still may result in

inaccurate results. Moreover, various GIS systems, such as ESRI ArcView[14], MapMerger[15], ER Mapper Image Web Server[16], Intergraph I/RASC[17] and Able R2V[18] provide the functionality to integrate different layers of geospatial data. However, these products do not provide automatic conflation, and human intervention is required to align multiple geospatial datasets.

This thesis is motivated by the lack of automatic, efficient and accurate vector to imagery and map to imagery conflation techniques for GIS applications, which can work on large regions retrieved on-demand. In particular, I consider the conflation of road vector data and orthoimagery (i.e., this imagery is altered from original photos so that it has the geometric properties of a map) and the conflation of street maps (i.e., maps showing roads) and orthoimagery.

1.2 Approach

This thesis provides novel and efficient solutions to automatically integrate road vector data, high resolution orthoimagery (with ground resolution greater than 1 m/pixel) and street maps by reducing their spatial inconsistencies.

Based on the conflation process discussed in the previous section, an automated conflation algorithm should have the capabilities to automatically and accurately detect control points from both datasets, compute the transformations between the control points, and then transform other objects accordingly. In this section, I

[14] http://www.esri.com/
[15] http://www.esea.com/products/
[16] http://www.earthetc.com/
[17] http://imgs.intergraph.com/irasc/
[18] http://www.ablesw.com/r2v/

describe my solution, a geospatial information integration approach, named Automatic Multi-Source conflation (AMS-conflation) [8, 9, 11], to automatically integrate vector data, imagery and maps. AMS-conflation automatically exploits information from each of the sources to be integrated to assist the alignment process.

Essentially, there are three general sources of information for automatically identifying control points: inferences on the data source, metadata and attribution information of the data sources and other sources of data that can be linked to the source. I consider each of these in turn.

- **Inferred Information from the Data Source:**

  There are a wide variety of techniques that can be applied to geospatial data sources to identify possible control points. For example, road network data can be analyzed to find intersection points. Similarly, image processing techniques (such as edge-detection, corner-detection, region-segmentation and histogram-based classification) can be used on both images and maps to find points, lines or homogeneous regions. Any of these image processing techniques may result in some missing points or noisy points, but these problems can be addressed since conflation algorithms only need a sparse distribution of accurate points to conflate two datasets.

- **Metadata and Attribution Information of the Data Source:**

  Recently, metadata (i.e., information about data) is used increasingly in geographic information systems to improve both availability and the quality of the spatial information delivered. I can exploit metadata from vector data,

imagery and maps to perform the automatic conflation procedure. For example, for imagery, metadata about the coordinates of data source and resolution is important in narrowing the search for corresponding control points. For maps, I can exploit the map scale, map resolution, map legend and map orientation to improve the performance of locating corresponding control points. Attribution information of the geospatial datasets can also provide helpful information to identify control points. For example, I can exploit (non-spatial) attributes of the road vector data, such as address ranges, road names, or even the number of lanes and type of road surface to efficiently and accurately locate control points.

- **Related Information to the Data Source:**

  Another important source of information includes peripheral data sources that can provide additional information. For example, I can use telephone books to look up the address of a named or pre-identified point in the imagery. Similarly, consider conflation of map and imagery. I can utilize a third data source (e.g., road vector data) that has relevant information to both sources to support map and imagery alignment.

My approach can automatically exploit these various sources of information to accurately identify spatial features, such as road intersections, as control points. More precisely, I consider two scenarios, vector-imagery conflation and map-imagery conflation in turn.

Vector and Imagery Conflation:

I can find the approximate location of road intersections on the images from the inferred knowledge of the corresponding vector data. For each intersection point, I perform image processing in a small area around the intersection point to find the corresponding point in the image. In addition to the approximate location of intersections, I also utilize the information inferred from vector data such as road-directions, road-widths and road-shapes, to locate intersections on the images. In particular, my approach generates a template inferred from all the vector information and then matches it against the small area in the image to find the corresponding intersection point on the imagery. The running time for this approach is dramatically lower than traditional image processing techniques because I can localize the image processing. Furthermore, the road direction and width information makes detecting roads in the image a much easier task, thus reducing the running time even further.

Map and Imagery Conflation:

I can utilize auxiliary information sources (i.e., road vector data) that are not part of a map or an image, but have information relevant to both sources. In other words, I utilize the road vector data as "glue" to align maps and imagery. First, I align road vector data with imagery using my vector-imagery conflation techniques. As a result the conflated intersection points on the road network are aligned with the intersection points on the imagery. I can then use the conflated intersection points as control points on the imagery. For the maps, I can utilize image processing techniques to detect promising points (e.g., road intersections) as control points. Furthermore, I

compare the distributions of the two point sets by exploiting road directions, map scales, etc, to determine the transformation between the map and imagery.

I evaluate my approach by presenting results of the experiments performed on the real-world datasets to show that my approach can automatically align road vector datasets (with diverse accuracy levels), various orthoimagery (with different ground resolutions) and diverse street maps (with different map scales).

1.3 Thesis Statement

In this dissertation, I propose a novel and efficient approach to automatically and accurately conflate road vector data, street maps and orthoimagery. The thesis of this dissertation is:

> **By exploiting multiple sources of geospatial information, we can achieve automatic and accurate conflation of road vector data, street maps and orthoimagery.**

1.4 Contributions

Overall, I make the following contributions in this dissertation:

- An efficient approach for vector to imagery conflation that exploits knowledge from each of the sources to be integrated to automatically identify a set of accurate control points. This approach can support automatic, rapid and accurate alignment of various vector datasets and imagery. The approach

is appropriate for those GIS applications that want to align vector data and imagery on large regions retrieved on-demand.

- An approach for map to imagery conflation that utilizes common vector datasets as "glue" to automatically integrate street maps with imagery. The approach makes it possible to automatically and accurately create intelligent images that combine the visual appeal and accuracy of imagery with the detailed attribution information often contained in diverse maps. This approach is also applicable for those GIS applications that want to align online maps and imagery for the regions retrieved on-demand.

1.5 Thesis Organization

The remainder of this dissertation is organized as follows:

Chapter 2 describes my automatic vector to imagery conflation approach. Chapter 3 describes my automatic map to imagery conflation approach. Chapter 4 reviews the related work on geospatial data alignment. Finally, Chapter 5 concludes this dissertation and discusses possible future work in this area.

Chapter 2

Automatic Conflation of Road Vectors and Orthoimagery

My AMS-conflation is a multi-step geospatial data alignment process that involves identification of matching features as control points, filtering of misidentified control points and computation of local transformations between two datasets based on filtered control points. In this section, I will describe my detailed techniques to automatically and accurately conflate road vector data and orthoimagery. I will also present my evaluation methodology and provide results of utilizing my approach to align real world data.

Figure 2.1 shows the overall approach for conflating road vector data and orthoimagery. First, my approach automatically exploits valuable information, the attributes of the road vector data and the metadata of the imagery to find the control point pairs. Next, my approach filters the control points. Then, it utilizes triangulation and rubber-sheeting to align the vector with imagery.[19]

2.1 Finding Control Points

A control point pair consists of a point in one dataset and a corresponding point in the other dataset. Finding accurate control point pairs is a very important step in the

[19] A pseudo code in Appendix A presents the vector to imagery conflation algorithm.

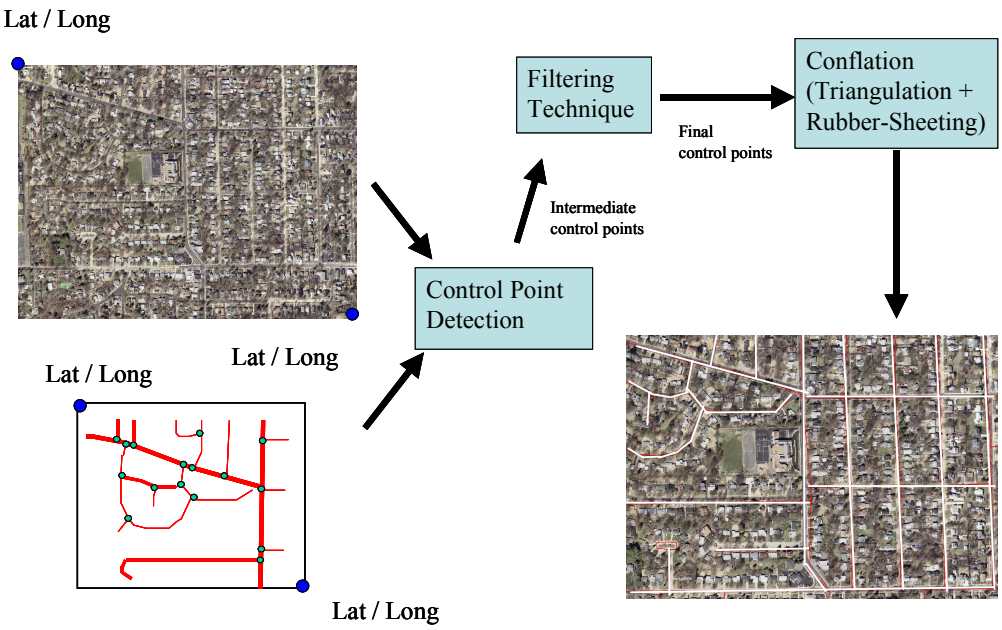


Figure 2.1: Overall approach to align vector with orthoimagery

conflation process as all the other points in both datasets are aligned based on the control point pairs.

Road intersections are good candidates for being control points, because road intersections are salient points to capture the major feature of the road network and the road shapes around intersections are often well-defined. In addition, various GIS researchers and computer vision researchers have shown that the intersection points on the road networks are good candidates to be identified as an accurate set of control points [10, 11, 18, 21]. In fact, several image processing algorithms to detect roads in the imagery have been utilized to identify intersection points in the imagery. Unfortunately, extracting road segments directly from imagery is a difficult task due to the complexity that characterizes natural scenes [1, 23]. Thus, extracting roads

from imagery is error-prone and may require manual intervention. Moreover, processing an image of a large area to extract roads requires substantial processing time.

Integrating vector data into the road extraction procedures alleviates these problems. In this section, I describe my technique, called localized template matching (LTM) [10], which takes advantage of the vector data attributes, image metadata as well as the color of imagery to accurately and efficiently find the intersection points of various roads in the imagery. Conceptually, the spatial information on the vector data represents the existing knowledge about the approximate location of the roads and intersection points in the imagery. My approach improves the accuracy and running time of the algorithms to detect intersection points in the image by utilizing the knowledge from the vector data. First, the LTM technique finds all the intersection points on the vector data. For each intersection point on the vector data, the LTM technique determines the general area in the image where the corresponding intersection point should be located. Finally, a template inferred from the vector information (such as road width, road directions and road intersections) is matched against this small area to identify the intersection points on the imagery. The area searched for the intersection is a small fraction of the entire image.

The LTM technique may not be able to find all intersection points on the image due to the existence of trees or other obstructions, such as cars and building shadows. However, the conflation process does not require a large number of control point

pairs to perform accurate conflation. Therefore, for a particular intersection point on the vector data, if LTM cannot find corresponding image intersection point within the given area, it will not greatly affect the conflation process. The entire process of locating road intersections in imagery using road network data is shown in Figure 2.2. I discuss the detailed procedure in the following sections.

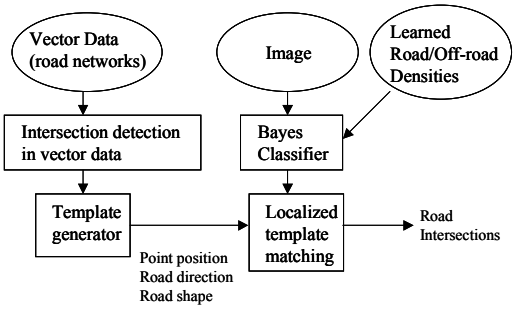


Figure 2.2: Road intersection identification

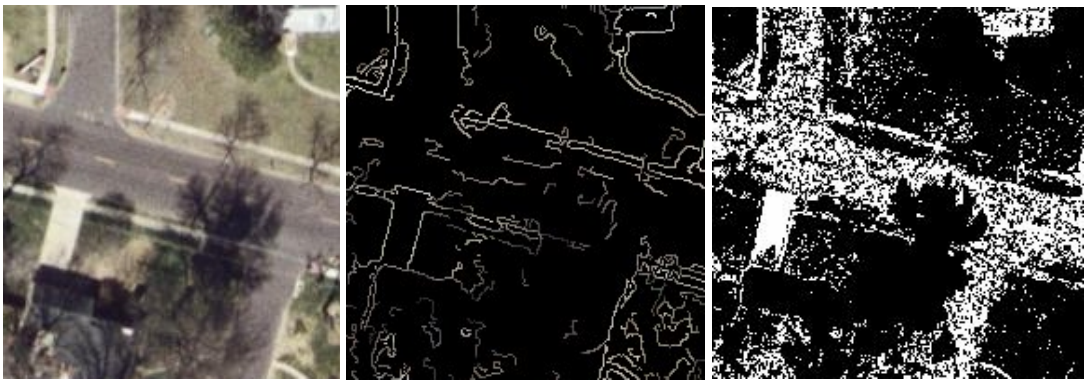
2.1.1 Road Networks Intersection Detection

The process of finding the intersection points on the road network from the vector data is divided into two steps. First, all candidate points are obtained by examining all line segments in the vector data. In this step, the endpoints of each line segment in the vector data are labeled as the candidate points. Second, the connectivity of these candidate points is examined to determine if they are intersection points. In this step, each candidate point is examined to see if there are more than two line segments connected at this point. If so, this point is marked as an intersection point and the directions of the line segments that are connected at the intersection point are calculated.

2.1.2 Imagery Road Intersection Detection

Towards the objective of identifying road intersections on imagery, the vital step is to understand the characteristics of roads on imagery. In low resolution imagery, roads are illustrated as lines, while in high resolution imagery, roads are exposed as elongated homogeneous regions with almost constant width and similar color along a road. In addition, roads contain quite well-defined geometrical properties. For example, the road direction changes tend to be smooth, and the connectivity of roads follows some topological regularities.

Road intersection can be viewed as the intersection of multiple road axes. Also, it is located at the overlapping area of some elongated road regions. These elongated road regions form a particular shape around the intersection. Therefore, I can match this shape against a template derived from road network data (discussed next) to locate the intersection. Based on the characteristics of roads, this shape is formed from either detected road-edges or homogeneous regions. Edge detectors (such as [32]) could be utilized to identify linear features on imagery to match against vector data to locate intersections. However, on high resolution imagery (such as up to 0.3 m/pixel, as an example shown in Figure 2.3(a)), more detailed outlines of spatial objects, such as edges of cars and buildings, introduce noisy edges. Hence, in some cases I may obtain fragmented edges that include real road edges, building edges, tree edges, etc. (see Figure 2.3(b)). This makes it a difficult task to identify real road edges by grouping-based methods (i.e., methods that group detected pixels belonging to the same edge as a line or a curve, such as the approach proposed in [41]). As the



a) High resolution color orthoimagery

b) Detected edges by Canny edge detector

c) Road-classified image (white pixels: image pixels pre-classified as roads)

Figure 2.3: An example of edge-detected image and road-classified image

example shown in Figure 2.3(b), more constraints (such as the road sides of the same roads are often in parallel) must be exploited to eliminate the impacts of noisy edges. However, I can make use of other useful information about roads, such as the color of roads, to overcome this problem. Therefore, in contrast to traditional edge-detection approach, I propose a more effective way to identify intersection points on high resolution imagery. My approach utilizes the Bayes classifier, a histogram based classifier [19, 28], to classify an images' pixels as on-road or off-road pixels (as in Figure 2.3(c)). I now describe these algorithms in detail.

2.1.2.1 Labeling imagery using the Bayes classifier

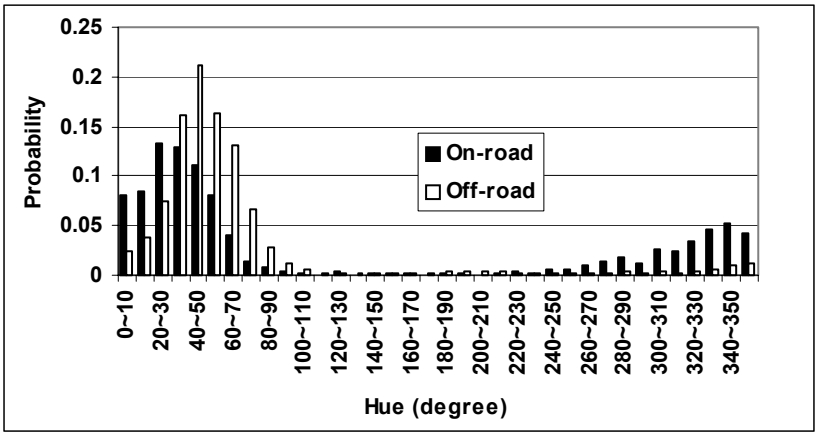
The histogram-based classification (Bayes classification) is based on the assumption of consistency of image color on road pixels. That is, road pixels can be dark, or white, or have color spectrum in a specific range, however; for the imagery set whose images were taken around the same time period using similar remote sensing equipments, I expect to find the same representative color on nearby road pixels. I

construct the statistical color distribution (called class-conditional density) of on-road/off-road pixels by utilizing a histogram learning technique as follows. I first randomly select a small partial area from the imagery where I intend to identify road intersections. Then, I interactively specify on-road regions and off-road regions respectively. From the manually labeled training pixels, my system learns the color distribution (histograms) for on-road and off-road pixels. Hence, I can construct on-road and off-road densities.

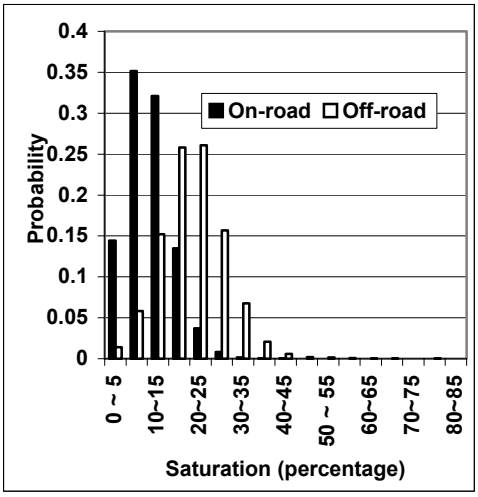
The off-line learning process requires manual labeling to obtain conditional density functions, but it is performed only when a new imagery dataset is introduced to the system. In addition, my system can apply the learned results to automatically identify intersections of an area that is much larger than the area my system learns from. In order to determine whether new training is needed for a given image or the current learning is sufficient for the classification of the new area, in the future, I plan to develop an automatic approach based on the statistical analysis of the color distribution of the target imagery.

Figure 2.4 shows the hue probability density and saturation probability density[20] after conducting the learning procedure on nearly 500 manually labeled rectangles from a set of USGS 0.3 m/pixel imagery (covering St. Louis County in Missouri of the United States). There are 50,000 pixels covered by these rectangles and it took about 2.5 hours to perform the labeling process.

[20] I eliminated the intensity (i.e. brightness of HSV model) density function. There is no obvious difference between the brightness distribution of on-road and off-road pixels, since these images were taken at the same time (i.e., under similar illumination conditions).



a) Hue density function



b) Saturation density function

Figure 2.4: Learned density function on HSV color space for On-road/Off-road pixels

Consider the hue density function on Figure 2.4(a). It shows the conditional probabilities *Probability(Hue/On-road)* and *Probability(Hue/Off-road)*, respectively. The X-axis of this figure depicts the hue value grouped every 10 degrees. The Y-axis shows the probability of on-road (and off-road) pixels that are within the hue range represented by the X-axis. For a particular image pixel, my system can compute its hue value *h*. Given the hue value *h*, if the probability for off-road is higher than on-road, my system would predict that the pixel is an off-road pixel. As shown in Figure

2.4, these density functions depict the different distribution of on-road and off-road image pixels on hue and saturation dimensions, respectively. Hence, I may use either of them to classify the image pixels as on-road or off-road. In my experiments, I utilized the hue density function for classification. In general, I can utilize the two chromatic components, hue and saturation, together.

Based on the learned hue density functions, automated road-labeling is conducted as follows. A particular image pixel whose hue value is h is classified as road if $\frac{Probability\,(h / On\text{-}road)}{Probability\,(h / Off\text{-}road)} \geq \theta$, where θ is a threshold. θ depends on the application-specific costs of classification errors and it can be selected using the ROC (Receiver Operating Characteristic) technique discussed in [28].

Since the system knows the approximate intersection locations on the images from the road network data (discussed next), the road-labeling procedure is applied only to image pixels within a radius of potential intersections. Therefore, my system does not need to exhaustively label each pixel on the entire image.

2.1.2.2 Analyzing imagery using road network data (Localized Template Matching)

Using the classified image (an example is shown in Figure 2.5(b)(c)) as input, my system can now match it with a template determined from the road network data to identify intersections.

a) Layout (left) of the original road network data around an intersection point and template (right) inferred by using the road network data

b) Original image

c) Road-labeled image (white pixels: labeled road pixels; black lines: existing road network data; black circles: intersections on vector data, implying approximate locations of intersections on imagery)

Figure 2.5: An example of the localized template matching

First, my LTM technique finds the geo-coordinates of all the intersection points on the road network data. Since the system also knows the geo-coordinates of the images (from image metadata), it can obtain the approximate location of intersections on the imagery (as in Figure 2.5(c)). For each intersection point on the road network data, LTM determines the area in the image where the corresponding intersection point should be located by picking a rectangular area (with width W and height H) in the image centered at the location of the intersection point from the road network data. Meanwhile, as an example shown in Figure 2.5(a), a template (with width w and height h) around an intersection on road network data is generated by the presence of regions inferred from the road network data using information, such as the road directions and road widths. LTM will then locate regions in the road-labeled image (see Figure 2.5(c)) that are similar (in shape) to the generated template as follows. Given a road-labeled image *I* with W x H pixels and a template *T* with w x h pixels, the system moves the template around the image and compares the

template against the overlapped image regions. My adapted similarity measure is a normalized cross correlation defined as:

$$C(x,y) = \frac{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} T(x',y')I(x+x',y+y')}{\sqrt{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} T(x',y')^2 \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} I(x+x',y+y')^2}} \qquad \text{Eq.(2.1)}$$

where *T(x,y)* equals one, if *(x,y)* belongs to a road region, otherwise; *T(x,y)* equals zero. *I(x,y)* equals one, if *(x,y)* is pre-classified as a road pixel; otherwise, *I(x,y)* equals zero. C(x,y) is the correlation on the pixel (x,y). In my implementation, I set W equal to H (i.e., a square area).

The highest computed correlation C(x,y) implies the location of the best match between the road-labeled image and the template. Furthermore, C(x,y) determines the degree of similarity between the matched road-labeled image and the template. An intersection will be identified, if C(x,y) is greater than a similarity threshold *t (0 ≤ t ≤ 1.0)*. When setting *t* to 0.5, I can keep the detected intersections that have higher similarity value (i.e., C(x,y)) than its dissimilarity value (i.e., 1.0 - C(x,y)). Hence, in my experiment, I set the threshold *t* to 0.5. Moreover, the square area dimension (i.e., the width W) can be determined based on the accuracy and resolution (such as ground resolution from image metadata) of the two datasets. In my implementation, I conduct experiments using various sizes and select the size that has the best performance. As an example, shown in Figure 2.6, I utilized a high quality road network, NAVTEQ NAVSTREETS, and the LTM technique with

various area sizes to identify intersections in a 1.5 km by 1.5 km USGS high resolution color imagery (with 106 actual road intersections).

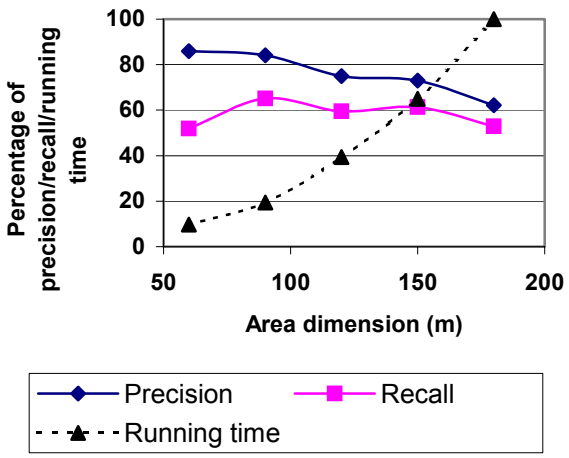


Figure 2.6: The impact of area dimension (dimension is increased by 30 m)

Figure 2.6 shows the performance of LTM by applying a "buffer method" to calculate recall and precision of identified intersections. Road intersection can be viewed as the intersection of multiple road axes. Also, it is located at the overlapping area (called buffer) of these elongated road regions. Identified road intersections that fall within the buffer are considered as "correctly identified intersections". Using this term, I define:

$$\text{Recall} = \frac{\text{Number of correctly identified intersections}}{\text{Number of intersections in the image}} \qquad \text{Eq.(2.2)}$$

$$\text{Precision} = \frac{\text{Number of correctly identified intersections}}{\text{Number of identified intersections}} \qquad \text{Eq.(2.3)}$$

As shown in Figure 2.6, I increase the area dimension from 60 m (i.e., 200 pixels in a 0.3 m/pixel image) to 180 m with the incremental dimension value 30 m. Then, I calculate the precision and recall. I also compute the normalized intersection detection running time (with respect to the running time of using 180 m as area dimension). The results show that the detection time dramatically increases as area dimension increases.

As shown in Figure 2.6, the precision decreases when the area dimension increases. This is because that larger area may involve more road intersections that have similar shape as the road template (e.g., some urban areas where roads are sometimes distributed in a grid shape). In addition, there could be more misclassified pixels (such as the house roof pixels which have similar color to road pixels and might be classified as on-road pixels) for a larger area, thus detecting some incorrect intersections. Due to the same reason, the recall also slightly decreases as the area dimension increases from 90 m to 180 m. However, I obtained lowest recall when setting the dimension size to 60 m. This implies that dimension 60 m is not large enough to capture most of the positional displacements between the vector and imagery. Therefore, based on these experimental results, I can select 90 m as the area dimension to identify intersections as control points on other neighbooring areas. This is because setting the area dimension to 90 m, the system achieved 84% precision and 64% recall. Although it is slightly smaller than the precision (86%) obtained using 60 m as area dimension, I obtain much better recall (64% vs. 52%).

The histogram-based classifier, as illustrated in the previous section, may generate fragmented results due to noisy objects, such as cars, tree-clusters and building shadows on the roads. Furthermore, some non-road objects whose color is similar to road pixels might be misclassified as roads. However, LTM can alleviate these problems by avoiding exhaustive search of all the intersection points on the entire image and usually locates the intersection point on the image that is the closest intersection point to the intersection point on the road network data. Moreover, this

technique does not require a classifier to label each pixel for the entire region. Only the areas near the intersections on the image need to be pre-classified. In addition, when utilizing localized template matching, it implicitly implies that the topology constraint (such as adjacency) is considered. This is because the template is generated based on the connectivity of several road segments merging at the same intersection point.

Note that the objective of the histogram-based classifier is not to correctly classify every single pixel of the image as off-road or on-road. Instead, as long as a majority of on-road pixels are identified so that the intersection-shape on the image is captured, LTM can successfully match it to the corresponding vector template. Even in the worst case, if the system misses an entire intersection, still the entire conflation process may be successful as long as a sufficient number of intersections are identified.

In sum, the running time of my LTM technique is dramatically lower than traditional image processing techniques, because image processing is performed on localized areas. Furthermore, exploiting the road direction information improves both the accuracy and efficiency of detecting road intersections in the image. Figure 2.7 shows an image indicating the

Figure 2.7: The intersections (rectangles) on road network data and the corresponding intersections (circles) on imagery

intersection points on road network data and the corresponding intersection points identified on imagery.

2.2 Filtering Control Points

The localized template matching may take misclassified pixels (such as the house roof pixels, which have similar color to road pixels) as on-road pixels, thus producing some inaccurate control point pairs. For example, Figure 2.8 shows the intersections (rectangles) on road network data and the corresponding intersections (circles) on imagery. The detected (and highlighted) control point pairs 1, 2 and 3 are inaccurate control point pairs.

Figure 2.8: Some inaccurate control point pairs

Because the conflation algorithm utilizes the control point pairs to align the vector data with the image, the inaccurate control point pairs reduce the accuracy of the alignment between two datasets. Therefore, it is very important to filter out inaccurate control point pairs. To address this issue, I can exploit the fact that there is a significant amount of regularity in terms of the relative positions of the controls points across data sets. This is due to the fact that my system is not trying to correct

individual errors, but rather to determine some local transformations across datasets that allow the system to integrate two separate data sources. More precisely, while there is no global transformation to align imagery and vector data, in small areas the relationship between the points on the imagery and the points on the vector data can be described by a transformation. The transformation can be attributed to different projections, accuracies, or coordinate systems used in the imagery data and the vector data. Due to the above-mentioned nature of the datasets, in a small region the control points on the imagery and the counterparts on vector data should be related by similar transformations. Therefore, the inaccurate control point pairs can be detected by identifying those pairs with significantly different relationship as compared to the other nearby control point pairs.

I explored a set of filtering techniques to filter out the inaccurate control point pairs. Section 2.2.1 describes the first filter, vector median filter (VMF) [3]. Section 2.2.2 discusses the other filter, distance-based outliers detector (DB-outliers detector) [30]. Both VMF and DB-outliers detector utilize the same underlying property (that is, there is a significant amount of regularity in terms of the relative positions of the controls points across data sets), but use different mathematical models to accomplish filtering.

2.2.1 Vector Median Filter (VMF)

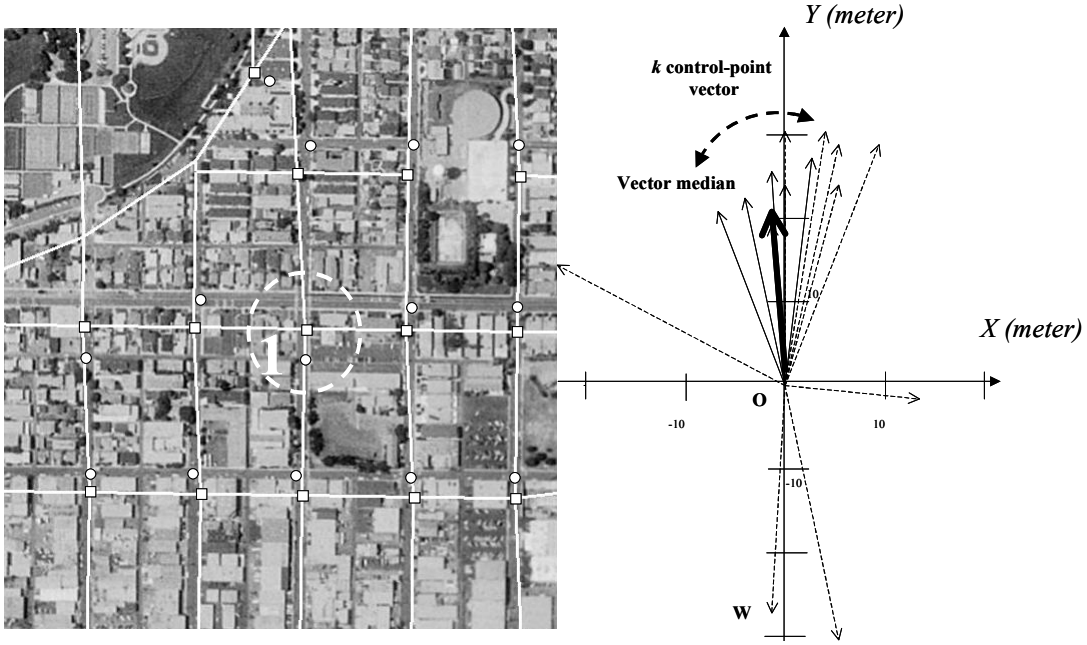
Vector Median Filter (VMF) [3] is a mathematical tool for signal processing to attenuate noise, and it is a popular filter to accomplish noise removal in image

processing. The VMF views the data points as vectors and filters out the data point with vectors significantly different from the median vector.

The relative position of the points of each control point pair can be viewed as a vector, termed control-point vector. Assuming that N control point pairs are detected in a small area by LTM technique. Hence, there are N control-point vectors denoted as $\{\vec{x}_i \mid \vec{x}_i = \overrightarrow{P_iQ_i}$ (i= 1, 2, 3, … N), where the tail P_i is an intersection point on the vector dataset, and the head Q_i is the (detected) corresponding point on the imagery$\}$.

Since vectors are invariant under translation, it is convenient to consider the tail P_i as located at the origin. Hence, the tail of each control-point vector coincides to the same origin. In the example reproduced in Figure 2.9, the control-point vectors for the seventeen detected control point pairs of Figure 2.9(a) are shown in Figure 2.9(b) as arrows (vectors). Because the spatial inconsistencies between the imagery and vector data in a local area are similar, the control-point vector whose direction and magnitude are significantly different from others is characterized as an inaccurate control-point vector (i.e., an outlier). Due to the similarities of these control-point vectors, their directions and magnitudes can be represented by the vector median. I modified the vector median filter to assist my system in identifying the control-point vectors that are significantly different. The system uses this filter to obtain the best matching set of control points.

The vector median of these N vectors $\vec{x}_i$ (i= 1, 2, 3, … N) can be defined as the vector $\vec{x}_{vm}$ such that



a). The distribution of seventeen control point pairs in a small area centered at the control point pair 1

b). The distribution of seventeen control-point vectors

Figure 2.9: VMF filter

1) The sum $\sum_{i=1}^{N} \| \vec{x}_{vm} - \vec{x}_i \|$ is minimized. Here $\| \ \|$ stands for L_2 norm (Euclidean distance).

2) $\vec{x}_{vm} \in \vec{x}_i$; i= 1,2,3,…,N.

Vector median has similar properties as the median operation. Intuitively, the median vector is the vector that has the shortest summed distance (Euclidean distance) to all other vectors.

The inputs for a vector median filter are *N* vectors $\vec{x}_i$ (i= 1, 2, 3, … N) and the output of the filter is the vector median $\vec{x}_{vm}$. I revised the output of vector median filter to accommodate not only $\vec{x}_{vm}$, but also *k* closest vectors to the vector median. I defined the distance *D*:

$D = \| \vec{x}_k - \vec{x}_{vm} \|_2$ where $\vec{x}_k$ is the k-th closest vector to $\vec{x}_{vm}$.

Then, the output of my vector median filter is

$\{ \vec{x}_i \mid \text{where } \| \vec{x}_i - \vec{x}_{vm} \| \leq D \text{ and } i = 1, 2, 3, \ldots N \}$

As a result of the modified Vector Median Filter, the k closest vectors to the vector median are selected and the other control-point vectors are filtered out. The possible value of k is an integer between 1 and N. Large value of k provides more control-point vectors, but may not filter out all inaccurate control point pairs. Based on my experiments of tuning different values for k, the VMF filter performs well when setting k to $\lceil \frac{N}{2} \rceil$. Hence, the system kept the $k = \lceil \frac{N}{2} \rceil$ closest vectors to the vector median and filtered out the remainder of the control point pairs. As a result, some accurate control-point vectors may be lost. However, the missing control point pairs would not greatly affect the conflation results, as some of the selected control point pairs close to the lost accurate control point pairs have similar directions and displacements.

Figure 2.9 graphically shows how the VMF works. For example, to determine whether the control point pair *1* (in Figure 2.9(a)) is an outlier or not, its corresponding control-point vector would be compared to other control-point vectors nearby. The seventeen neighboring control-point vectors within a radius of 300 m to the control point pair *1* are shown in Figure 2.9(b) as the arrows. The thickest arrow is the vector median among these control-point vectors. After applying the modified Vector Median Filter, only nine (k=9) closest vectors to the vector median are not categorized as outliers. As shown in Figure 2.10, the circles mark the control point

pairs categorized as outliers by VMF. The control point pair *1* will be filtered out, because its corresponding control-point vector (represented as $\overrightarrow{OW}$ in Figure 2.9(b)) is categorized as an outlier. The system can repeat the same process to filter out other outliers.

Figure 2.10: Control point pair 1 is filtered out, after applying VMF on Figure 2.9(a).

2.2.2 Distance-based Outliers Detector

Intuitively, to locate misidentified control points is to find the control-point vector that is extreme to its neighbors. The way to measure the "degree of extremity " of a specific control-point vector to its neighbors is calculating the Euclidean distance between them. In the previous section, I use the vector median whose summed distance to other neighboring control-point vectors is minimum to filter out outliers. Instead of utilizing the vector median to detect misidentified control points, in this section, I describe an alternative way by utilizing a distance-based outliers detector (termed DB(*p,D*)-outliers detector) [30] (In fact, it is an example of spatial outlier detector proposed in [40]). For a dataset *T* with *N* objects, an object *O* in *T* is a DB(*p, D*)-outlier if at least a fraction *p* of the objects in *T* lies greater than distance *D* from *O* [30]. The DB(*p, D*)-outliers detector works based on the observation: the control-

point vectors that have similar directions and magnitudes tend to form some clusters that exclude the extreme control-point vectors.

For example, again, to determine whether control point pair *1* (in Figure 2.9(a)) is an outlier or not by using DB(*p, D*)-outliers detector, its corresponding control-point vector would be compared to other control-point vectors nearby. The seventeen neighboring control-point vectors within a radius of 300 m to the control point pair *1* are shown in Figure 2.9(b) as the arrows. Consider the head point *W* (see Figure 2.9(b)) of the control-point vector $\overrightarrow{OW}$ of the control point pair *1*. When setting *p* to 50% and *D* to 15 m, there are only 12% of head points of other control-point vectors within 15 m to the point *W*. Hence, the point *W* is characterized as an outlier, and its corresponding control point pair (i.e., control point pair *1*) is categorized as an outlier to be filtered out. The system can repeat the same process to filter out other outliers. The choice of *p* and *D* depends on the application-specific costs of outlier-detection errors and it can be determined by experiments.

Different configurations for *p* and *D* will result in different filtering results. Compared to VMF, there is no similar set of parameters that needs to be determined for VMF in advance, although I can adjust the parameter *k* of VMF to retain or filter out additional control points. However, consider a very inconsistent control-point vector set (e.g., none of the control-point vector has similar direction and magnitude). VMF is not appropriate to filter control points in this scenario, since it may misidentify an inaccurate control-point vector as the vector median. In contrast, with proper parameter configurations, the DB(*p,D*)-outliers detector can handle this

scenario. However, from my practical experiments on real world data, this extreme scenario is rather rare.

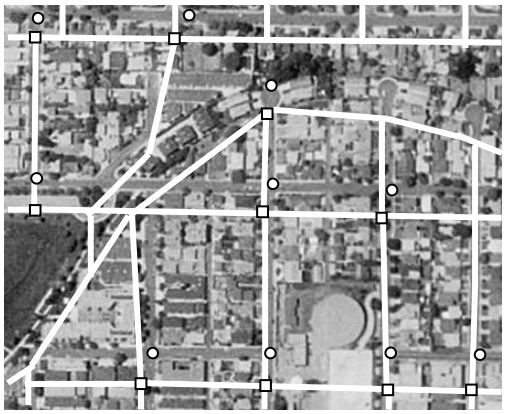
2.3 Aligning Road Vector Data with Imagery

After filtering the control point pairs, I have an accurate set of control point pairs for the imagery and vector data. Each pair of corresponding control points from the two datasets indicates corresponding positions on the datasets. Transformations are calculated from the control point pairs. Other points in both datasets are aligned based on these transformations. The Delaunay triangulation [5] and piecewise linear rubber sheeting [45] are utilized to find the appropriate transformations. The Delaunay Triangulation is discussed in Section 2.3.1, and rubber-sheeting is explained in Section 2.3.2. Moreover, a novel technique to reduce the spatial inconsistencies for the area where the system does not have any control point pairs is discussed in Section 2.3.3.

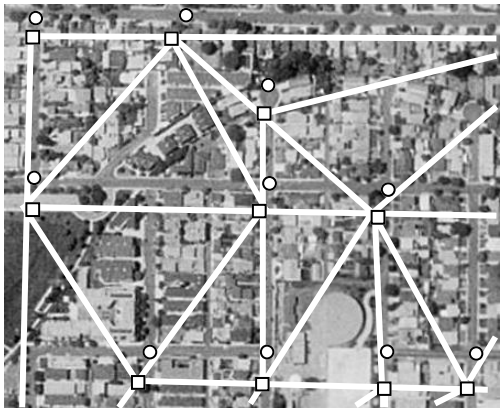
2.3.1 Space Partitioning Using Delaunay Triangulation

To achieve overall alignment of imagery and vector data, vector data must be adjusted locally to conform to the imagery. The system can align the two datasets based on local adjustments, because small changes in one area usually do not affect the geometry at long distances. To accomplish local adjustments, the domain space is partitioned into small pieces based on accurately identified control point pairs. Then, local adjustments are applied on each individual piece. Triangulation is an effective strategy to partition the domain space into triangles to define local adjustments.

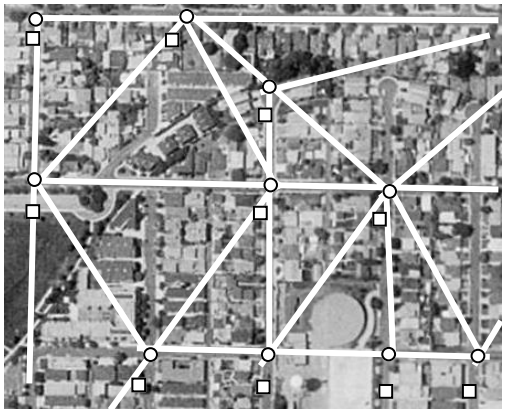
There are different ways to utilize the control points to partition the space into triangles. One particular type of triangulation, Delaunay triangulation, is especially suited for conflation systems [35]. A Delaunay triangulation is a triangulation of the point set with the property that no point falls in the interior of the circumcircle of any triangle (the circle passing through the three triangle vertices). The Delaunay triangulation maximizes the minimum angle of all the angles in the triangulation, thus avoiding triangles with extremely small angles [5]. My system performs the Delaunay triangulation with the set of control points on the vector data, and makes a set of equivalent triangles with corresponding control points on the imagery. Figure 2.11 shows an example of a resulting Delaunay triangulation on some detected control points.



a) The original road network and detected control point pairs (one is marked by rectangle and the corresponding point is represented as circle)



b) Delaunay triangulation based on detected control points on road vector data



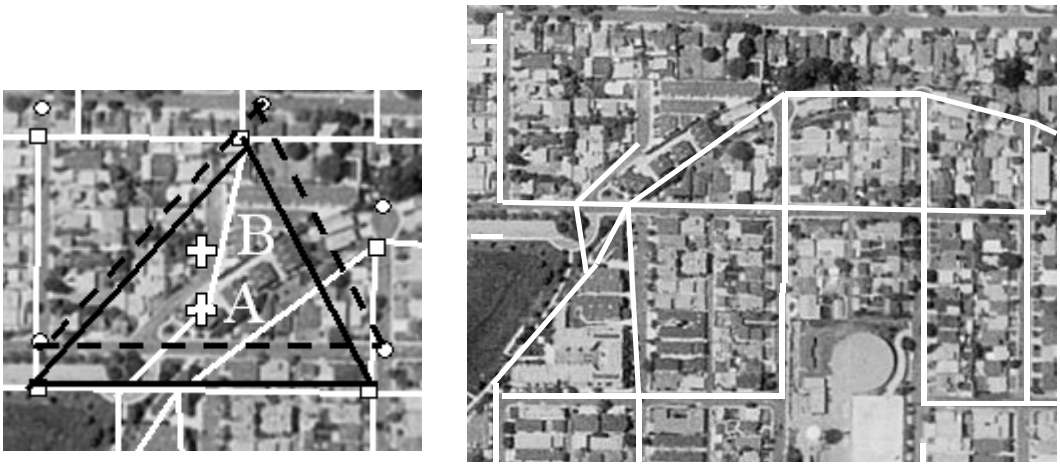
c) Corresponding Delaunay triangulation based on detected control points on image

Figure 2.11: An example of Delaunay triangulation

The Delaunay triangulation can be performed in O(n*log n) time in worst case, where n is the number of control point pairs. The details of the triangulation algorithms are discussed in [5, 26, 35].

2.3.2 Piecewise Linear Rubber-sheeting

Imagine stretching a vector map as if it were made of rubber. My system deforms algorithmically, forcing registration of control points over the vector data with their corresponding points on the imagery. This technique is called "Piecewise linear rubber sheeting" [45]. There are two steps to rubber sheeting. First, the transformation coefficients (i.e., the affine transformation that is composed of translation, rotation and scaling) to map each Delaunay triangle on vector data onto its corresponding triangles on the imagery are calculated. Second, the system applies the same transformation coefficients to transform the endpoints of each vector line segment within each Delaunay triangle to the imagery. Consider the example shown in Figure 2.12(a). White lines represent the road network. The rectangles are the control points on the road vector data and the circles are the corresponding control points on the imagery. The two triangles shown are Delaunay triangles formed by three corresponding control point pairs and one endpoint *A* of the original road segments is located within the triangle on the road vector data. The rubber sheeting technique transforms endpoint *A* to the point *B* on the imagery (*B* becomes an road endpoint on the image). The conflated road network is constructed by connecting these transformed endpoints (see Figure 2.12(b)).



a) The endpoint A on the original road segments (white lines) will be transformed to the point B on the imagery, after applying rubber-sheeting

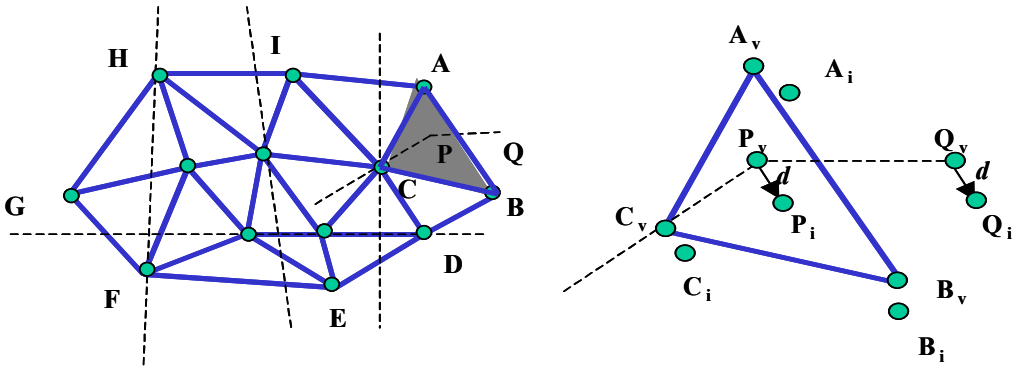
b) Partial conflated roads after applying rubber-sheeting to Figure 2.11(a)

Figure 2.12: An example of rubber-sheeting

Piecewise linear rubber sheeting based on triangles with extremely small angles (i.e., long and thin triangles) results in distorted conflation lines. Since the Delanuay triangulation avoids triangles with extremely small angles, it alleviates the problem. The detailed piecewise linear rubber-sheeting algorithms can be found in [35, 45].

2.3.3 Region Expansion

I developed a technique named "Region Expansion" to reduce the spatial inconsistencies for the regions where there are no feature points (e.g., intersection point) on the vector data and imagery to perform conflation. From the boundary of the conflation area (the convex hull formed by control points, as the polygon *ABDEFGHI* in Figure 2.13(a)), the system estimates new control points based on existing control points. Using these new control points, the original conflation area where the control points are found can be expanded. The key computation of region expansion is estimating the new control points from the Delaunay triangles closest to



a) Convex hull formed by existing control points (dash lines: partial roads, points: control points on vector data; triangles: Delaunay triangles)

b) New control point pair Q_v and Q_i (focus on a boundary triangle ΔABC)

Figure 2.13: Generate new control point pairs based on existing control points

the boundary of the conflation area. Intuitively, the estimation is based on the assumption that the displacements between intersections in the vector and the corresponding points in the imagery are similar in a small area.

Consider the example shown in Figure 2.13. The Delaunay triangle vertices are the control points on the vector data, and the dash lines represent a partial road network. The gray Delaunay triangle ΔABC is a boundary triangle (i.e., a triangle with at least one edge that is not shared with any adjacent triangles). Figure 2.13(b) focus on the control point distributions of the three vertices of triangle ΔABC, where A_v, B_v and C_v are the control points on vector data and A_i, B_i and C_i are the corresponding control points on the underlying image, respectively. P_v (within ΔABC) is an endpoint of road segment $\overline{P_vQ_v}$ on the road network (i.e., dashed lines), and P_i is the corresponding point on imagery after applying the rubber-sheeting transformation. To generate a new control point out of the control point convex hull, the system has to find a point on the road network and its corresponding point on the

imagery. Q_v (another endpoint of the segment $\overline{P_vQ_v}$) is a good candidate as a control point on the vector data. Furthermore, since in a small area the displacements between intersections in the vector and the corresponding points in the imagery are similar, the corresponding point of Q_v on the imagery (i.e., Q_i) can be estimated by adding the same displacement (noted as d in Figure 2.13(b)) as that between P_v and P_i. Hence, the system obtains a new control point pair (i.e., Q_v and Q_i). Then, the system can further apply Delaunay triangulation and rubber-sheeting based on the original and these new-added control point pairs. Region expansion works effectively for the areas where the positional discrepancies between vector data and imagery are similar. However, if the existing control points are not accurate, the new control points will not be accurate either.

2.4 Performance Evaluation

In this section, I evaluated my approaches by conducting several experiments on various real world data. The tested datasets are described in detail in Section 2.4.1. The purpose of the experiments is to evaluate the utility of my algorithms in integrating real world data. I am interested in measuring the improvement in the accuracy of the integration of road vector and imagery using my techniques. To that end, I performed experiments to validate the hypothesis: using AMS-conflation, we can automatically improve the alignment of different accuracy level road vectors with orthoimagery. Section 2.4.1 describes the experimental setup and the datasets used to evaluate my approach. Section 2.4.2 presents my evaluation methodology to

measure the performance improvement. Section 2.4.3 discusses the experimental results.

2.4.1 Experimental Setup

I used the following two datasets for the experiments:

(1) Orthoimagery

The imagery used in the experiments is the geo-referenced USGS high resolution color orthoimagery (with 0.3 m/pixel resolution) and geo-referenced USGS gray-scale DOQ imagery with 1 m/pixel resolution. In particular, I tested the color imagery that covers a partial area of the county of St. Louis, MO, and the gray-scale imagery that covers a partial area of the city of El Segundo, CA. This imagery is available online and can be queried from the Microsoft TerraService web service [4].

(2) Vector data (road networks)

Three road networks from different data providers were used as the vector data:

- TIGER/Lines from U.S. Census: The TIGER system was developed by the U.S. Bureau of Census. The Census Bureau has developed the TIGER/Line files, which are extractions of selected geographic and cartographic information from the TIGER database. In particular, I focus on the road networks queried from TIGER/Line files (called TIGER/Lines henceforth).
- NAVSTREETS from NAVTEQ: NAVSTREETS is a commercial product consisting of high quality vector data with highly accurate geometry. It is regularly updated by NAVTEQ using base maps acquired from a variety of

sources including local governments, commercial mapping agencies, and other public agencies. It is primarily used for car navigation systems and road route planning. Many online street map services, such as MapQuest, utilize NAVSTREETS as the base dataset for route planning.

- Road network data (called MO-DOT henceforth) from the Missouri Department of Transportation [42]: MO-DOT is also a high quality dataset.

In general, all the road network data listed above have rich attribution; however, TIGER/Lines has both poor positional accuracy and road geometry. With TIGER/Lines and MO-DOT, the number of lanes can be inferred from the attribute "*CFCC (Census Feature Class Codes)*" associated with each road segment, while the number of lanes can be obtained from the attribute "*LANE_CAT*" in NAVSTREETS. Furthermore, the locations of road intersections and the road directions around each intersection are calculated by analyzing these road networks using the algorithms described in Section 2.1.1. In general, NAVSTREETS and MO-DOT are high quality road vectors, but with various accuracy levels. Figure 2.14

a) Original TIGER/Lines with imagery b) Original NAVSTREETS with imagery c) Original MO-DOT with imagery

Figure 2.14: Original road vector (white lines) superimposed with imagery

shows some sample images to indicate that there are various spatial inconsistencies between the USGS high resolution color imagery and the three different vectors.

My automatic conflation system was developed in C#. The algorithm allows the user to specify the following parameters: the two datasets to conflate (e.g., "imagery and TIGER/Lines", "imagery and NAVSTREETS" or "imagery and MO-DOT") and the type of filtering techniques. The output of my conflation system was a set of conflated roads for the three different types of vector datasets. The experiments were conducted on a Pentium III 1.2GHz processor with 512MB memory and Windows XP (with .NET framework installed). In order to evaluate my approach on various real world data, I applied AMS-conflation to the diverse areas as summarized in Table 2.1. In addition, I manually plotted the real road axes (called reference roads) as the ground truth with which I compare the conflated roads.

2.4.2 Evaluation Methodology

In order to evaluate the performance of my conflation algorithm, I manually drew reference roads to compare with conflated roads, and I also developed an evaluation schema to measure: (1) *The percentage of the reference roads in imagery for which the system generated conflated roads*, (2) *The percentage of correctly conflated roads with respect to the total conflated roads*, (3) *The percentage of the total length of the conflated roads that is within a specific distance of the reference roads*. Toward that end, I compared the conflated road network with the reference road network.

Table 2.1: Tested datasets used in the experiments

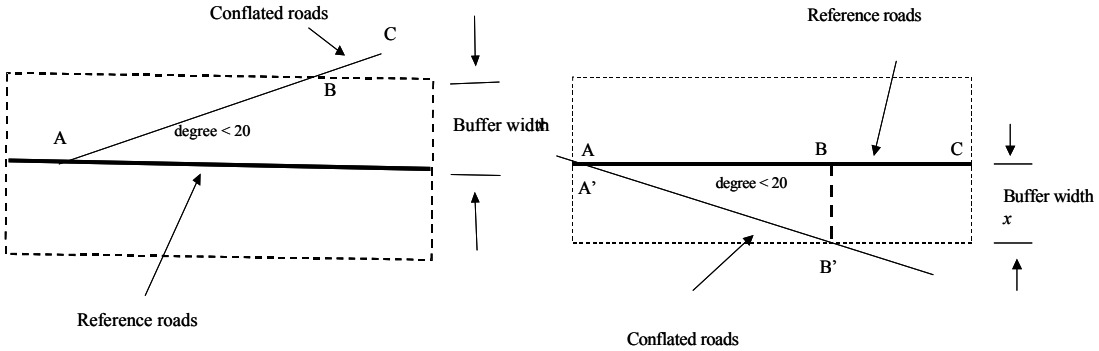
	Test area 1	Test area 2	Test area 3	Test area 4
Area covered*	Latitude: 38.5808 to 38.5947 Longitude: -90.4049 to -90.388 Width: 1.5 km Height: 1.5 km	Latitude: 38.5703 to 38.5842 Longitude: -90.543 to -90.526 Width: 1.5 km Height: 1.5km	Latitude: 38.5962 to 38.6101 Longitude: -90.490 to -90.473 Width: 1.5 km Height: 1.5km	Latitude: 33.914 to 33.932 Longitude: -118.4209 to -118.399 Width: 2 km Height: 2 km
Total road length of TIGER/Lines (m)	23534.52	21443.96	7966.62	46580.64
Total road length of NAVSTREETS (m)	24360.00	21921.29	9876.02	N/A**
Total road length of MO-DOT (m)	24759.30	21796.92	9431.68	N/A**
Total road length of reference roads(m)	25471.63	21999.00	9252.01***	46660.20
Area features	0.3 m/pixel color imagery. Suburban area (covering some urban area) with high road density (11.3 km/km^2) and high house density	0.3 m/pixel color imagery. Suburban area with high road density (9.7 km/km^2) and high house density. Perceptually, the majority of road color in this area is different from the road color in test area 1.	0.3 m/pixel color imagery. Rural area with medium road density (4 km/km^2). 12% of the roads are highways.	1 m/pixel gray-scale imagery. Urban area with high road density (12.83 km/km^2) and high house density.

* Test area 1, 2, and 3 cover partial areas of the county of St. Louis, MO. Test area 4 covers a partial area of the city of El Segundo, CA.
** These road vector datasets were inaccessible at the time the experiments were performed.
*** These reference roads are shorter than vector data because some straight reference roads are depicted as curves in vector datasets.

In the computer vision literature on automatically extracting roads from imagery, there are many methodologies proposed to evaluate the extracted roads against real roads [22, 46]. Due to the natural similarity between the problem of evaluating extracted roads and my problem of evaluating conflated roads, I can utilize these existing algorithms to evaluate the conflation results. In particular, I adapted the "road-buffer method" proposed in [46]. The road-buffer method is utilized to

measure the accuracy of automatically extracted roads with respect to real roads. I revised this method to measure the accuracy of conflated roads with respect to real roads.

According to the algorithm proposed in [46], to compare two road networks (in my case, they are the reference road network and the conflated road network), the first step is to split both networks into short pieces of equal length. Then, a constant predefined buffer width is constructed around the reference road network (as the examples shown in Figure 2.15). Every portion of the conflated road network within the given distance (i.e., the buffer width) from the reference road network is considered to be matched. Furthermore, the direction difference between the matched and reference road axes must be less than a pre-defined threshold *d* (*d* was set to 20 degree in [46]). The drawback of this procedure is that the performance is highly affected by the predefined constant buffer width. Instead of using the constant buffer width for each road segment, I used the real road widths in the imagery as the buffer. Hence, the roads with different widths have different buffer widths. The



a) Segment AB: matched conflated road b) Segment AB: matched reference road

Figure 2.15: Buffer method for evaluating completeness and correctness

pieces of the conflated roads within the buffer to the reference roads with consistent direction are considered as matched.

Figure 2.15(a) shows the example of matched conflated roads with respect to reference roads. Segment AB is calculated as *matched conflated road*, while BC is not. Figure 2.15(b) shows the example of matched reference roads with respect to conflated roads. Segments AB is categorized as *matched reference road*, since the conflated road segment A'B' can be used to "complete" the reference road segment AB. Segment BC is unmatched reference road.

Using this term, two measurements, completeness and correctness, are defined as follows [46].

$$\text{Completeness} = \frac{\text{Length of matched reference roads}}{\text{Total length of reference roads}} \qquad \text{Eq.(2.4)}$$

$$\text{Correctness} = \frac{\text{Length of matched conflated roads}}{\text{Total length of conflated roads}} \qquad \text{Eq.(2.5)}$$

Basically, the completeness is *the percentage of the reference roads in imagery for which the system generated conflated roads*. On the other hand, correctness is *the percentage of correctly conflated roads with respect to the total conflated roads*. However, the other measurement, RMS (root-mean-square error), described in [46] does not meet the requirements to compute *how far the conflated road network is from the reference road network*, since it only measures *how far the matched conflated road network is from the reference road network*. Instead of computing a number (e.g., average distance) to illustrate how far from each other the two networks are, I would like to measure the percentage of the total length of the

(conflated) roads that is within a specified distance *x* relative to the reference roads (e.g., 95% of the conflated roads are within five meters of the reference roads or 50% of the conflated roads are within 3-6 meters of the reference roads). The method proposed in [20] is a technique to assess positional accuracy. In the example shown in Figure 2.16, I consider a buffer with width *x* around the reference road network, then compute the proportion *p* of the conflated roads that lies within the buffer [20]. Using this technique, only the distance difference between two roads is considered. The errors due to the difference of directions between roads are captured by completeness and correctness.

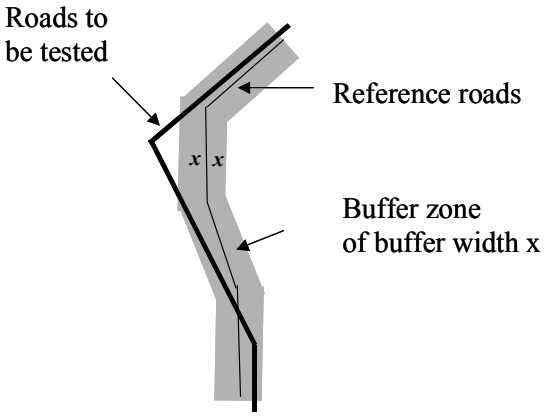


Figure 2.16: Positional accuracy evaluation

I conducted the experiments as follows for test area 1, 2 and 3:

- Step 1: Learn the histogram (as shown in Figure 2.4) from nearly 500 manually labeled rectangles[21] from some color orthoimages covering partial areas of the County of St. Louis, MO.
- Step 2: Apply AMS-conflation algorithm to conflate each area (image) with TIGER/Lines, NAVSTREETS and MO-DOT respectively.

[21] There are 50,000 pixels covered by these rectangles and it took about 2.5 hours to perform the labeling process.

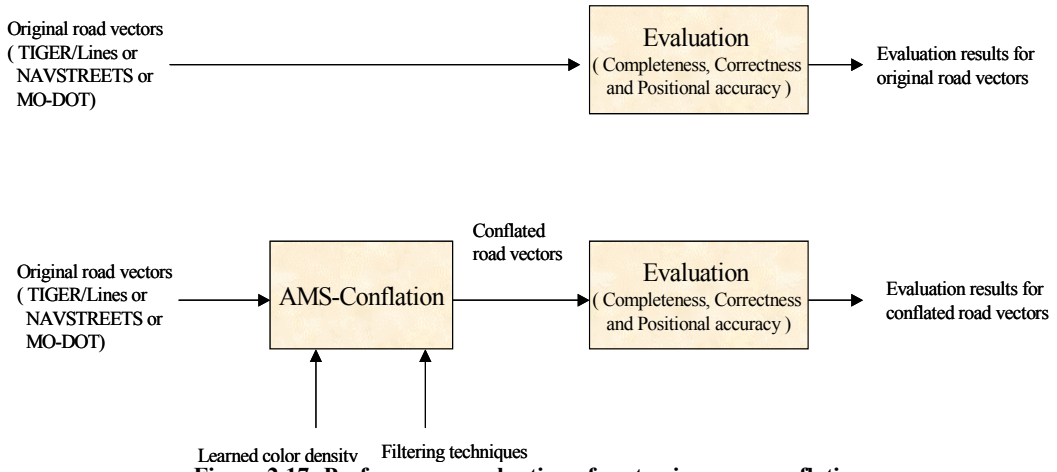


Figure 2.17: Performance evaluation of vector-imagery conflation

As shown in Figure 2.17, for test area 1, 2 and 3, I conducted experiments to measure the accuracy of the original road vectors and conflated road vectors generated by AMS-conflation. Then, I compare the evaluation results for conflated road vectors with the results for original road vectors. In addition, I also measured the quality of the detected control points (before and after applying filtering techniques) by the precision and recall metrics defined in Section 2.1.2.2 (Eq.(2.2) and Eq.(2.3)). For the fourth test area (covering part of the city of El Segundo, CA), basically, I repeated the same process as above, but the system learned road color information from black-white images that cover some areas of El Segundo and I only conflated TIGER/Lines. I discuss the detailed experimental results in the following section.

2.4.3 Experimental Results and Interpretations

I obtained similar conflation performance for the two filtering techniques, VMF and DB(p,D)-outliers detector (under the setting that 50% was assigned to p and D was

15 m). Therefore, in this section, I will take the VMF as the filtering technique to illustrate the conflation results.

2.4.3.1 Results of precision and recall of identified intersections

Finding accurate control point pairs is a very important step in the conflation process as all the other points in both datasets are aligned based on the control point pairs. Hence, I evaluated the LTM performance by calculating the precision and recall (the metrics are defined in Eq.(2.2) and Eq.(2.3) of Section 2.1.2.2) of the detected control points (before and after applying filters). The results are listed in Table 2.2. I also included the precision/recall of original road network in Table 2.2 to

Table 2.2: Results of identified intersections

		Test area 1	Test area 2	Test area 3	Test area 4
Original road network	Precision	7.1%	8.7%	4.8%	3.8%
	Recall	6.7%	7.7%	4.5%	3.7%
Without filtering	Precision	72.3%	82.1%	57.9%	78.9%
	Recall	68.1%	24.2%	52.4%	52.1%
Using VMF filtering	Precision	87.1%	83.1%	69.2%	94.8%
	Recall	45.4%	21.2%	42.9%	34.0%

a). Precision/recall of identified intersections for TIGER/Lines

		Test area 1	Test area 2	Test area 3
Original road network	Precision	15.6%	23.3%	19.1%
	Recall	15.2%	23.1%	18.2%
Without filtering	Precision	87.7%	82.1%	64.7%
	Recall	76.2%	40.7%	52.4%
Using VMF filtering	Precision	97.1%	92.6%	83.3%
	Recall	54.1%	27.5%	47.6%

b). Precision/recall of identified intersections for NAVSTREETS

		Test area 1	Test area 2	Test area 3
Original road network	Precision	57.1%	32.2%	28.6%
	Recall	55.2%	31.9%	27.2%
Without filtering	Precision	83.8%	82.1%	83.3%
	Recall	73.9%	50.5%	71.4%
Using VMF filtering	Precision	98.1%	97.1%	90%
	Recall	42.9%	37.3%	43%

c). Precision/recall of identified intersections for MO-DOT

demonstrate that LTM technique improved both precision/recall of original vector data.

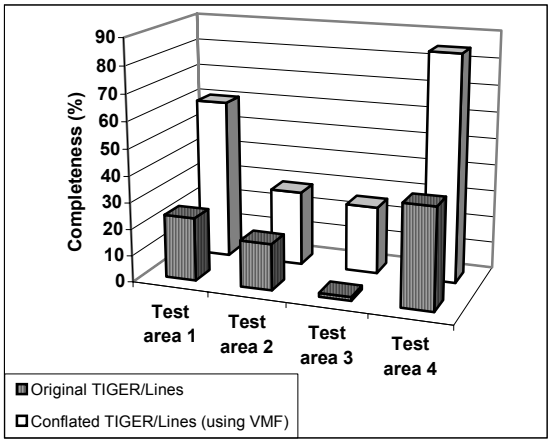
As shown in Table 2.2, LTM performed differently for various real world scenarios. This is because these vectors have different qualities and the orthoimagery has various levels of complexity. Hence, I obtained high precision (up to 98%) control points in some areas (such as test area 1), while medium precision (about 70%) in other areas (such as the alignment of TIGER/Lines and imagery in test area 3). In general, I improve the precision after applying filtering techniques, although this step reduces the recall. However, for the conflation process higher precision is more important than higher recall, since my approach is not trying to correct individual errors, but rather to determine the local transformations that allow the system to integrate vector datasets and imagery.

2.4.3.2 Results of completeness and correctness of conflated roads

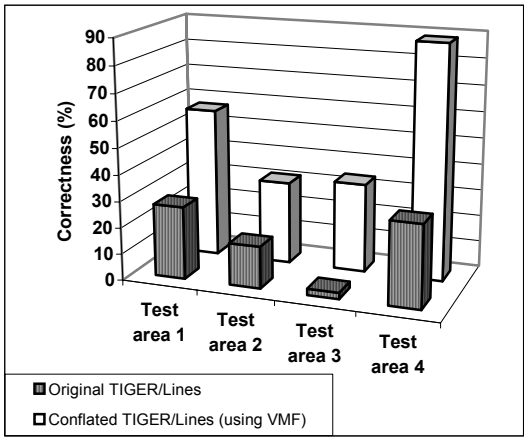
The experimental results of completeness and correctness for each vector dataset are listed in Figure 2.18(a)-(f). Intuitively, the completeness corresponds to the users' demands: *how much is missing in the original road network or in the conflated road network with respect to the reference road network*. The correctness, on the other hand, is related to *the percentage of the original or conflated road segments that could be considered as reference road segments*. I showed the completeness (and correctness) for each utilized vector dataset respectively. In addition, the completeness (and correctness) values are grouped by tested areas as the X-axis of

Figure 2.18. The Y-axis in Figure 2.18(a)(c)(e) shows the completeness for original road vector and conflated road vectors using VMF-filtered intersections as control points. Furthermore, in Figure 2.18(b)(d)(f), the Y-axis depicts the correctness. For example, as shown in Figure 2.18(a) and (b), when utilizing VMF-filtered intersection points to generate conflated TIGER/Lines for test area 1 (on color images of the county of St. Louis, MO), my approach improved the completeness from 19% to 59%, and correctness from 25% to 55%. Another example, as the results for test area 4 shown in Figure 2.18(a)(b), my approach improved the completeness and correctness 2.5 times over the original TIGER/Lines (on black-white images of the city of El Segundo, CA). In addition, there are some immediate observations from this figure:

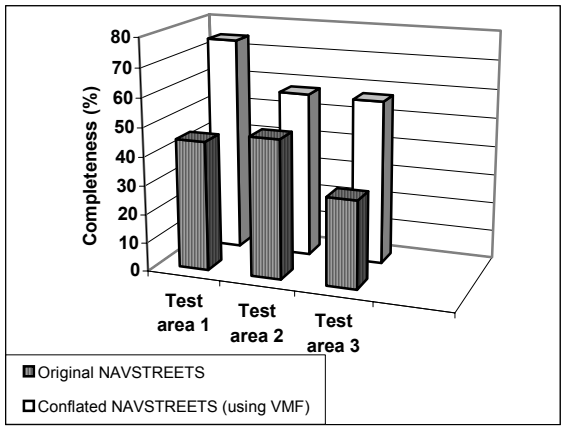
- The data quality of MO-DOT is superior to NAVSTREETS and much better than TIGER/Lines, which is consistent with our prior knowledge about these three different datasets. Moreover, from the diverse completeness and correctness in each vector dataset for different test areas, I concluded that each vector dataset itself has various accuracies. This is also consistent with the vector data quality statements quoted by the data providers.



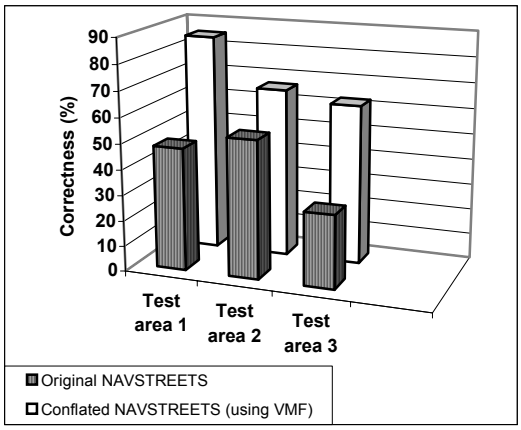
a) Completeness assessment for TIGER/Lines



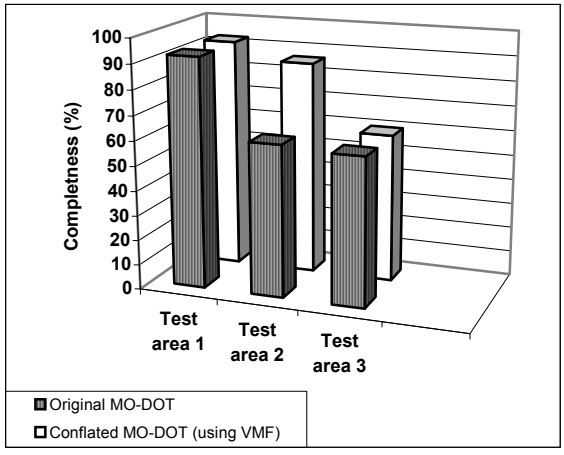
b) Correctness assessment for TIGER/Lines



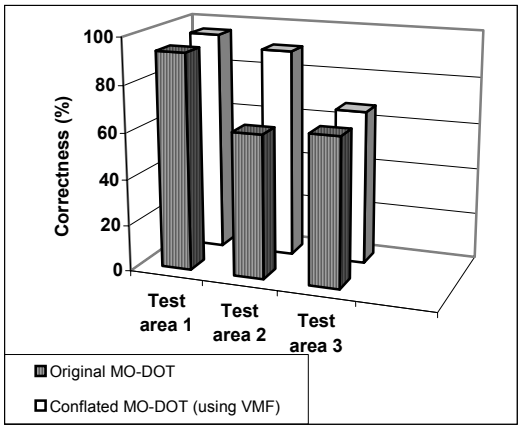
c) Completeness assessment for NAVSTREETS



d) Correctness assessment for NAVSTREETS



e) Completeness assessment for MO-DOT



f) Correctness assessment for MO-DOT

Figure 2.18: Evaluation results for completeness and correctness

- Consider TIGER/Lines as the vector data source. The shapes (and geometry) of the original TIGER/Lines are often inconsistent with the corresponding roads in the imagery. Hence, as shown in Table 2.2 and Figure 2.18, I obtained low completeness/correctness for original TIGER/Lines and low precision/recall for the intersections of original TIGER/Lines. For a particular road segment, if the shape of the original vector data is inconsistent with roads in the imagery (as in the example of TIGER/Lines), my approach may not align them well, although the majority of intersections might be aligned. This is mainly because my matching is at the point level, not at the edge level. As illustrated by the TIGER/Lines in test area 1 (see Table 2.2(a)), my approach improved the node (intersection) alignment (as the precision improved from original 7% to 87.1%), while it achieved completeness from 19.1% to 55% and correctness from 20.6% to 55%. However, recently, not only is the imagery quality enhanced, the quality of vector data is also significantly improved. Consider the conflation of high quality imagery and high quality vector dataset, such as NAVSTREETS. The road shapes of NAVSTREETS are very similar to the road shapes in the imagery. Hence, the major issue is that there are some local inconsistencies between them. AMS-conflation can capture these local transformations (based on intersection to intersection matching information) and maintain the road shapes.
- On average, good improvements were achieved for TIGER/Lines (as shown in Figure 2.18(a)(b)). For NAVSTREETS, my approach performed 1.3 to 1.9

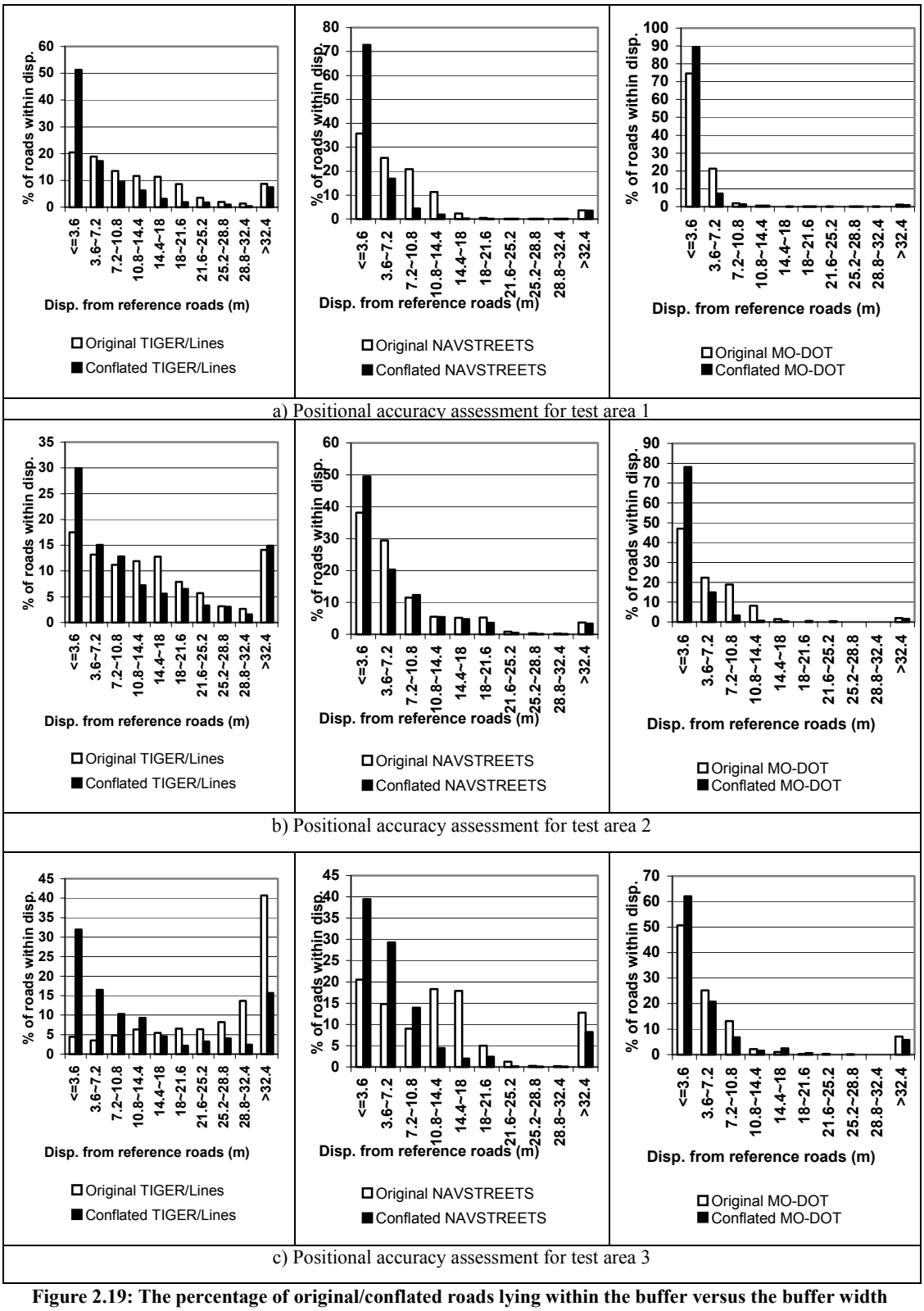
times better than the original data (as in Figure 2.18(c)(d)), while the approach only gained marginal improvements for MO-DOT data on test area 1 and 3 (see Figure 2.18(e)(f)). This is due to high completeness (92.54%) and correctness (93.38%) of the original MO-DOT data in test area 1. Some roads are not aligned well around highways in test area 3. The road widths of highways vary and are difficult to predict. The problem could be addressed by integrating other information sources with those details. In fact, after visual examination, I found many misaligned road segments are close to the margins of road buffers (i.e., roadsides). When relaxing the "buffer-widths", I can obtain higher completeness and correctness. This kind of assessment is illustrated by the "positional accuracy" evaluation described next.

2.4.3.3 Results of positional accuracy of conflated roads

The experimental results of "positional accuracy" categorized by road vectors for each test area are illustrated in Figure 2.19(a)-(c). Intuitively, the "positional accuracy" corresponds to the users' demand: *how far is the conflated road network from the centerlines of the real (reference) road network*. I evaluated these displacements between two networks by gradually increasing the buffer-widths constructed around the reference road network. The buffer-width was increased by 3.6 m (i.e., the U.S. standard lane width). As shown in the X-axis of Figure 2.19, the displacement values are grouped every 3.6 m. The Y-axis shows the percentage of conflated roads lying within the displacement range represented by the X-axis. For

example, as shown in Figure 2.19(a), when utilizing VMF-filtered intersection points to generate conflated NAVSTREETS for the first test area, about 75% of the conflated roads are within 3.6 m from the reference roads, and only 35% of the original NAVSTREETS are within 3.6 m displacement. Although my approach did not achieve significant improvements on completeness/correctness for MO-DOT data (as stated earlier), it achieved better positional accuracy: On average, 91% of the conflated MO-DOT roads are within 7.2 m of the reference roads compared to 80.3% of the original MO-DOT.

Even higher improvements were achieved for TIGER/Lines and NAVSTREETS. On average, 76% of conflated NAVSTREETS are within 7.2 m displacement versus 54.6% of original NAVSTREETS and 53.9% of conflated TIGER/Lines are within 7.2 meters versus 25.9% for the original TIGER/Lines. In particular, compared to NAVSTREETS and MO-DOT data, TIGER/Lines have poor positional accuracy and poor geometry, because large portions of curve-shaped roads were simplified as straight lines. For such severely distorted original TIGER/Line segments, our approach is limited in aligning imagery curves and vector lines, although the detected intersections are matched (as shown in Table 2.2). Hence, only about 47% of conflated TIGER/Lines in test area 2 and 3 are within 7.2 m of the reference roads, while 70% to 85% of conflated NAVSTREETS and MO-DOT are within 7.2 m. However, compared to the original TIGER/Lines, my approach significantly improved the positional accuracy.



a) Positional accuracy assessment for test area 1

b) Positional accuracy assessment for test area 2

c) Positional accuracy assessment for test area 3

Figure 2.19: The percentage of original/conflated roads lying within the buffer versus the buffer width

In the first test area, there are about 25% of the streets in grid shape and all three road vectors provide accurate road shapes over these street grids. Therefore, my approach obtained better performance in test area 1. The issue of quality of the vector data can be addressed by starting with higher quality data such as NAVSTREETS or MO-DOT data. However, there are small portions of the conflated roads not aligning well to the imagery. This is mainly because the color of these misaligned roads are very different from what the system learned or the roads are close to the conflation area margins where long and thin Delaunay triangles were constructed. These issues could be alleviated by doing more training to recognize a wider range of road colors and applying the conflation on larger areas.

In the fourth test area, the approach achieved high improvement for TIGER/Lines. 85% of conflated TIGER/Lines are within 8 m displacement versus 43% of original TIGER/Lines (see Figure 2.20). This demonstrates the utility of my approach to conflate vector data with black-white and lower resolution imagery.

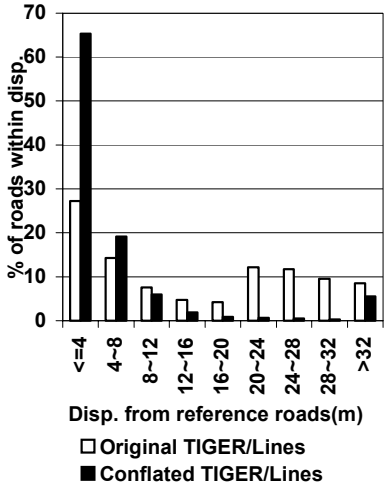


Figure 2.20: Positional accuracy assessment for test area 4

2.4.3.4 Results of using filtered control points vs. using unfiltered control points

Finally, I also compared the performance of running the conflation algorithm with filtered and unfiltered

control points. I conducted experiments to measure positional accuracy for the original MO-DOT and conflated MO-DOT in test area 2.

The results shown in Figure 2.21 indicate that the conflated roads generated by filtered control points outperformed those generated by unfiltered control points, especially for displacements greater than 7.2 m. This is because the conflation process does not require a large number of control point pairs to perform an accurate alignment. In fact, a set of control points with higher accuracy would serve better for the conflation process, which is what my filtering techniques do. Therefore, I only consider the conflation performance (as illustrated in Figure 2.18 and Figure 2.19) by utilizing the filtered control points.

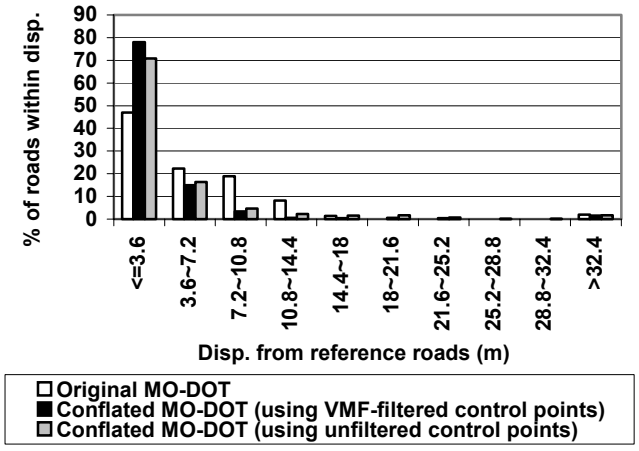


Figure 2.21: Comparison of filtered vs. unfiltered conflation results for test area 2

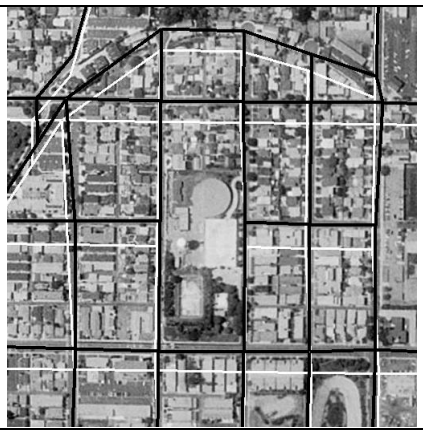
2.4.3.5 Execution time and summary

Since the running time of AMS-conflation was mainly dominated by the LTM routine to detect road intersections, I used the running time of LTM as the overall execution time (the query time for retrieving online images or vector datasets was not included). On average, the execution time for locating an intersection in a localized area was about three to five seconds (it depends on the radius setting for

a) Conflated TIGER/Lines with color imagery

b) Conflated NAVSTREETS with color imagery

c) Conflated MO-DOT with color imagery



d) Conflated TIGER/Lines with black-white imagery

Figure 2.22: Vector-Imagery conflation results (white lines: original road network; black lines: after applying conflation)

LTM). For example, the total running time for generating conflated roads for a small area of 900 m by 900 m with 20 intersections is about one minute.

In sum, these experimental results indicate that my approach automatically and efficiently improves the alignments of various vector datasets with orthoimagery. This validates my experimental hypothesis. Figure 2.22 shows some sample images, after applying conflation to each of the three road vectors, respectively.

Chapter 3

Automatic Conflation of Street Maps and Orthoimagery

In addition to road vector datasets and orthoimagery, raster maps are another type of geospatial dataset that is readily available from online map services, such as Yahoo Map Service, MapQuest, and Microsoft TerraService. Some services, such as TerraService, can display the imagery and related maps in separate frames. However, there is no service with the capability to automatically align imagery and maps in a single framework. Furthermore, to the best of my knowledge, there is no research that addresses the problem of automatic conflation of maps and orthoimagery.

In this section, I present an approach to automatically conflate streets maps with high resolution georeferenced orthoimagery [9]. I also present the experimental results of conflating various real world street maps and orthoimagery. Figure 3.1 shows the overall approach for conflating imagery and maps[22]. First, my approach automatically conflates the road vector data with the orthoimagery to find the intersections in the image. Next, the approach finds the road intersection points on the street map. Then, it computes the alignment between the two point sets to obtain a commom point pattern as the set of control point pairs. Finally, it deforms one of

[22] A pseudo code in Appendix B presents the map-imagery conflation algorithm.

the datasets (e.g., maps) to align the other (e.g., imagery) utilizing these identified control point pairs.

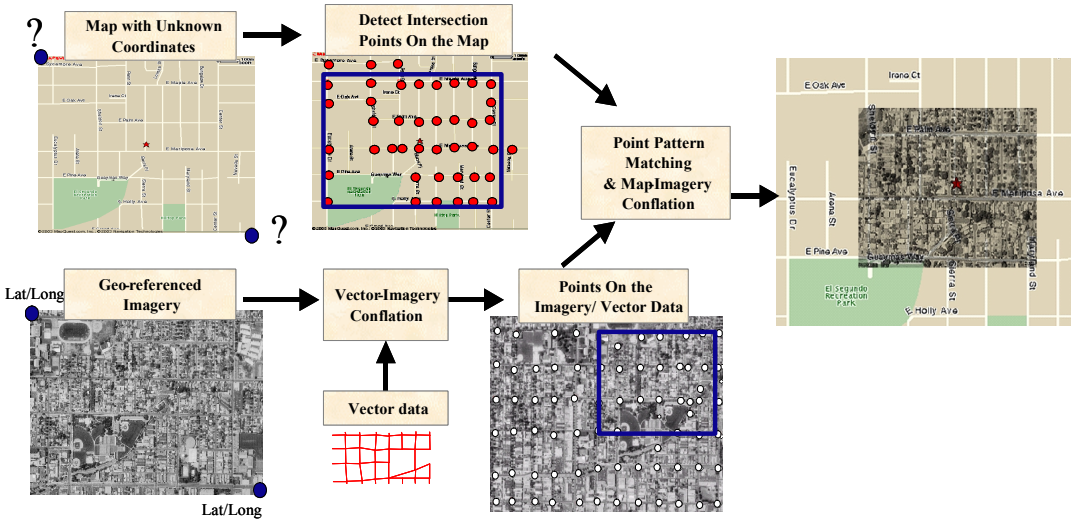


Figure 3.1: Overall approach to align orthoimagery and street maps

3.1 Identifying Feature Points (Road Intersections)

Similar to vector-imagery conflation, the essential task of map and imagery conflation is to find accurate control point pairs from both datasets. In addition to the similar challenges as vector-imagery conflation, aligning maps with imagery is a more difficult problem. Mainly, this is because neither map nor imagery has reference points (such as road intersections or landmarks) established in advance for a matching problem. A straight forward method is analyzing the map and imagery to identify common and corresponding spatial objects (such as road segments or road intersections) from both datasets as control points. However, this process requires significant CPU time to process an image in its entirety and still may result in inaccurate results. Moreover, it is unclear which kinds of features are good

candidates to be identified from both datasets, especially when the system does not know the context (and the geo-coordinates) of the maps or imagery in advance.

Since I focus on street maps and imagery conflation, I can utilize the road intersections detected from both datasets as control points [11, 17, 18, 21]. However, as stated in Section 2.1.2, automatic extraction of road intersection points from imagery as feature points is a difficult task due to the complexity that characterizes natural scenes. Furthermore, it is also difficult to detect road intersections from maps and discover the matched intersections from both datasets. I propose a solution to address these issues in this section. Basically, the proposed solution for map-imagery conflation conforms to that of vector-imagery conflation. That is, I exploit information from each of the sources to be integrated to assist the automatic feature detection and alignment process.

3.1.1 Identifying Intersections on Imagery

Although automatic extraction of road intersection points from imagery is a challenging task [1, 22, 23], I can take full advantage of my vector-imagery conflation algorithm described in Section 2 to overcome the issue. The idea is to utilize auxiliary information sources (i.e., road vector data) that are not part of a map or an image, but have information relevant to both sources. In other words, I utilize the road vector data as "glue" to align maps and imagery. First, I align road vector data with imagery using vector-imagery conflation. As a result the conflated intersection points on the road network are aligned with the intersection points on the

imagery. I can then use the conflated intersection points as intersection points on the imagery. Figure 3.2 shows an example illustrating the detected intersection points on an image before and after conflating the image with a road network.

a) Imagery with road network, before conflation

b) Detected intersection points on imagery, after conflation

Figure 3.2: Intersection points automatically detected on imagery

3.1.2 Identifying Intersections on Street Maps

Since there are few online street maps with known geo-coordinates, I cannot apply the same localized image processing (i.e., LTM), described in Section 2.1.2, to find intersection points on maps. This is because I cannot find the corresponding vector data for the map, since the map geo-coordinates are unknown. Hence, for those maps whose geo-coordinates are unknown in advance, I utilize automatic map processing and pattern recognition algorithms described below to identify the intersection points on maps.

Ideally, intersection points on street maps could be extracted by simply detecting road lines. However, due to the varying thickness of lines on diverse maps, accurate extraction of intersection points from maps is difficult [13, 31, 38]. In addition, there

is often noisy information, such as symbols, contour lines and alphanumeric characters on the map, which make it even harder to accurately identify the intersection points.

To overcome these problems, I adapted the automatic map processing algorithm described in [31] to skeletonize the maps for extracting intersection points. The basic idea is to detect intersection points only on the map that has been pre-processed by line thinning algorithms and noise-removal procedures. In particular, the process can be divided into the following subtasks: (1) isolate map data by a threshold, (2) decrease line width by thinning algorithms, such as [2], (3) recognize intersection points by crossing number (CN), the number of lines emanating from an intersection point [2], (4) remove misidentified intersections caused by noisy information (such as symbols and text). The details of the line intersection detection algorithm are discussed in [31]. However, this algorithm assumes that the roads are illustrated as multiple single-lined segments on the maps. Therefore, it is not appropriate for the maps where roads are depicted as double lines. In particular, from my experiments with diverse single line online street maps, this algorithm achieved 65% to 95% precision in identifying road intersections, while it worked poorly (with 20% to 30% precision) for double line street maps.

To overcome this problem, instead of using "crossing number (CN)" (for an intersection, its CN must be greater than two) to detect intersections, I utilize feature-detection functions implemented in OpenCV[23] to detect promising points, such as

[23] http://sourceforge.net/projects/opencvlibrary

corners and distinct points. Then, a verification process is conducted to check whether there is any linear structure around each detected corner point. If so, the detected point will be characterized as an intersection point. I found that my revised approach can achieve 76% precision (on average) on my tested street maps (including both single line and double line street maps).

My above-mentioned map intersection detection algorithm [9] is further improved by the approach proposed in [13] to deal with more complicated maps, such as USGS topographic maps,[24] which illustrate not only roads, but also other features, such as contour lines, railroads, etc. Chiang et al. [13] enhanced my algorithm in several ways: (1) separating the linear structures from the maps by dynamically investigating thresholds and using the text/graphics separation techniques proposed in [6], (2) using morphological operators (e.g., erosion and dilation operators) to reconnect and clarify the potential road segments. On average, this method [13] can achieve 92% precision and 67% recall to detect map intersections.[25] Moreover, it has the capability to compute the number of road segments that are incident to an intersection (called the degree of an intersection [35]) and the directions of those incident segments. This additional information can help to improve my point pattern matching algorithm (described next).

Figure 3.3 shows an example illustrating the detected intersection points on a topographic map queried from Microsoft TerraService. Although the algorithm

[24] http://topomaps.usgs.gov/drg/

[25] A map intersection is characterized as an accurately detected point if and only if its location is less than five pixels from the exact position of the actual map intersection.

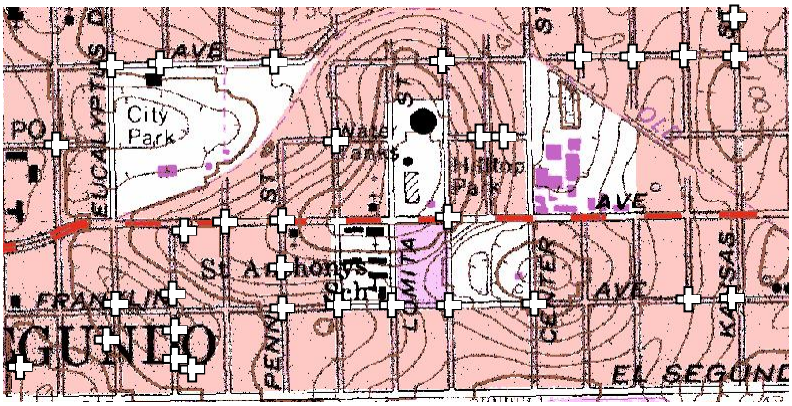


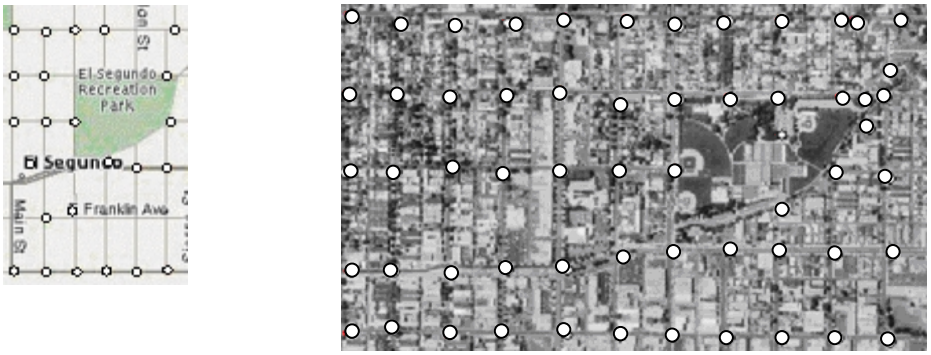
Figure 3.3: Intersection points detected on a map (each detected intersection is marked as a cross)

described above can significantly reduce the rate of misidentified intersection points on the maps, it is still possible that both noisy points are detected as intersection points and some intersections may be missed. For example, the points near the lower left corner (e.g., the detected points in "NDO") were mistaken for some road intersections. However, my point pattern matching algorithm can tolerate the existence of misidentified intersection points.

3.2 Generating Control Points by Point Pattern Matching

So far I have identified a set of intersections on both the street map and the imagery. Figure 3.4 shows an example of the two point sets on a map and an image, respectively. The remaining problem is to match these points in order to generate a set of control point pairs.

Let $M = \{m_i \mid m_i = (x_i, y_i)$, where (x_i, y_i) is the location of detected intersections on the map$\}$ and $S = \{s_i \mid s_i = (lon_i, lat_i)$, where (lon_i, lat_i) is the location of identified intersections on the imagery$\}$.



a) A map with some detected intersections b) An image with detected intersections

Figure 3.4: Intersection points detected on a map and an image

Our objective is to locate the set: Rel_{Pat}={(m_i,s_i) | where m_i is the accurately detected point on the map and s_i is the corresponding point on the imagery. That is, point m_i and s_i, are formed by the same incident roads.}.

This set Rel_{Pat} would be utilized as control point pairs to align maps and imagery. If the system can recognize the names of road segments that are incident to intersections, it can use these road names to infer the set Rel_{Pat}. However, it is difficult to recognize the road names automatically from the raster maps [6]. In addition, road vector data may not be associated with the attribute, road name. Instead, my approach relies on some prominent geometric information, such as the distribution of points, the degree of each point and the direction of incident road segments, to locate the matching point pattern. In other words, the problem of point pattern matching is at its core a geometric point sets matching problem.

The basic idea is to find the transformation *T* between the layout (with relative distances) of the intersection point set *M* on the map and the intersection point set *S* on the imagery. The key computation of matching the two sets of points is

calculating a proper transformation T, which is a 2D rigid motion (rotation and translation) with scaling. Because the majority of map and imagery are oriented such that north is up, I only compute the translation transformation with scaling. Without loss of generality, I consider how to compute the transformation where I transform from a fraction α of the points on maps to the points on imagery. The reason why only a fraction α of the points on the maps is considered is that there are misidentified points arising from the processes of image recognition (i.e., identifying intersection points on maps). Moreover, there may be some missing intersection points or noisy points on the imagery as well.

The transformation T brings at least a fraction α of the points of M into a subset of S on the imagery. This implies:

$\exists\ T$ and $M' \subseteq M$, such that $T(M') \subseteq S$, where $|M'| \geq \alpha |M|$ and $T(M')$ denotes the set of points that results from applying T to the points of M'. Or equivalently, for a 2D point (x, y), in the point set $M' \subseteq M$, $\exists\ T$ in the matrix form $\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ Tx & Ty & 1 \end{bmatrix}$ (S_x and S_y are scale factors along x and y direction, respectively, while T_x and T_y are translation factors along x and y directions, respectively), such that

$[x, y, 1] * \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ Tx & Ty & 1 \end{bmatrix} = [longitude, latitude, 1]$, where $|M'| \geq \alpha |M|$ and the 2D point (*longitude, latitude*), belongs to the intersection point set S on the imagery. With this setting, I do not expect point coordinates to match exactly because of

finite-precision computation or small errors in the datasets. Therefore, when checking whether a 2D point *s* belongs to the point set *S*, I declare that $s \in S$, if there exists a point in *S* that is within Euclidean distance δ of *s* for a small fixed positive constant δ, which controls the degree of inaccuracy. The minimum δ such that there is a match of *M'* into *S* is called the *Hausdorff distance*. Different computations of the minimum *Hausdorff distance* have been studied in great depth in the computational geometry community [12]. I do not seek to minimize δ but rather adopt an acceptable threshold for δ. The threshold is relatively small compared to the average inter-point distances in *S.* In fact, this sort of problem was categorized as the "Nearly Exact" point matching problem in [7].

Given the parameters α and δ, to obtain a proper transformation *T*, my approach needs to compute the values of the four unknown parameters S_x, S_y, T_x and T_y. This implies that at least four different equations are required and my approach needs to locate matching point pairs to resolve these four equations. A straight forward (brute-force) method (as the pseudo code shown in Figure 3.5) to resolve the point pattern matching problem is generating all possible matching point pairs from both point

```
Function BruteForcePPM (MapPointSet M, ImagePointSet S)
1.  for each pair m1, m2 ∈ M
2.    for each pair s1, s2 ∈ S                                  //pair-generation phase
3.      Compute the transformation T' mapping m1 → s1, m2 → s2, if one exists
4.      Compute T'(M)                                          //apply T to all map points
5.      if (more than α% of the points in T'(M) match points in S)  //transformation-exam phase
6.          Store the point pair ((m1, s1), (m2, s2)) in the matching point pair record
7.  Pick one point pair from the matching point pair record to compute the transformation T
8.  return T
```

Figure 3.5: Pseudo code of brute force algorithm

sets (called as the "pair-generation" phase) and examining the computed transformations generated by these potential matching point pairs (called as the "transformation-examination" phase). More precisely, the brute-force method is first choosing a point pair (x_1, y_1) and (x_2, y_2) from *M,* Then, for every pair of distinct points (lon_1, lat_1) and (lon_2, lat_2) in *S*, the transformation *T'* that maps the point pair on *M* to the point pair on *S* is computed by solving the following four equations:

$$S_x * x_1 + T_x = lon_1 \quad \text{Eq.(3.1)}$$

$$S_y * y_1 + T_y = lat_1 \quad \text{Eq.(3.2)}$$

$$S_x * x_2 + T_x = lon_2 \quad \text{Eq.(3.3)}$$

$$S_y * y_2 + T_y = lat_2 \quad \text{Eq.(3.4)}$$

Each transformation *T'* thus generated is applied to the entire set of points in *M* to check whether there are more than $\alpha|M|$ points that can be aligned with some points on *S* within the threshold δ. The above-mentioned process is repeated for each possible point pair from *M*, which implies that it could require examining $O(|M|^2)$ pairs in the worst case. Since for each such pair, the algorithm needs to try all possible point pairs on *S* (i.e., $O(|S|^2)$) and spends $O(|M| \log|S|)$ time to examine the transformation *T'* generated. This method has a worst case running time of $O(|M|^3 |S|^2 \log|S|)$. The advantage of this approach is that I can find a mapping (if the mapping exists) with a proper threshold δ, even in the presence of very noisy data. However, it suffers from high computation time. One obvious way to improve the efficiency of the algorithm is to utilize randomization in choosing the pair of points from *M* as proposed in [27], thus achieving the running time of $O(|S|^2 |M| \log|S|)$.

Table 3.1: Summary of notations

Symbol	Meaning
M	the set of detected intersections on the map
$\|M\|$	number of detected map intersections = number of items in M
S	the set of identified intersections on the imagery
$\|S\|$	number of identified imagery intersections = number of items in S
T	the computed transformation that transforms some points from one point set to the corresponding points on the other point set
δ	a distance threshold used to determine whether a transformed map point (image point) matched to an image point (map point).
α	a pre-defined fraction threshold used to define the least percentage of map points (image points) should be mapped to image points (map points), when applying the computed transformation T.

However, their approach is not appropriate for our datasets because the extracted intersection points from maps or imagery could include a number of misidentified intersection points. In addition, there could be some missing intersections on both point sets. Instead, I developed efficient techniques discussed in the following sections to prune the search space of possible point pattern matches (by reducing the numbers of potential matching points needed to be examined).

Table 3.1 summarizes the notations that I use through this section.

3.2.1 Enhanced PPM Algorithm: GeoPPM

Due to the poor performance of the brute-force point pattern matching algorithm mentioned above, I developed a number techniques to improve its performance by exploiting auxiliary information, such as map scale (or map resolution[26]), the degree of intersections (i.e., the number of intersected road segments) and the density of these intersections. The basic idea is to exclude all unlikely matching point pairs. For example, given a point pair (x_1, y_1) and (x_2, y_2) of M, I need to only consider pairs

[26] We can determine the map resolution for a raster map from the known map scale.

(lon_1, lat_1) and (lon_2, lat_2) of *S*, such that the real world distance between (x_1, y_1) and (x_2, y_2) is close to the real world distance between (lon_1, lat_1) and (lon_2, lat_2). In addition, (x_1, y_1) and (lon_1, lat_1) would be considered as a possible matching point if and only if they have similar road degrees and road directions. The entire process[27] to determine potential matching pairs by exploiting map scale, road directions, and intersection density is shown in Figure 3.6. I will discuss this in detail in the following sections. Since the geometric point set matching in two or higher dimensions is a well-studied family of problems with application to different areas such as computer vision, biology, and astronomy [12, 27], I do not intend to invent a novel algorithm to resolve the general point pattern matching problem. Instead, I focus on the datasets I am conflating and particularly design efficient and accurate matching algorithms for the application domain to match the geospatial point patterns. I termed the specialized point pattern matching algorithm as GeoPPM[28].

3.2.1.1 Improvement by Exploiting Map Scale

If the map scale is provided, I can improve the (brute-force) point matching algorithm by exploiting information on direction and relative distances available from the vector sets and maps. The information on direction and distance is used as prior knowledge to prune the search space of the possible mapping between the two datasets. More precisely, for any point pair (x_1, y_1) and (x_2, y_2) on the map, I need to only consider pairs (lon_1, lat_1) and (lon_2, lat_2) in the imagery, such that the ground

[27] A pseudo code in Appendix B presents the algorithm.
[28] I assume that map and imagery are oriented such that north is up.

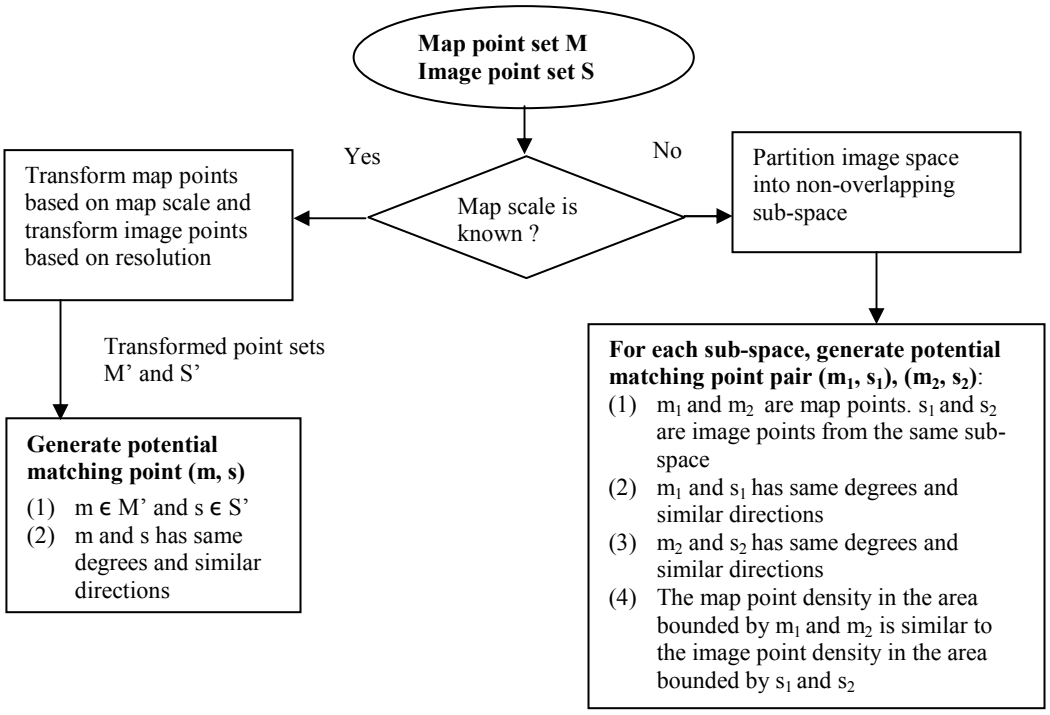


Figure 3.6:Pruning search space by exploiting geospatial information

distance between (x_1, y_1) and (x_2, y_2) is close to the ground distance between (lon_1, lat_1) and (lon_2, lat_2). The ground distance between (x_1, y_1) and (x_2, y_2) is calculated by multiplying their Euclidean distance by map scale. Furthermore, the orientations of (x_1, y_1) and (x_2, y_2) should also be close to the orientations[29] of (lon_1, lat_1) and (lon_2, lat_2). There are $|S|^{1.33}$ such pairs in S [43]. Hence, this enhanced algorithm runs in $O(|M|^3 |S|^{1.33} log|S|)$.

I can further improve the performance by transforming the points on maps and imagery to a 2D Euclidean space, where the distance measurement is ground distance. The real world distance is used between points in the transformed space. Therefore, I only consider translation transformation without scaling in such space.

[29] The orientation consistency is also valid for maps with unknown map scale.

In particular, the process (as shown in Figure 3.7) can be divided into the following subtasks: (1) consider the points on the maps: choose one point P as the origin (0,0), then determine the coordinates of other points Q_i (X_i, Y_i) as follows. X_i is the ground distance between P and Q_i in east-west orientation, while Y_i is the ground distance between P and Q_i in north-south orientation. Note X_i is negative, if Q_i is west of P. Y_i is negative, if Q_i is south of P. (2) repeat the similar transformation to the points on imagery. (3) compare the two point patterns from these two transformed spaces: I now only consider the translation transformation T between the two transformed point patterns. The revised algorithm runs in $O(|M|^2 \ |S| \ log|S|)$. Hence, the running time is significally improved by at least two orders of magnitude. For example, consider $|M|$ equals 57 and $|S|$ equals 1059. The system needs to examine about $3.6*10^9$ (i.e., $57^2 * 1059^2$) potential matching point pairs, whereas it

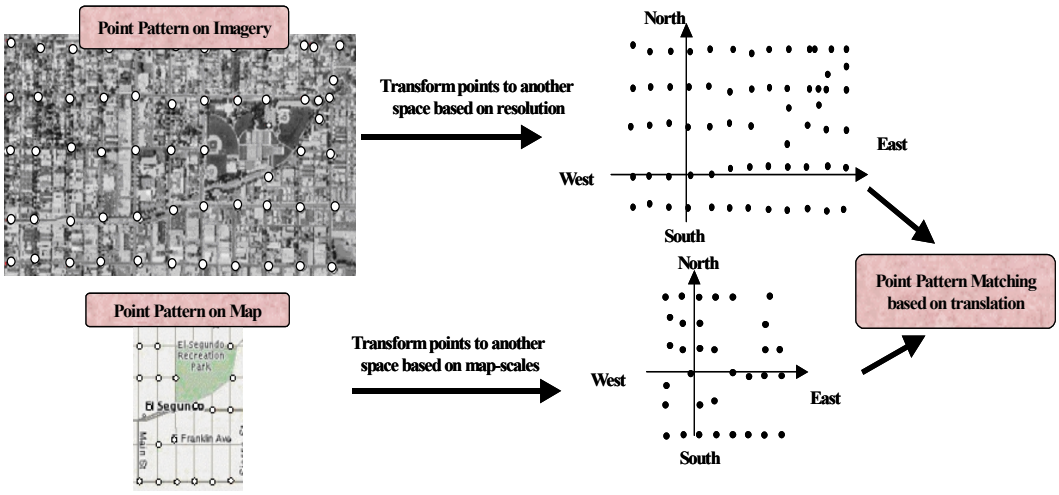


Figure 3.7: Enhanced point pattern matching process using map scales

only has to examine $6*10^4$ (i.e., 57 * 1059) potential matching points while using map scale.[30]

3.2.1.2 Improvement by Exploiting the Geometric Information

In addition to utilizing map scales, I also exploit the degree of a road intersection and directions of road segments. More precisely, for any point *p* on the map, I need to only consider point *q* in the imagery, such that the numbers of intersected road segments (i.e., degree) at both points are the same. Furthermore, the direction difference between the road axis intersected at *p* and the corresponding road axis intersected at *q* must be less than a pre-defined threshold *d* (*d* was set to 20 degrees in my experiments).

3.2.1.3 Improvement by Exploiting Point Density and the Localized Distribution of Points

Although both map scales and geometric information are useful to prune the search space of possible matching point pairs, there are two major issues I need to address:

- There are some online maps with unknown map scales, such as the maps available from ESRI's ArcWeb Map Viewer Application.[31]
- Based on roads degree and directions, the system may still need to examine large numbers of potential matching point pairs. Particularly, in some urban

[30] The assumed numbers of points are based on some of the experiments described in Section 3.4.
[31] http://arcweb.esri.com/sc/viewer/index.html

areas, roads could follow a grid such that most road degrees equal to 3 or 4 and have similar angles.

Hence, I further exploit other information, such as the density of points and localized distribution of points, to reduce the search space further.

For imagery with resolution lower than 16 m/pixel, it is difficult to identify small spatial objects. Hence, it is not practical to conflate a map with images of resolution lower than 16 m/pixel resolution. Furthermore, most of online maps depict detailed and more accurate road network geometry (without simplification or generalization) under certain resolution levels (about 1 to 15 m/pixel) and start to do generalization after about 16 m/pixel. This implies that for medium and high resolution maps, the road networks are depicted with almost all intersections noticeable (as in the Yahoo map examples shown in Figure 3.8). Hence, whether there are any intersections not being identified mainly depends on the map intersection detector performance, and not the map itself. Fortunately, the utilized map intersection detector discussed in Section 3.1.2 works well on medium and high resolution maps.

a) A map with resolution 4.2 m/pixel	b) A map with resolution 14 m/pixel	c) A map with resolution lower than 16 m/pixel (roads are simplified to abstract level)

Figure 3.8: Some sample Yahoo maps with different resolutions

Because there are not many missing points or noisy points in medium or high resolution maps, I can exploit the density of points and localized distribution of points to find the matched point pattern. Without loss of generality, I consider the scenario that the points on the imagery cover a subset of the points on the map and the map scales are unknown in advance. I describe the methods to exploit point density and localized distribution of points in turn.

(1) Point density: The density of the points in the map should be similar to the density of the matched points in the imagery. As an example shown in Figure 3.9, given a point pair P_1 and P_2 of M, I do not need to consider pairs Q_1 and Q_2 of S. This is because the number of points (about 40 points in this example) included in the bounding box Bm (formed by P_1 and P_2) is significantly different from the number of points (about 800 points) in the bounding box Bs (formed by Q_1 and Q_2).

(2) Localized distribution of points: Not only do I consider the point density, but also exploit the point distribution in a localized area. When looking at a real data set, I

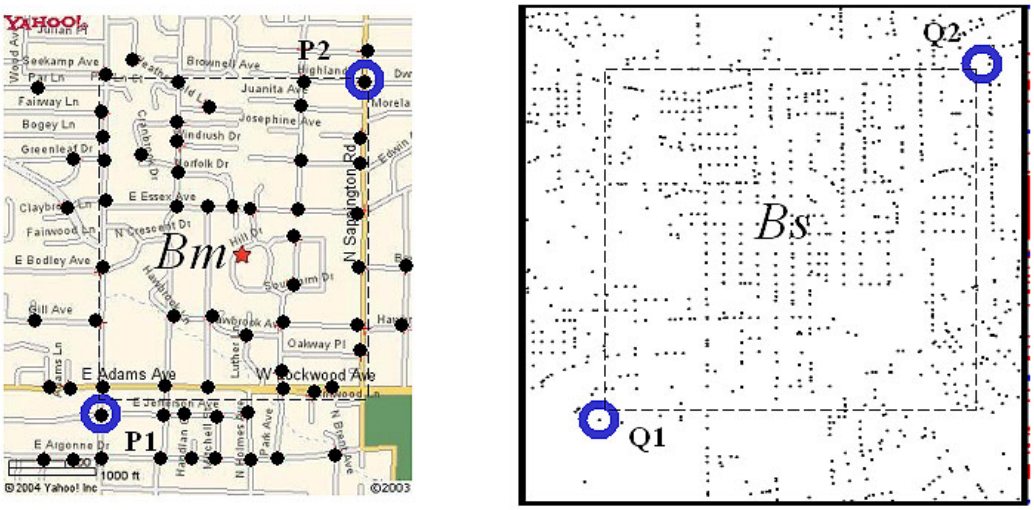


Figure 3.9: An example of utilizing point density to prune the search space

notice that it is not necessary to evaluate the whole search space in one step but it is sufficient to partition the search space into smaller sub-parts and evaluate each independently. The reason is that the points of the matched pattern tend to scatter in neighboring (or localized) areas. Hence, the desired transformation can be computed from some potential matching point pairs locally without considering all pairs from the entire data set. In addition, my objective is not to find the best common point pattern from the map and imagery point sets. In fact, finding any correct matching point pair to form the transformation that can transform the majority of map points to some image points is good enough to solve the point pattern matching problem. Consider the example shown in Figure 3.10. There are 57 detected intersections on the map and there are 1059 intersections on the image[32]. The image space is partitioned into 16 equi-sized cells (e.g., cells AFQB, BQTC, etc). In order to explain my algorithm, the matched point pattern on the imagery is highlighted by a dashed rectangle and I also mark some matching point pairs (e.g., the points m_1, m_2 ,m_3, m_4, m_5 and m_6 on the maps correspond to the six points s_1, s_2 ,s_3, s_4, s_5 and s_6 shown in the enlarged dashed-area of Figure 3.10(c)).

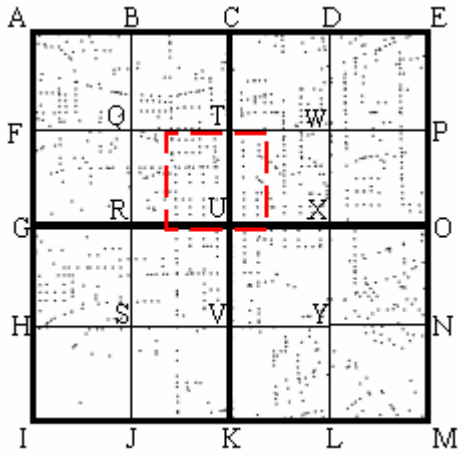
There are two steps to locate matching point pairs using this grid structure:

Step 1: The system first chooses a point pair P_1 (x_1, y_1) and P_2 (x_2, y_2) from *M*. Then, for every pair of distinct points Q_1 (lon_1, lat_1) and Q_2 (lon_2, lat_2) in the same cell (e.g., cell AFQB), the system computes the transformation *T*, if

[32] I removed the background imagery in order to clearly display the distribution of points on the imagery.

- there is similar point density in the bounding boxes formed by (P_1, P_2) and (Q_1,Q_2), respectively.
- the road directions of P_1 (P_2) are similar to the road direction of Q_1 (Q_2).

The system then applies transformation T to the entire points in M to check whether there are more than $\alpha|M|$ points that can be aligned with some points on S within the threshold δ. The above-mentioned process is repeated for each possible point pair from M.



a) Partition the image space (with 1059 intersections) into 16 equi-sized cells. The corresponding area to the map is highlighted.

b) 57 detected intersections on a map

c) The enlarged overlapping area in the imagery

Figure 3.10: An example of utilizing localized point distribution to prune search space

With proper settings of α and δ, it is very likely that my approach can obtain some correct matching point pairs, such as $((m_2,s_2), (m_5,s_5))$[33] or $((m_3, s_3),(m_4, s_4))$ in cell QRUT or $((m_1,s_1),(m_6, s_6))$ in cell TUXW. This saves running time, because the approach does not need to examine the point pairs that are located in different cells (e.g., the image points s_1 and s_4).

Step 2: In the worst case, the approach may not be able to locate any matching point pair after examining each cell. It then repeats the first step, but checks the points in the cells of one level higher (e.g., cells AGUC, CUOE, etc). If it still fails to find any matching point pair, it searches in an even higher level (i.e., grid AIME). The process stops whenever the approach finds some matching point pairs or whenever it reaches the highest level (i.e., the entire point set). Since the points of the matched pattern tend to scatter in localized areas, it has higher possibility that my approach can find some matching point pairs in the cells of lower levels. Hence, it can avoid searching matching point pairs in cells of higher levels.

I developed a hierarchical grid structure (called HiGrid) to implement the above-mentioned idea. An example of HiGrid where the system recursively subdivides the space into four sub-spaces to the depth 3 is shown in Figure 3.11.

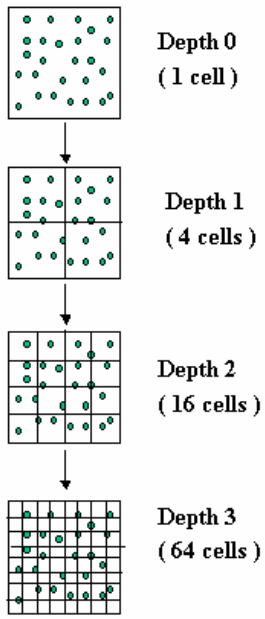


Figure 3.11: An example of HiGrid

[33] The matching point pairs notation $((m_i, s_i),(m_j, s_j))$ implies that m_i matches s_i and m_j matches s_j.

Table 3.2: An example of matching point pair record

d_i (meters)* / Matching point pairs	d_0	d_1	d_2	……	d_j ** (i.e., δ)	d_{j+1}	……	d_l (i.e., 2* δ)
$[(m_2, s_2), (m_5,s_5)]$	5***	13	25		45	47		49
$[(m_3, s_3), (m_4, s_4)]$	4	12	22		46	48		50
$[(m_1, s_1), (m_6, s_6)]$	5	12	24		46	47		49
: :								

* In my experiments, I set d_0 to 3 m. d_{i+1} equals to d_i +3.

** Note: d_j (i.e., δ) is set to 30 meters, based on experiments with some random sample datasets.

***Note: The numbers in the gray cells are the amount of matched points using specific threshold d_i. For example, consider the number "5" in the cell ($[(m_2, s_2), (m_5, s_5)]$, d_0), where the potential matching $[(m_2, s_2), (m_5,s_5)]$ is used to generate transformation T. After applying T to all map points, there are *five* points belongs to some image points within the distance threshold d_0 *meters*.

In practice, my algorithm maintains a table to record matching results. The example reproduced in Table 3.2 shows a matching point pair table that stores the matching results for all possible matching point pairs examined so far in each cell. Note that the algorithm not only records the number of map intersections that can be aligned with some intersections on imagery within the threshold δ (i.e., the number 45, 46, etc. in Table 3.2), but also records the number under different threshold settings (i.e., different d_i in Table 3.2). The values of these thresholds vary from d_0 m to 2*δ m (In my experiments, I set d_0 to 3 m. d_{i+1} equals to d_i +3 and δ is 30 m). This additional information can be utilized in a dynamic threshold adjustment algorithm described in Section 3.2.2. Using this table, the algorithm can avoid performing any redundant examination of matching point pairs in higher-level cells. The remaining problem is how to determine the depth k (the highest level has the depth zero) of HiGrid, during its construction. k is calculated based on the number of points in the imagery. Assume that I have $|S|$ points in the image and I partition the grid into b sub-grids when building the HiGrid structure. In addition, assume that the points are

uniformally distributed over the space and I intend to have *n* points (on average) for each cell of the lowest level. Using this term, I infer the inequality:

$$n*b^{k} \leq |S| < n*b^{k+1} \qquad \text{Eq.(3.5)}$$

This implies that the depth *k* of the HiGrid structure is $\left\lfloor \log_b \frac{|S|}{n} \right\rfloor$.

Utilizing HiGrid results in an efficient, systematic and hierarchical way to search for matched point patterns in local (i.e., small) regions. Furthermore, each cell of the same level is independent (i.e., there is no overlaps) and can be processed in parallel. The disadvantage of HiGrid is that it employs equi-sized cells at the same level, and the point density of each cell may vary (because the real world data may not obey a uniform distribution) and it may not have a similar point density to that of the map. However, if the approach does not find any matching point pairs in the lower levels, it will search the upper levels. Hence, I can avoid these problems by utilizing the hierarchical structure. In the worst case, the system may need to search for the pattern in the entire space. Under this extreme scenario, the approach still can reduce the running time by applying the point density checking in bounding boxes (as the example shown in Figure 3.9). From my experiments discussed in Section 3.4, this approach significantly reduces the execution time and locates the accurate pattern.

In sum, my approach utilizes these exploited geospatial information (e.g., map scale, road directions and density of intersections) simultaneously to prune search space. More precisely, if the map scale is known in advance, the approach utilizes map scale and road intersection directions to identify the common point pattern. Otherwise, it locates the point pattern by using the HiGrid structure, point density

and road intersection directions. That is, in my current implementation for GeoPPM, I do not use HiGrid for maps with known map scales (due to the fact that I have derived good performance by using just the map scale, as described in Section 3.4.3.1). In general, I can also utilize HiGrid structure and point density for maps with known map scales.

3.2.2 Other Enhancements for GeoPPM

So far I have described how to exploit additional information to improve GeoPPM. In this section, I will discuss the issues that will affect the accuracy of GeoPPM and I describe my solutions to overcome these issues.

There are three major reasons why it may fail to locate the matched point pattern (if such pattern exists).

- Intersection consistency issue: Some road intersection degree and directions are inconsistent between the map points and the corresponding imagery points. Although degree and directions are helpful information, sometimes they may mislead the matching process, due to this kind of data inconsistency. This inconsistency occurs because of the various resolutions of diverse geospatial datasets, the ways of computing road directions and noisy information of each dataset. With some random samples tested in my experiments, I found that 10.2% of the map intersections have different degrees from the corresponding imagery points. Moreover, considering the road direction dissimilarity as well, more than 18.3% of the map intersections have different directions from the corresponding

imagery points. This is even worse with low resolution maps (e.g., less than 10 m/pixel) and imagery. In particular, this is because the map intersection detector does not perform well on low resolution maps. Therefore if the GeoPPM algorithm cannot locate the matched pattern using degree and directions, it will relax the degree and directions constraints and search again.

- Similar point distribution issue: The examples of similar point distribution over the neighboring regions could be found in urban areas where some roads are in a grid shape with similar block distances, similar intersection degrees and directions. This is a challenging situation. Without knowing any other attribution information (e.g., road names), my approach can only alleviate this problem by focusing on larger maps where there is more likely to have a unique pattern of points.
- Setting of the thresholds α and δ: For each type of map, the parameters α and δ, two fixed constants used in the point pattern matching algorithm, are determined by trying a few values and examining the results. However, these parameters cannot be formulated in a general way, since these values are dependent on the datasets, data quality, and resolutions. Using the same thresholds for different resolution maps or even different types of maps may result in either finding multiple matching point pairs or not finding any matching point pairs at all.

For the first scenario (i.e., finding multiple matching point pairs), I perform a refinement process to obtain the best matching point pair among them. For instance, Table 3.2 (an example of the matching point pair record) includes a list

of all possible matching point pairs. Among these possible matching point pairs, I only consider the matching point pairs that meet the threshold requirements (i.e., the settings of α and δ). I define a weighted score for these matching point pairs that meet the threshold settings:

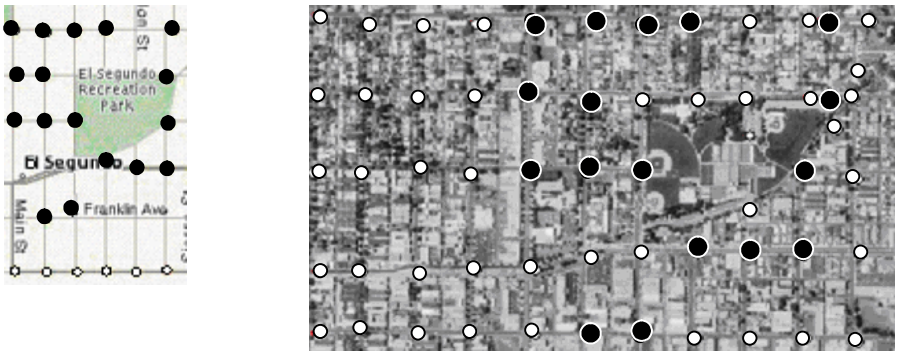
WeightedScore = $\sum_{i=0}^{j} w_i * (n_i - n_{i-1})$, where n_i is the number[34] of matched points when the threshold is set to d_i (e.g., the number in gray cells of Table 3.2). d_j equals to δ and n_j is greater than or equals to $\alpha * |M|$. w_i is the weight which is inversely proportional to d_i (i.e., smaller d_i has higher weight w_i).

My system calculates the weighted score for these matching point pairs that meet the threshold settings of α and δ. Subsequently, the system uses the weighted score to sort these possible matches and picks the one with the highest score (then computes the transformation). By calculating weighted scores, I can overcome this issue of ambiguity (because multiple candidate matching point pairs are found).

For the second scenario (i.e., not finding any matching point pairs), I utilize a threshold adjustment algorithm. I fix the parameter α and dynamically adjust the threshold δ. More precisely, if GeoPPM cannot find a match under the initial value of δ and α, it would increase δ, and then search again. This process repeats until GeoPPM finds a matching point pair or reaches the upper bound of possible thresholds, which then will return with no match found. This may seem that it

[34] Note that $n_k = 0$, if $k<0$.

increases the running time due to the adjustments. However, the numbers of matched points due to the different threshold settings are stored in the matching records (as in the example shown in Table 3.2) when GeoPPM is first executed. Hence, if a potential point pair is examined and stored in the matching records, the algorithm does not have to examine that point pair again. The computation first becomes a table-lookup. If there is no such record, then GeoPPM examines this new possible matching point pair.



a) A map with matched point pattern heightened as black circles

b) An image with corresponding point pattern heightened as black circles

Figure 3.12: A sample result of GeoPPM

3.3 Aligning Map with Imagery

Now that I have a set of control point pairs for the map and imagery (as in the example shown in Figure 3.12), I can deform one of the datasets (the source image) to align the other (the target image) utilizing these identified control point pairs. Without loss of generality, I assume that the map is the source image, while the orthoimage is the target image.

To achieve overall alignment of an image and a map, the system must locally adjust the map to conform to the image. To accomplish local adjustments, the system

partitions the domain space into small pieces. Then, it applies local adjustments on each single piece. Similar to the vector-imagery conflation, the system performs the space partition technique, Delaunay triangulation, with the set of control points on the map, and makes a set of equivalent triangles with corresponding control points on the imagery. Then, it uses the "Rubber sheeting" technique to deform the map pixels algorithmically, forcing registration of control points on the map with their corresponding points on the imagery. As discussed in Section 2.3.2, the system first calculates the transformation coefficients to map each Delaunay triangle on the map onto its corresponding triangle on the imagery. Second, for each pixel in each triangle on the imagery, the system replaces it semi-transparently with the corresponding pixel on the map by using the computed transformation coefficients.

Figure 3.13 shows an example of Delaunay triangulation, and the arrow illustrates that the pixels of the triangle on the imagery would be (semi-transparently) overlaid by the corresponding pixels on the map (i.e., rubber-sheeting). In practice, if the conflation area (i.e., the convex hull formed by control points) of the source

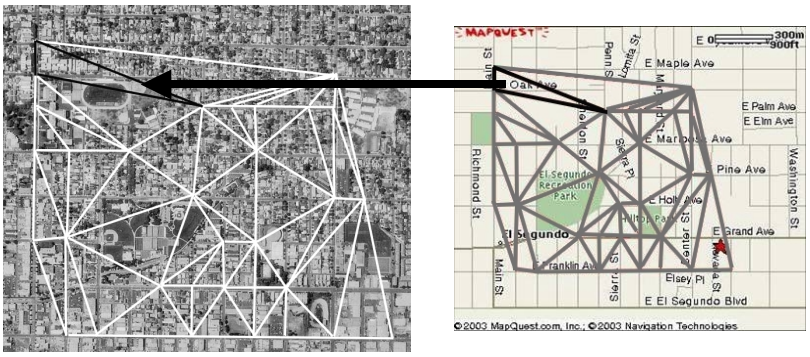


Figure 3.13: Delaunay triangulation on imagery and a map, using matched point pairs as control point pairs

image is much larger than that of the target image, the rubber-sheeting results will be distorted because the sampling frequency is insufficient. I solve this problem by rescaling (resampling) the conflation area on the map and imagery to identical sizes before applying triangulation and rubber-sheeting.

3.4 Performance Evaluation

I utilized a set of online street maps and imagery to evaluate my approach. The purpose of the integration experiment was to evaluate the utility of my algorithms in integrating real world data. I am interested in measuring the accuracy of the integration of maps and imagery using my techniques. To that end, I performed several experiments to validate the hypothesis that using my techniques we can automatically and accurately align maps and imagery.

3.4.1 Experimental Setup

Table 3.3 summarizes the datasets and test sites used for my experiments. I described these datasets in turn.

Table 3.3: Tested datasets used in experiments

	Test data set 1 (El Segundo, CA)	**Test data set 2 (St. Louis, MO)**
Imagery	Geo-referenced USGS DOQ orthoimagery with 1m/pixel resolutions	Geo-referenced USGS high resolution color orthoimagery with 0.3m/pixel resolution
Maps (with various sizes/scales)	5 ESRI maps, 5 MapQuest maps, 5 Yahoo maps, 5 TIGER maps, 5 USGS Topographic maps,	5 ESRI maps, 5 MapQuest maps, 5 Yahoo maps, 5 TIGER maps, 5 USGS Topographic maps
Vector data	U.S. Census TIGER/Lines Length: 84.32km About 300 intersections	USGS MO-DOT road vector Length: 364.28km About 1130 intersections
Area covered	Latitude:33.9164 to 33.9301 Longitude:-118.4351 to -118.3702 Width: 5.2km Height: 1.6km	Latitude: 38.5534 to 38.6091 Longitude: -90.4389 to -90.3720 Width: 6 km Height: 6 km

(1) Orthoimagery

The imagery used in the experiments is the same as that used for vector-imagery conflation evaluation. That is, I tested the 0.3 m/pixel color imagery that covers some areas of the county of St. Louis, MO, and the 1 m/pixel gray-scale imagery that covers some areas of the city of El Segundo, CA. Figure 3.14 shows about 0.6% of the imagery used in test data set 2.

(2) Street maps

I used streets maps queried from five different online map services. They are ESRI map,[35] MapQuest map,[36] Yahoo map,[37] U.S. Census TIGER map[38] and USGS

Figure 3.14: Sample imagery in test area 2

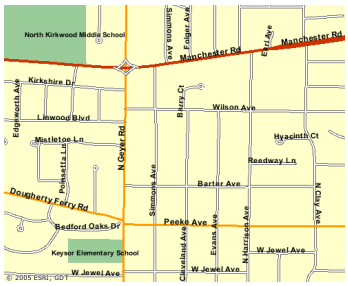
[35] ESRI provides various online map services. In order to evaluate my proposed map-imagery conflation technique for maps with unknown map scale, I used the ESRI maps available at http://arcweb.esri.com/sc/viewer/index.html in the experiments. Neither map scale nor geo-coordinates of ERSI maps are provided from this web site.
[36] http://www.mapquest.com
[37] http://maps.yahoo.com/
[38] http://tiger.census.gov/cgi-bin/mapsurfer

topographic map.[39] Although these street maps mainly depict roads, sometimes they also show and name prominent natural and cultural features, such as rivers, parks and schools. I described these street maps (e.g., the sources of the road information) as follows.

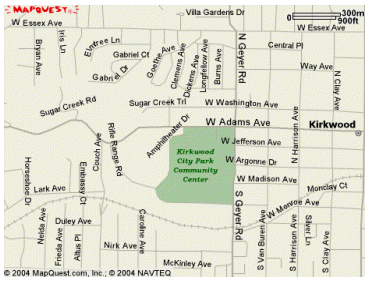
- ESRI map used in the experiments is generated based on the GDT data from Geographic Data Technology (GDT). It is high quality street map data with highly accurate street geometry. Neither map scale nor geo-coordinates for ESRI maps are provided online.
- MapQuest map and Yahoo map are produced based on NAVTEQ NAVSTREETS. They are also high quality street map data with highly accurate street geometry and they illustrate street maps in diverse map scales, sizes and colors.
- U.S. Census TIGER map is generated from U.S. Census TIGER/Line files (discussed in Section 2.4.1). TIGER map has both poor positional accuracy and road geometry.
- USGS topographic map depicts roads, prominent natural and cultural features of an area. It also depicts contour lines to show elevation differences. Such detail is useful for local area planning and helpful to hikers (because this map can show elevation changes along a trail).

Note that only TIGER map server provides the map geo-coordinates. The maps evaluated in these experiments involve various map resolutions (or map scales)

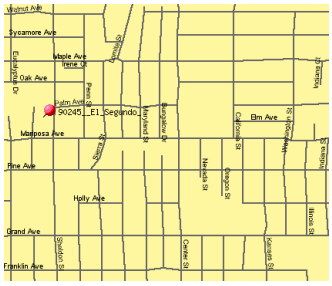
[39] http://terraserver-usa.com/



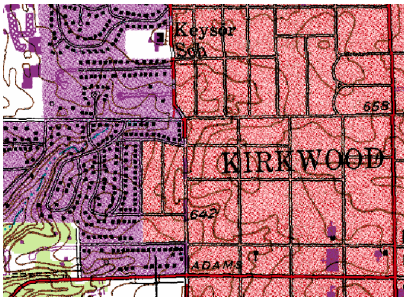
a) An ESRI map with unknown map scale



b) A MapQuest map with resolution 4.8 m/pixel



c) A TIGER map with resolution 4.17 m/pixel



d) A topographic map with resolution 2 m/pixel

Figure 3.15: Samples of different street maps used in the experiment

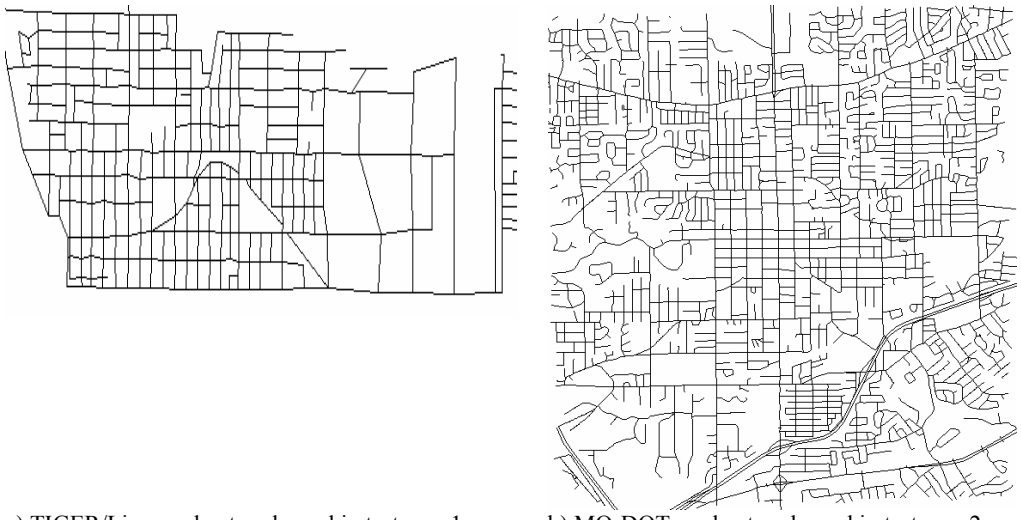
ranging from 1.2 m/pixel to 14.08 m/pixel. Figure 3.15 shows some examples of these street maps.

(3) Vector data (road networks)

Two different road networks were used as "glue" to align maps with imagery: U.S. Census TIGER/Lines is utilized in test data set 1, while USGS MO-DOT data is used in test data set 2. Figure 3.16 shows the road networks utilized.

3.4.2 Evaluation Methodology

The evaluation of map-imagery conflation has not been studied before. Hence, I developed a novel evaluation method to measure how well the features on the map align to the corresponding features on the imagery. In particular, I consider how well the conflated map roads align to the corresponding roads on the imagery. Toward that end, I "vectorize" the conflated map road pixels, and utilize the same evaluation



a) TIGER/Line road network used in test area 1 b) MO-DOT road network used in test area 2

Figure 3.16: The road networks used in experiments

schema described in vector-imagery conflation (i.e., Section 2.4.2) to compare the conflated (and vectorized) map road network with the reference imagery road network by measuring (1) *Completeness: the percentage of the reference roads in imagery for which my approach generated conflated map roads,* (2) *Correctness: the percentage of correctly conflated map roads with respect to the total conflated map roads*, (3) *Positional accuracy: the percentage of the total length of the conflated map roads that is within a specific distance of the reference roads*.

Additionally, I developed two metrics, precision and recall, to measure the performance of our GeoPPM technique, since the accuracy of matched points significantly affects the conflation results.

Let the point pattern generated by GeoPPM defined as a set:

Ret_{Pat}={(m_i, s_i) | where m_i is the point on the map and s_i is the corresponding imagery point located by GeoPPM.}

To measure the performance of GeoPPM, I need to compare the set Ret_{Pat} with respective to the real matched point pattern set Rel_{Pat} (defined in Section 3.2).

Using this term, I define

$$\text{Precision}=\frac{|Ret_{pat} \bigcap Rel_{pat}|}{|Ret_{pat}|} \qquad \text{Eq.(3.6)}$$

$$\text{Recall}=\frac{|Ret_{pat} \bigcap Rel_{pat}|}{|Rel_{pat}|} \qquad \text{Eq.(3.7)}$$

In my experiments, the set Ret_{Pat} qualifies as a matched point pattern if and only if precision is greater than 80% and recall is higher than 60%. I set higher precision, because the conflation process does not require a large number of control point pairs to perform accurate alignment. In fact, a smaller set of control points with higher accuracy would be sufficient for the conflation process.

I conducted the following experiments. I first obtained online orthoimages covering the experimental areas and identified road intersection points on the images by utilizing information inferred from the vector dataset (as described in Section 3.1.1 and the performance evaluation illustrated in Section 2.4). Then, I randomly downloaded various street maps (with diverse sizes and map scales) within these areas from the above-mentioned five map services and extracted an intersection point set for each map. Then, GeoPPM computed the alignment between the point set on each map with the point set on the image. Finally, my system aligned each map and imagery based on the matched point pattern. I evaluate the conflation results by the

evaluation methodology discussed and acquired the experimental results as described in the following section.

3.4.3 Experimental Results and Interpretations

I discuss the performance of GeoPPM algorithm and the performance of the overall map-imagery conflation as follows.

3.4.3.1 Performance of GeoPPM

After conflating road vector data with imagery, the system identified 281 intersection points on the image of test data set 1 (with 88.6% precision and 86.1% recall) and 1059 intersections on the image of test data set 2 (with 91.6% precision and 88.3% recall[40]). Because the tested maps are of diverse sizes and scales, the number of points detected on each map is different. On average, there are about 60 points on each map and the system achieved 92% precision and 67% recall (on average) for identifying road intersections on different maps.

When applying GeoPPM to these detected point sets, my system exploited additional information to improve the performance. In particular, if the map scale is known in advance, the system utilized map scale and road intersection directions to identify the common point pattern. Otherwise, the system located the point pattern by using the HiGrid structure and road intersection directions. That is, in the experiments, my system did not use HiGrid for maps with known map scales (due to

[40] The precision and recall of intersection points on the imagery are computed based on Eq.(2.2) and Eq.(2.3). Since most of the intersections on the imagery have corresponding intersections on road vector data, there is only slightly difference between precision and recall.

the fact that the system has derived good performance by using just the map scale). In general, my approach can also utilize HiGrid for maps with known map scales. In addition, for the identified point pattern, my system applied a post-filter to eliminate the matched points that have different degrees or significantly different angles. Higher precision of control points is more important than higher recall for the conflation process. By utilizing the post-filter, I improved the precision at the cost of reducing the recall. Since the map intersection detector did not perform well to compute road directions in low resolution maps (e.g., less than 10 m/pixel), GeoPPM cannot rely on this direction information. Hence, for low resolution maps, GeoPPM examined only point degree and did not check the directions. Moreover, GeoPPM did not apply the post-filter to low resolution maps either, due to the same reason.

Table 3.4 shows the performance of GeoPPM with respect to different scenarios. There is only one of our fifty tested maps (i.e., 2%) where the intersection point set is not accurately aligned with the corresponding point pattern on the image. This map is a 1.85 m/pixel resolution TIGER-map with 13 detected intersections (see Figure 3.17(a)). As shown in Figure 3.17(c),[41] the identified point pattern on the image was shifted one block to the right. We can notice that this misalignment is because the roads on this mis-aligned map is in grid shape with similar block distances and it covers a smaller area compared with other maps.

[41] I removed the background imagery in order to clearly display the distribution of points on the imagery.

Table 3.4: The performance of GeoPPM

	ESRI map	MapQuest map	Yahoo map	TIGER map	Topographic map
Precision	96.0%	95.2%	94.0%	84.2%*	93.9%
Recall	80.2%	84.8%	88.3%	75.6%*	80.94%

a). GeoPPM performance with respect to different map services

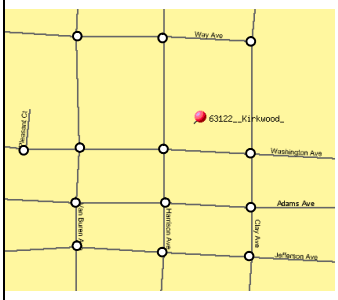
	Test data set 1 (El Segundo, CA)	**Test data set 2 (St. Louis, MO)**
Precision	91.9%	93.4%**
Recall	84.6%	77.4%**

b). GeoPPM performance with respect to different regions

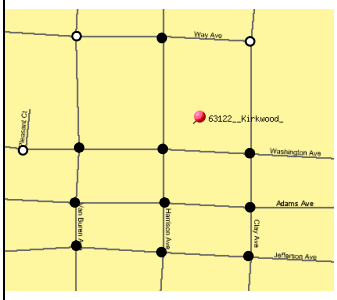
	Precision	**Recall**
Resolution $\leq$ 2 m/pixel (15 maps)	87.4%***	78.2%***
2 m/pixel $<$ **Resolution** $\leq$ 4 m/pixel (7 maps)	92.9%	84.0%
4 m/pixel $<$ **Resolution** $\leq$ 7 m/pixel (13 maps)	96.4%	88.6%
Resolution $>$ 7 m/pixel (5 maps)	91.6%	77.1%

c). GeoPPM performance with respect to different resolution maps

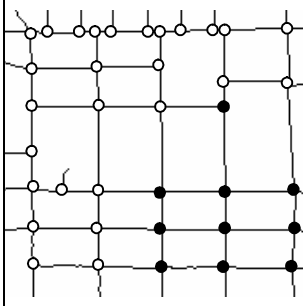
* If we exclude the misaligned TIGER-map, the precision is 93.6% and recall is 84.2%.
** If we exclude the misaligned TIGER-map, the precision is 97.2% and recall is 80.6%.
*** If we exclude the misaligned TIGER-map, the precision is 93.2% and recall is 82.5%.



a) Detected map intersections (white circles)



b) Identified point pattern on the map after applying GeoPPM (point pattern marked as black circles)



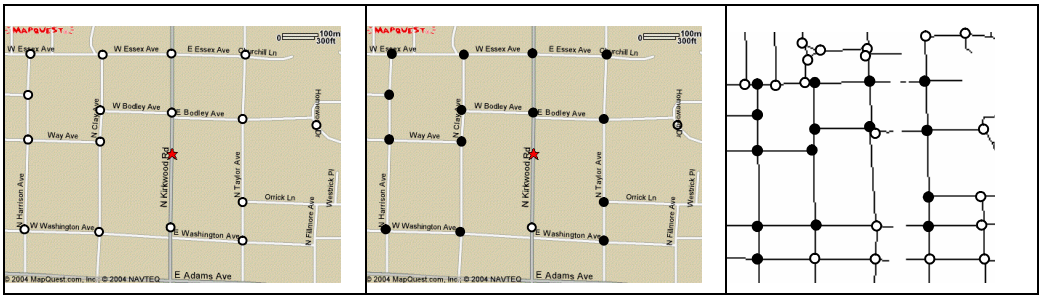
c) Identified point pattern on the imagery after apply GeoPPM (point pattern marked as black circles)

Figure 3.17: The map whose point pattern does not align with the corresponding point pattern on the imagery

The maps available on TIGER map service and MapQuest are in fixed dimensions. The covered area becomes smaller whenever one zooms in the area of interest. If these small maps contain a unique point pattern across the dataset, GeoPPM can still identify the matched pattern from the maps even with very few points (as in the MapQuest example shown in Figure 3.18). However, sometimes, there is no unique pattern in the points of such large-scale, very small maps (as Figure 3.17(a)). My approach can achieve higher accuracy by focusing on larger maps where there is more likely to be a unique pattern of points.

In general, I make the following observations from Table 3.4:

- GeoPPM performs well with respect to maps queried from diverse online map services. It has the worst performance over TIGER maps because it found a mis-aligned point pattern from one of the TIGER maps (i.e, with precision 0% and recall 0%).
- There is no significant difference in the performance over various resolutions of maps. Even for low resolution maps, GeoPPM still obtained high precision



a) Detected 16 map intersections (white circles)	b) Identified point pattern after apply GeoPPM (point pattern marked as black circles)	c) Identified point pattern on the imagery after apply GeoPPM (point pattern marked as black circles)

Figure 3.18: The map whose point pattern aligns with the corresponding point pattern on the imagery

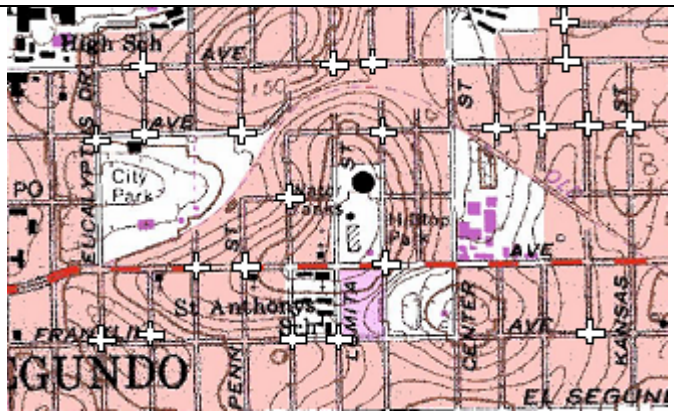
and recall. However; there are four low resolution maps (lower than 7 m/pixel) with very low recall rates for detected map intersections (with precision 79.4% and recall 21.2%, on average), due to the poor performance of the map intersection detector on low resolution maps. This will affect the conflation results, even though GeoPPM obtained high precision and recall from these detected map points.

Moreover, there are two major reasons why GeoPPM did not achieve 100% precision for the high to medium resolution maps:

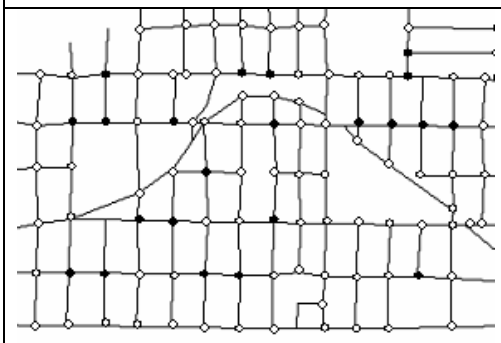
- GeoPPM can tolerate the existence of misidentified intersections in the maps. In the example shown in Figure 3.19(a), GeoPPM filtered the misidentified points in the lower left corner (as shown in Figure 3.3). However, for the intersection points that are very close to the map intersections, it is difficult to filter out those points. The point, as highlighted in Figure 3.19(c), was identified as a matched point, since it is very close to the real map intersection and it even has similar degree and directions as the corresponding imagery point.
- Similarly, GeoPPM may detect some points that are misaligned with the imagery points, but very close to the real imagery intersections.

3.4.3.2 Performance of overall map to imagery conflation

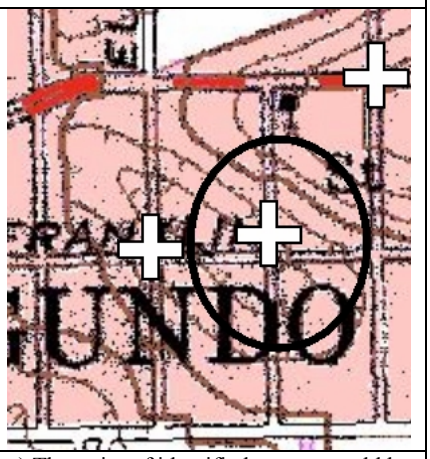
After applying GeoPPM, the system generated an accurate control point pair set for each map. Then, my approach used these control points to conflate the maps with



a) Identified point pattern after apply GeoPPM (point pattern marked as cross). If we compare this figure with Figure 3.3, the noisy map points at the left lower corner are filtered out.



b) Identified point pattern on the imagery after apply GeoPPM (point pattern marked as black circles)



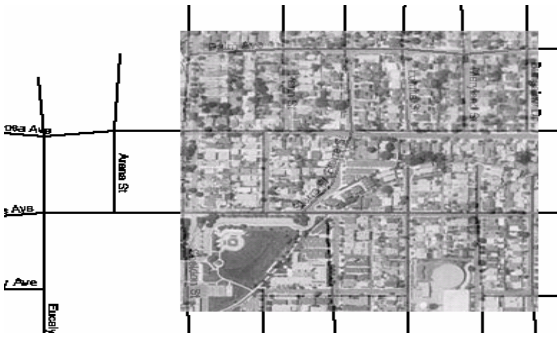
c) The point of identified pattern could be very close to a map intersection (after zooming in the left lower corner of (a))

Figure 3.19: An example of matched point pattern

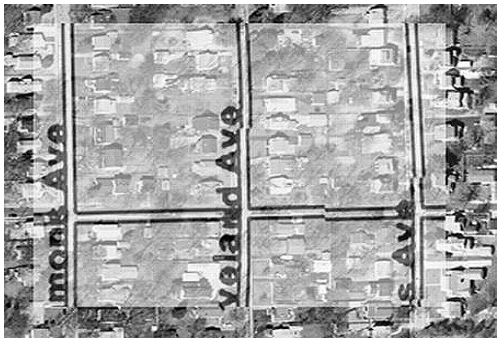
imagery. To demonstrate the accuracy of my conflation techniques, some results are shown in Figure 3.20. As shown in these aligned images, the system can annotate spatial objects (e.g., streets) on imagery with the attribution information contained in maps.

I also conducted a quantitative analysis to the conflation results. Towards that end, I randomly selected a set of TIGER maps and imagery from the test data set 2. These selected maps and imagery cover 8.3% of the tested area. Furthermore, after

a) MapQuest map to imagery conflation (semi-transparent map) for El Segundo, CA



b) TIGER map to imagery conflation (semi-transparent image) for El Segundo, CA



c) ESRI map to high resolution imagery conflation (semi-transparent map) for St. Louis, MO

d) MapQuest map to high resolution imagery conflation (semi-transparent map) for St. Louis, MO

Figure 3.20: Examples of map-imagery conflation results

applying GeoPPM against the tested TIGER maps and imagery, GeoPPM accurately obtained aligned control point sets (with 100% precision and 82.7% recall). The reasons why I choose TIGER maps are:

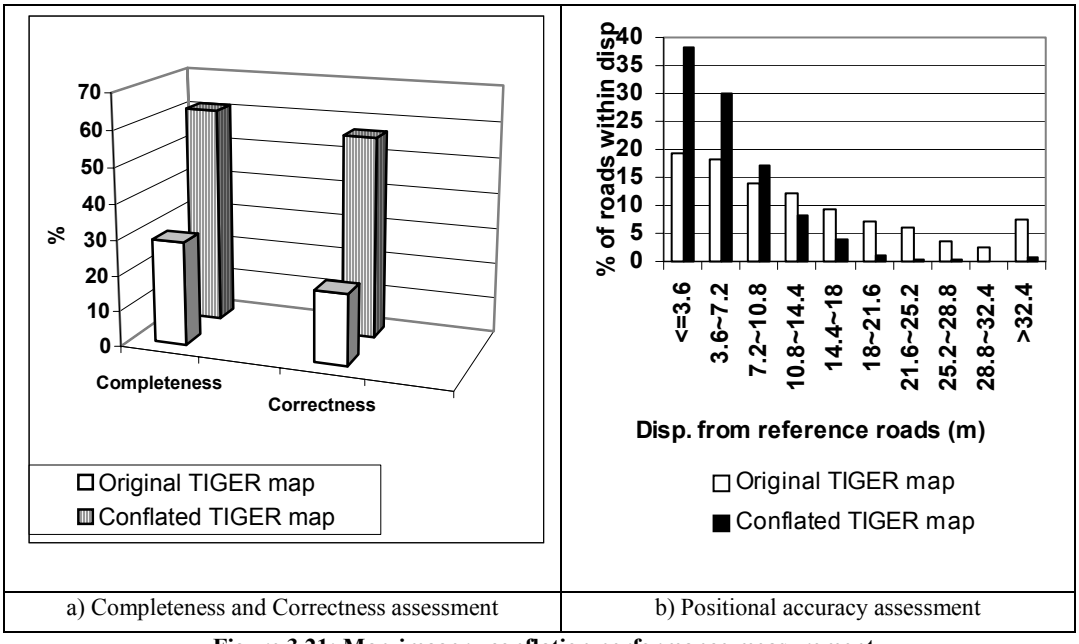
- The geographic coordinates are provided by the data source. Therefore, I can simply combine the TIGER maps with the corresponding imagery based on the provided geographic coordinates. The integration results were then compared with the conflation results by utilizing my approach.
- I do not have to specify (i.e., vectorize) the streets on TIGER maps manually for the evaluation purpose, since I can utilize road vector dataset TIGER/Lines as the vectorized map roads. The roads on the TIGER maps

align well with the TIGER/Lines, because they are generated from TIGER/Lines.

Using these vectorized map roads and applying map-imagery conflation, I then evaluate the results by measuring completeness, correctness and positional accuracy against the same reference roads used in the vector-imagery conflation experiments. The experimental results are shown in Figure 3.21. My approach improved the original TIGER map alignment about 2.5 times for completeness and correctness.

There are three reasons why the completeness and correctness are not that high:

- The errors from original TIGER-maps: For a particular road segment, if the shape of the original TIGER map road is inconsistent with roads in the imagery, my approach may not align them well (although the intersections might be aligned using GeoPPM).

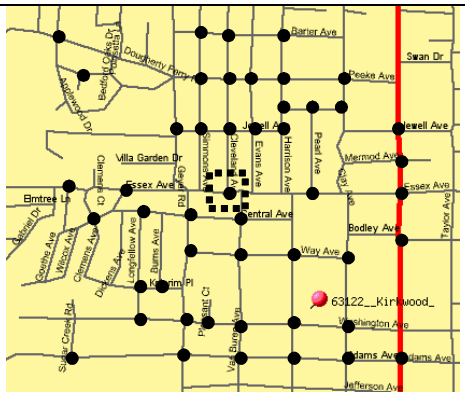


a) Completeness and Correctness assessment

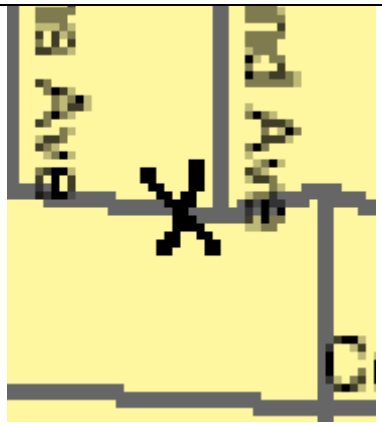
b) Positional accuracy assessment

Figure 3.21: Map-imagery conflation performance measurement

- The errors from the resizing: The TIGER-map used in my experiment is 4.17 m/pixel, while the imagery is 0.3 m/pixel. After finding the matched point pattern, the system deformed and resized the map to align the imagery. Due to the large difference in resolution for the two datasets, some errors will be amplified after resizing. Consider the detected road intersection point shown in Figure 3.22(a) and (b). This point is characterized as an accurately detected



a) Detected point pattern on the map (one point is highlighted by dash rectangle)



b) The location of detected point is marked as "X", after zooming in the highlighted area of (a). It is two pixels away from the actual intersection location.

c) The black circle represents the matched imagery point. Its location is very accurately (the solid lines are road vector data and the dash lines are road sides)

d) The two pixels displacement will be amplified due to resizing (the solid regions are map roads and the dash lines are road sides)

Figure 3.22: Explanations of the conflation errors

map point, because it is two pixels away from the exact position of intersection (less than the threshold five pixels). Although the imagery intersection is so accurately identified (see Figure 3.22(c)), the resizing process will make the "2 pixels" displacement become "30 pixels" (as in Figure 3.22(d)).

- The errors from the vectorization process: Although the road vector TIGER/Lines align well with the map road, there are still a few pixels difference between them. When utilizing TIGER/Lines as the map roads and resizing to the imagery size, the small errors will also be amplified.

When relaxing the "buffer-width" used to measure completeness/correctness, I can obtain higher completeness/correctness. This kind of assessment is illustrated by the positional accuracy shown in Figure 3.21(b). Intuitively, the "positional accuracy" corresponds to the users' demand: *how far is the conflated road network on the map from the centerlines of the real (reference) road network*. I evaluated these displacements between two networks by gradually increasing the buffer-widths constructed around the reference road network. The buffer-width was increased by 3.6 m. As shown in the X-axis of Figure 3.21(b), the displacement values are grouped every 3.6 m. The Y-axis shows the percentage of conflated map roads lying within the displacement range represented by the X-axis. Although my approach did not achieve high completeness/correctness (as stated earlier), it achieved better positional accuracy: 85.2% of the conflated map roads are within 10.8 m of the reference roads compared to 51.7% of the original TIGER map. Furthermore, there

are very few road pixels (0.7%) have more than 32.4 m displacement for conflated roads, compared with 7.5% for the original map. This implies that the conflated map roads are very close to the real roads, although they might not be within the road sides (i.e., road buffer widths).

3.4.3.3 The execution time

Finally, I present the running time of my conflation algorithm. Since the running time of my techniques is mainly dominated by the point matching routine, I use the running time of the GeoPPM routine as the overall execution time (the query time for retrieving online images or maps was not included). In addition, the running time of the GeoPPM algorithm mainly depends on the number of road intersections on the maps, not on the maps sizes or map scales. Therefore, I evaluated the time by gradually increasing the number of points on the imagery. In order to compare the time for maps with known map scales and those without known map scales, I randomly selected a Yahoo map (with 57 detected points that is close to the average number of intersections of our tested maps) from my test data set 2. I executed the GeoPPM against the Yahoo map using the known map scale and then I repeated the same process but assumed that the map scale is unknown. In addition, I recursively partition the imagery space into four sub-grids when building the HiGrid structure. I also adjust the parameter n in Eq.(3.5) to examine the performance due to different values. This parameter implies the average number of points in the lowest level. Hence, it controls the depth of HiGrid. Because the number of points on each tested

map is rather small compared with the entire set of points in the imagery, I only consider the partition of the image space but not the map space. In general, my approach can consider the partition for both spaces.

I conflated the Yahoo map with images of different area sizes (i.e., there are different number of image points). The execution time is shown in Table 3.5. There are some immediate conclusions from this table:

- Using map scale information, GeoPPM improves the execution time.
- For the map with known scale, the performance of using road directions is better than just using map scale information.

Table 3.5: The execution time of GeoPPM

	Using map scale only	Using map scale and road directions
402 imagery points	171 seconds	16 seconds
591 imagery points	317 seconds	26 seconds
800 imagery points	540 seconds	42 seconds
1059 imagery points	934 seconds	70 seconds

a) First scenario: map scale is known (there are 57 map points)

	Brute force algorithm	Using road directions	Using HiGrid and road directions
402 imagery points	5 hours 58 minutes	503 seconds	11 seconds
591 imagery points	N/A*	1049 seconds	17 seconds
800 imagery points	N/A*	2449 seconds	26 seconds
1059 imagery points	N/A*	5298 seconds	38 seconds

b) Second scenario: map scale is unknown (there are 57 map points)

	Using HiGrid and road directions
n = 10	38 seconds
n= 28 (i.e., half of map point number)	104 seconds
n= 57 (i.e., map point number)	106 seconds

c) The running time of using different HiGrid parameter *n*

* Due to the poor performance of brute-force algorithm, we did not collect the running time.

- Although my approach did not utilize the HiGrid structure for the map with known map scale, it shows similar performance with respect to the one whose map scale is unknown and utilizes HiGrid. Hence, exploiting the map scale is an effective way to detect matched point pattern.
- Although the road direction information significantly improves the brute-force algorithm, it still may need to examine a large number of potential matching point pairs. This results in long running time for datasets with large number of points.
- GeoPPM utilizing HiGrid outperforms the algorithm that just utilizes road directions.
- Using small HiGrid parameter *n* (i.e., HiGrid with large depth), my approach can efficiently locate the matched point pattern without losing accuracy. This implies that the points are scattered in local areas. In addition, as shown in Table 3.5(c), there is no performance difference for the value 28 (half of map point number) and 57 (map point number), because they result in the same HiGrid depth (according to Eq.(3.5)).

Chapter 4

Related work

Conflation was first proposed and implemented in 1988 by Saalfeld [36], and the initial focus of conflation was to eliminate the spatial inconsistency between two overlapping vector maps in order to improve the spatial accuracy of vector maps. Once the spatial discrepancy is eliminated, it is possible and easier to transfer attributes among datasets to achieve geospatial data fusion. Several important application domains that can benefit from such data fusion are the crisis management, city traffic planning, and military intelligence applications.

Automatic conflation of geospatial datasets is a complex process that may utilize work from a broad range of disciplines that include GIS, cartography, computational geometry, graph theory, image processing, pattern recognition, and statistics. In general, based on the types of geospatial datasets dealt with, the conflation technologies can be categorized into the following three groups:

- Vector to vector data conflation: For example, the integration of two road networks of different accuracy levels.
- Vector to raster data conflation: For example, the integration of road network and imagery or road network and raster maps.

- Raster to raster data conflation: For example, the integration of two images with different resolutions or the integration of raster maps and imagery.

I review some related research on these in turn.

4.1 Vector to Vector Data Conflation

There have been a number of efforts to automatically or semi-automatically accomplish vector to vector conflation [14, 35, 44, 47]. These approaches are different, because of the different methods utilized for locating the counterpart elements from both vector datasets. Most of the proposed approaches focus on handling the integration of two road networks. Saalfeld [35] discussed different strategies to partition space based on the matched entities and concluded that Delaunay triangulation is the most appropriate partition mechanism used for conflation (because the Delanuay triangulation avoids triangles with extremely small angles). Walter and Fritsch [44] proposed a relational matching approach to find matched spatial objects based on the similarity of spatial objects at the geometry level (e.g., node to node matching based on distance) and based on the relations between the elements in a data set. They investigated the "similarity" of spatial objects based on statistical information derived from a random sample of the vector datasets to be integrated. However, their approach requires human intervention to perform an initial affine transformation between datasets. Their approach was utilized in NEXUS [33], an open platform for spatially aware applications, to support the interoperability between datasets. In addition to performing feature matching at

the geometry level, Cobb et al. in [14] proposed an approach to perform feature-matching at the object level. For example, when comparing two road segments, their approach not only matches the road endpoints, but also matches the non-spatial properties such as street names and widths. Yuan and Tao [47] pointed out the necessity to model common conflation steps (such as feature-matching, match-checking, triangulation and rubber-sheeting) as separate components to speed up the development of GIS applications for integrating diverse types of geospatial datasets. They described how to develop conflation components and demonstrated their approach for vector to vector data conflation. Various commercial products, such as MapMerger and Conflex[42], support automatic vector to vector data conflation with limited human intervention to consolidate multiple vector datasets.

4.2 Vector to Raster Data Conflation

Compared with vector to vector conflation, there are fewer research activities on vector to raster data conflation. In particular, in this section, I review some related research about vector to imagery conflation. Due to recent advances in remote sensing technology to capture high resolution imagery, vector to imagery conflation has become one of the central issues in GIS. In fact, various vector datasets (e.g., NAVSTREETS from NAVTEQ) are routinely revised using satellite imagery or aerial photographs. Utilizing automatic conflation techniques, this accurate imagery can be used to update vector datasets. Moreover, the abundant information often

[42] http://www.digitalcorp.com/conflex.htm

contained in diverse vector data can be utilized to annotate objects, such as roads and buildings, in the imagery. Traditionally, these problems of integration of vector and raster data have been in the domain of image processing and GIS.

In Section 4.2.1, I review some techniques that detect counterpart elements in the datasets and apply the traditional conflation algorithm (i.e. establishing the correspondence between the matched entities and transforming other objects accordingly). Section 4.2.2 describes the approaches that utilize active counter models [29] to align each vector road segment to the corresponding road edge in the imagery.

4.2.1 Aligning vector data and imagery using identified features

Hild and Fritsch [24] processed vector data to extract vector polygons and performed image segmentation on imagery to find image polygons. Then, a polygon matching (or shape matching) algorithm is applied on both images and vector polygons to find a set of 2D conjugate points. In order to obtain a successful matching between an image and vector data, the datasets must contain polygonal features like forest, villages, grassland or lakes. This approach will fail when polygonal features cannot be found, such as in high resolution urban areas.

Filin and Doytsher [17] propose a linear conflation algorithm to align vector and imagery. First, all edges (such as road edges and building edges) are detected from the imagery (without using the existing vector data as prior knowledge) and converted to vector format. Then, their approach matches the detected edges with

vector data to identify real road edges. For those detected edges where there is no corresponding edge detected in the imagery, they will be transformed according to the influence regions formed by matched edges. However, extracting features directly from imagery and converting to vector format is a challenging task. There are several algorithms for extracting roads utilizing the characteristics of roads and context of imagery as prior knowledge [25, 34], but none of them give good results in all circumstances [1, 23] and most of them are heavily CPU intensive.

Flavie et al. [18] try to find the junction points of all detected lines in the imagery, then match the junction points of the road vector with the image junctions. Then, the vector lines are moved according to the matched junctions (i.e., no space partition method, such as Delaunay triangulation, is used to build the influence regions of matched junctions). Finally, their system applies the active contour models technique [29] (discussed next) to refine the matched road segments. However, their method suffers from the high computation cost of finding all possible junctions of detected lines on images.

My AMS-conflation for automatic vector to imagery alignment significantly differs from the work mentioned above in terms of my approach to locate matched entities. AMS-conflation is the first that automatically exploits auxiliary structured data (such as image color, image metadata, road vector directions, road intersections and road coordinates provided by vector data) to improve the feature recognition techniques on imagery. In addition, my approach performs a template matching

around each road intersection (i.e., a localized area, instead of the entire image). This will improve both the accuracy and efficiency.

4.2.2 Aligning vector data and imagery using Snakes-related techniques

In the computer vision literature on automatically aligning vector lines with imagery edges, the active contour models (i.e., Snakes [29]) is one of the most prevalent methods to "attach" vector datasets (e.g., road segments) to the corresponding features (e.g., road edges) in the imagery (often with the objective to detect changes of roads or detect real road edges to update pre-existing vector data). Snakes is a parametric curve and it is often modeled as a spline linked by multiple control points. The active contour models evolve their shape by moving their control points towards the image features and maintaining their smoothness at the same time. The evolution is based on the principle of energy (including internal and external energy) minimization. The "internal energy" enforces geometric constraints, such length and smoothness of the Snakes, while the "external energy" pushes the Snakes towards images features. By minimizing internal and external energy simultaneously, image information and geometric properties are fused to accomplish the evolution of the Snakes. The Snakes method requires some seed points as control points to start the evolution and these seed points should be close to the real roads. One option is utilizing pre-existing vector data as the (initial) approximate outline of the roads. However, the Snakes method is not appropriate for aligning roads in highly textured areas such as forests or urban areas, due to the following weaknesses: (1) Noisy edge

pixels might make the Snakes attach to these noisy pixels − this prevents the Snakes from converging on the desired edges; (2) It is a greedy algorithm and demands a lot of calculations when trying all possible (and local) solutions and picking the best one; (3) If the placement of the Snakes is not well initialized, the Snakes will diverge; and (4) Relaxing the internal energy tends to destroy the shape of the Snakes. Furthermore, each vector road segment needs to perform the Snakes evolution to accomplish the alignment.

Compared my AMS-conflation with the Snakes method, my matching mechanism is not based on entire road segments (lines) but on partial road segments around the intersections. For a particular road segment, if the shape of the original vector data is inconsistent with roads in the imagery, my approach may not align them well (although the intersections might be aligned). Considering the Snakes techniques, this type of poorly aligned original vector will also harm the evolution of Snakes. In the worst case, it may cause the Snakes to diverge. However, recently, not only the imagery quality is enhanced (up to 0.3 m/pixel), the quality of vector data is also significantly improved. Consider the conflation of high quality imagery and a high quality vector dataset, such as NAVSTREETS. The road shapes of NAVSTREETS are very similar to the road shapes in the imagery. Hence, the major issue is that there are some local inconsistencies between them. AMS-conflation can capture these local transformations and maintain the road shapes, while the Snakes method may change the road shapes. Hence, AMS-conflation saves computation time by only detecting some (salient) feature points and transforming other points

(and lines) utilizing Delaunay triangulation and rubber-sheeting. In fact, my conflated roads are very close to the road axes on the imagery (based on my experiments) and they can be utilized as good seed points for a more robust Snakes-related algorithm[43] in the future.

4.3 Raster to Raster Data Conflation

With the improvement of mapping technologies to geospatial raster data and with the enhancement of rendering capabilities of personal computers, more and more geospatial datasets are represented as imagery to display the ground truth and as raster maps to display abundant attributes. Using a raster to raster conflation technique, we can generate a super-image that either has the best features or attribute information from each of the individual images or highlights the changes across multiple images. There have been a number of efforts to automatically or semi-automatically accomplish raster to raster conflation. However, most of these studies are dedicated to register imagery with imagery [15, 16, 39]. Many commercial GIS products, such as Able R2V and Intergraph I/RASC provide the functionality of conflating imagery and maps (i.e., raster to raster registration) using different types of transformation methods. However, these products do not provide automatic conflation, so users need to manually pick control points for conflation. To the best of my knowledge, there is no research work dedicated to the automatic conflation of raster maps and orthoimagery.

[43] Many variants of the active contour models are developed to improve the efficiency and accuracy to make it appropriate for different scenarios. This proposed improvement of the active contour models is a different research topic that is beyond the scope of this thesis.

I review some related work on imagery to imagery conflation. In [37], Sato et al. describe how an edge detection process can be used to determine a set of features that can be used to conflate two image data sets. However, their work requires that the coordinates of both image data sets be known in advance. My work does not assume that coordinates for the maps are known in advance, although I do assume that we know the general region. Dare and Dowman [15] proposed a feature-based registration technique (based on multi-feature extraction and matching techniques) to integrate two images. However, their approach requires users to manually select some initial control points to align two images at the first stage. Seedahmed and Martucci [39] proposed an approach, named GIPSC, to extract features from imagery by Moravec feature detector and obtain transformation parameters by investigating the strongest clusters in the parameter space. They assume that there are certain overlapping real spatial objects from both images. Hence, their approach is not appropriate for raster maps and imagery registration. In addition, their approach requires significant CPU time, due to the examination of all potential matching point pairs to solve registration parameters. They also did not address the performance issues. Furthermore, none of the techniques mentioned above proposed a way to evaluate the registration results, while I proposed a novel way to evaluate raster map and imagery conflation.

Although I only discuss the automatic vector-imagery and map-imagery conflation capabilities of AMS-conflation approach in this thesis, it can be easily utilized to support imagery-imagery conflation. Similar to map-imagery conflation, I

can utilize a common road vector data as “glue” to align both images. The idea is utilizing vector-imagery conflation to align the common road vector data with the first and second image separately, and then use these image-aligned intersections to conflate two images. If I cannot find a common road vector dataset for both images, I may utilize different but appropriate road vector datasets to align each image, then utilize GeoPPM technique to find the transformation between the two images.

Chapter 5

Conclusions and Future Work

In this dissertation, I showed that we can achieve automatic and accurate conflation of road vector data, street maps and orthoimagery by exploiting multiple sources of geospatial information. My approach, AMS-conflation, accomplishes the integration of multiple geospatial datasets in a single visualization by eliminating the spatial discrepancies between them. This integrated view of various geospatial datasets then can be utilized to annotate spatial objects in the imagery, update legacy vector data, or highlight the changes across multiple images. Many important application domains can benefit from such geospatial data integration are crisis management, city planning, and military intelligence applications.

Moreover, compared to existing approaches and commercial products, my approach is the first that exploits the metadata, attributes, and inferred knowledge from road vector data, maps, orthoimagery and relevant datasets to achieve the automatic feature detection.

Furthermore, rather than processing each source of information separately in isolation, AMS-conflation processes the sources and utilizes information obtained from one source to help the processing of the other source. In particular, in this

dissertation, I considered two cases, vector-imagery conflation and map-imagery conflation.

Vector and imagery conflation: My approach finds the approximate location of intersections on the images from the inferred knowledge of the corresponding vector data. For each intersection point, the approach performs image processing in a small area around the intersection point to find the corresponding point in the image. In addition to the approximate location of intersections, to locate the intersections on the images, my approach also utilizes other information inferred from vector data such as road-directions, road-widths and road-shapes. More precisely, the approach generates a template inferred from vector data and then matches it against the small area in the image to find the corresponding intersection point on the imagery. The running time for this technique (called LTM) is dramatically lower than traditional image processing techniques due to the localized image processing. Furthermore, the road direction and width information makes detecting roads in the image a much easier problem, thus reducing the running time even more. However, due to the complexity that characterizes natural scenes in imagery, LTM may detect noisy points as control points. To address this issue, I exploit the fact that there is a significant amount of regularity in terms of the relative positions of the nearby control points across data sets. This is due to the fact that my approach is not trying to correct individual errors, but rather to determine some local transformations that allow the system to integrate two separate data sources. Thus, my approach explored a set of filtering techniques and found that they were very effective at eliminating

noisy control points in our data sets. The system can then use these accurate control points to align road vector data with imagery. Based on my experimental results, these automatic conflation techniques demonstrate the utility to automatically align the road vector data to orthoimagery, such that in one of my experiments, 85% of the conflated roads were within 4.5 m from the real road axes compared to 55% for the original roads for partial areas of the county of St. Louis, MO.

Map and imagery conflation: My approach automatically aligns a map and an image by utilizing an auxiliary information source (i.e., road vector data) that is part of neither a map nor an image, but is related to both sources. The approach can take a map of unknown coordinates and automatically align it with an image. First, it aligns road vector data with imagery using my vector-imagery conflation techniques. As a result the conflated intersection points on the road network are aligned with the intersection points on the imagery. It can then use the conflated intersection points as control points on the imagery. For the maps, my approach utilizes image processing techniques to detect road intersections as control points. Furthermore, I developed an efficient technique (called GeoPPM) to compare the distributions of the two point sets by exploiting road direction, map scale, and point density to determine the transformation between the map and imagery. This transformation results in a set of control point pairs to align maps with imagery. The experimental results showed that GeoPPM only misidentified one point pattern from the fifty tested maps. The experimental results also showed that my approach aligned a set of TIGER maps for an area in St. Louis, MO to the corresponding orthoimagery such that in one of my

experiments, 85.2% of the conflated map roads are within 10.8 m from the real road axes. This is a significant improvement considering that simply combining the TIGER maps with the corresponding imagery based on geographic coordinates provided by the sources places 51.7% of the conflated map roads within 10.8 m from the real road axes.

Although I focus on vector-imagery and map-imagery conflation in this dissertation, AMS-conflation can be also utilized to achieve automatic imagery-imagery, map-map and vector-map alignments:

Imagery and imagery conflation: The alignment of multiple images can generate a super-image that either has the best features of each of the individual images or highlights the changes across multiple images. In the former case, the images are available from the same area but with different qualities, and in the latter case multiple images are taken at different time points from the same area. Similar to map-imagery conflation, I can utilize common road vector data as "glue" to conflate to each image separately, and subsequently conflate the two target images.

Map and map conflation: The alignment of multiple maps can generate a super-map that integrates the attribution information from each of the individual maps. I can utilize map intersection detection technique to identify road intersections from each map. Then, I can align the two point sets by using GeoPPM technique. Finally, the two target maps can be aligned based on identified matched point pattern.

Vector and map conflation: The alignment of maps and road vector dataset can also generate a super-map that integrates the attribution information from each of the

individual datasets. I can utilize map intersection detector to identify road intersections from the map. Then, I can align this map point set with road intersections on the vector data by utilizing GeoPPM technique. Subsequently, these control point pairs are used to align vector and maps.

5.1 Future Directions

The directions for future work from this research fall in two main categories. First are directions for further improving or enhancing the existing functionalities of AMS-Conflation. Second are directions for generalizing AMS-conflation to handle other types of geospatial datasets. I discuss each of these in turn.

5.1.1 Improvement of AMS-Conflation to address remaining challenges

There are a number of challenges remaining for vector-imagery conflation and map-imagery conflation, respectively. I describe these challenges as follows.

Vector to imagery conflation: In order to match an intersection template extracted from the vector data to image, I need to identify the corresponding road intersection on the image. So far, I have utilized a learning algorithm based on Bayes classifiers to soft-classify image pixels into on-road and off-road classes. The classifier is trained based on sample imagery tiles and manual identification of on-road and off-road pixels. In order to generalize my technique for imagery of any arbitrary area, I need to develop an approach to learn the road color on the image. In other words, I need to devise an automatic approach to decide whether new training is needed for a given image or whether the current learning is sufficient for the

classification of the new area. I cannot avoid human input in the training process, but I plan to simplify the human task to a simple approval or disapproval of decisions on only a small set of training tiles and only if needed.

Map to imagery conflation: First, I intend to use OCR-related techniques to extract textual information from the maps in order to reduce the impact of these alphanumeric characters. In addition, this pre-extracted textual information (such as road names) can be used to label the detected intersections. Therefore, I can even further prune the search space of possible point pattern matching by using these labeled intersections. Second, I would like to apply my technique to other types of maps besides just street maps. This includes a wide variety of maps that are available from various government agencies, such as property survey maps and maps of oil and natural gas fields. Finally, an interesting direction with respect to integrating maps is to be able to take arbitrary maps with unknown map scale and/or geo-coordinates and determine their map scale and/or location anywhere within a city, state, country, or even the world. Road vector data covering most of the world is available, so the real challenge is enhancing the HiGrid technique for the point pattern matching to make such a search tractable.

5.1.2 Generalization of AMS-Conflation to handle other geospatial data

The AMS-conflation approach can also be generalized to a wide variety of geospatial data sources. The basic idea is to exploit and augment whatever information is available about different geospatial products to automatically determine an accurate

set of control points. Thus, I can apply this approach to conflate a combination of image, map, vector, point and elevation datasets. I focus on two potentially very useful types of alignments for which I believe AMS-conflation can be utilized efficiently.

- Point to map conflation: Consider the case of conflating a database of oil well points with a map of the oil fields where the precise coordinates of the map are unknown. In an example I explored, certain types of oil wells were shown on the map and some of these points were also in the database. Since I need to find a set of common control point pairs, I ran a simple image processing algorithm over the map to identify the location of the oil wells on the map. I then used an algorithm similar to GeoPPM for finding the mapping between the layout (with relative distances) of two sets of points, where there are possibly missing points in both datasets. Once the mapping was found, the database of oil wells could be used to determine the geo-coordinates of some of the points, which could in turn be used to determine the geo-coordinates of the map. Given this information, the database of oil well points can now be superimposed on the map. Now given the geo-coordinates of the maps, other layers such as road vectors could be easily integrated with these maps.
- Elevation data conflation: Consider the case of conflating high-resolution elevation data (e.g. data represented by spot heights) with lower resolution data (e.g., USGS DEM data represented by raster format). This can be

performed using a technique similar to GeoPPM to map two feature point sets. For each of the two elevation data sets, I could identify the highest and/or lowest points. Even though the low resolution data may be missing the highest or lowest points, I can still use GeoPPM because this is similar to the missing intersections in the maps. Using the points identified in each data set, I would then search for the mapping between the highest/lowest points in each data set. Of course, because of the difference in resolution, the current GeoPPM algorithm would need to be extended to support the mapping of points within some threshold. Once the mapping is found, then the two data sets can be conflated. The conflated dataset can show the DEM elevation surface with prominent points (e.g., highest/lowest points) annotated with higher accurate spot height data. Hence, it can be used for subsequent decision-making processes that are based on the elevation information. Similarly, I can utilize the same idea to conflate hydrographic data in vector format with hydrographic in raster format by matching the points with largest/smallest curvatures. Moreover, I could utilize additional elevation information (e.g., USGS DEM) to prune the search space, since the hydrographic data usually have lower elevations.

Bibliography

[1] M.-F. Aculair-Fortier, D. Ziou, C. Armenakis, and S. Wang. "Survey of Work on Road Extraction in Aerial and Satellite Images", *Technical Report*, Universite de Sherbrooke, 2000.

[2] C. Arcelli and G. Sanniti di Baja. "A width-independent fast thinning algorithm", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 7(4): 463-474, 1985.

[3] J. Astola, P. Haavisto, and Y. Neuvo. "Vector Median Filter", *Proceedings of IEEE*, Vol. 78(4): 678-689, 1990.

[4] T. Barclay, J. Gray, and D. Stuz. "Microsoft TerraServer: A Spatial Data Warehouse", In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, TX, ACM Press, May 14-19, 2000, pp. 307-318.

[5] M.D. Berg, M.V. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 1997.

[6] R. Cao and C.L. Tan. "Text/Graphics Separation in Maps", In *Proceedings of the 4th International Workshop on Graphics Recognition Algorithms and Applications*, Ontario, Canada, September 7-8, 2001, pp. 167-177.

[7] D.E. Cardoze and L.J. Schulman. "Pattern Matching for Spatial Point Sets", In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1998, pp. 156-165.

[8] C.-C. Chen, C.A. Knoblock, and C. Shahabi. "Automatically Conflating Road Vector Data with Orthoimagery", *Submitted to GeoInformatica for publication*.

[9] C.-C. Chen, C.A. Knoblock, C. Shahabi, Y.-Y. Chiang, and S. Thakkar. "Automatically and Accurately Conflating Orthoimagery and Street Maps", In *Proceedings of the 12th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS'04)*, Washington, D.C, ACM Press, November 12-13, 2004, pp. 47-56.

[10] C.-C. Chen, C. Shahabi, and C.A. Knoblock. "Utilizing Road Network Data for Automatic Identification of Road Intersections from High Resolution Color Orthoimagery", In *Proceedings of the Second Workshop on Spatio-Temporal Database Management(STDBM'04), colocated with VLDB*, Toronto, Canada, August 30, 2004, 2004, pp. 17-24.

[11] C.-C. Chen, S. Thakkar, C.A. Knoblok, and C. Shahabi. "Automatically Annotating and Integrating Spatial Datasets", In *Proceedings of the 8th International Symposium on Spatial and Temporal Databases (SSTD'03)*, Santorini Island, Greece, July 24-27, 2003, pp. 469-488.

[12] L.P. Chew, M.T. Goodrich, D.P. Huttenlocher, K. Kedem, J.M. Kleinberg, and D. Kravets. "Geometric pattern matching under Euclidean motion", In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, 1993, pp. 151-156.

[13] Y.-Y. Chiang, C.A. Knoblock, C. Shahabi, and C.-C. Chen. "Automatic Road Intersection Extraction for Raster Maps", *Submitted for publication*.

[14] M. Cobb, M.J. Chung, V. Miller, H.I. Foley, F.E. Petry, and K.B. Shaw. "A Rule-Based Approach for the Conflation of Attributed Vector Data", *GeoInformatica*, Vol. 2(1): 7-35, 1998.

[15] P. Dare and I. Dowman. "A new approach to automatic feature based registration of SAR and SPOT images", *International Archives of Photogrammetry and Remote Sensing*, Vol. 33(B2): 125-130, 2000.

[16] L. Ferreres, L.A. Ruiz, F. Artero, and I.S. Kyun. "Using a wavelet based method for high resolution satellite image fusion", In *Proceedings of the IX National Symposium on Pattern Recognition and Image Analysis*, Castell, 2001.

[17] S. Filin and Y. Doytsher. "A Linear Conflation Approach for the Integration of Photogrammetric Information and GIS Data", *International archives of photogrammetry and remote sensing*, Vol. 33(B3/1): 282-288, 2000.

[18] M. Flavie, A. Fortier, D. Ziou, C. Armenakis, and S. Wang. "Automated Updating of Road Information from Aerial Images", In *Proceedings of the American Society Photogrammetry and Remote Sensing Conference*, Amsterdam, Holland, July 16-23, 2000.

[19] D. Forsyth and J. Ponce. *Computer Vision : A Mordern Approach*, Prentice-Hall, 2001.

[20] M.F. Goodchild and G.J. Hunter. "A simple positional accuracy measure for linear features", *International Journal of Geographical Information Science*, Vol. 11(3): 299-306, 1997.

[21] A. Habib, Uebbing, R., Asmamaw, A. "Automatic Extraction of Primitives for Conflation of Raster Maps", *Technical Report*, The Center for Mapping, The Ohio State University, 1999.

[22] W.A. Harvey. "Performance evaluation for road extraction", *The Bulletin de la Societe Francaise de Photogrammetrie et Teledetection*, Vol. 153(1999-1): 79-87, 1999.

[23] C. Heipke, H. Mayer, and C. Wiedemann. "Evaluation of Automatic Road Extraction", *International Archives of Photogrammetry and Remote Sensing*, Vol. 32(3-2W3): 47-56, 1997.

[24] H. Hild and D. Fritsch. "Integration of vector data and satellite imagery for geocoding", *International Archives of Photogrammetry and Remote Sensing*, Vol. 32(4): 246-251, 1998.

[25] S. Hinz, A. Baumgartner, C. Steger, H. Mayer, W. Eckstein, H. Ebner, and B. Radig. "Road Extraction in Rural and Urban Areas", In *Proceedings of the Semantic Modeling for the Acquisition of Topographic Information from Images and Maps*, Munchen, 1999, pp. 7-27.

[26] J.-R. Hwang, J.-H. Oh, and K.-J. Li. "Query Transformation Method by Delaunay Triangulation for Multi-Source Distributed Spatial Database Systems", In *Proceedings of the 9th ACM Symposium on Advances in Geographic Information Systems (ACM-GIS'01)*, Atlanta, Georgia, ACM Press, November 9-10, 2001, pp. 41-46.

[27] S. Irani and P. Raghavan. "Combinatorial and experimental results for randomized point matching algorithms", *Computational Geometry*, Vol. 12(1-2): 17-31, 1999.

[28] M. Jones and J. Rehg. "Statistical color models with application to skin detection", In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Fort Collins, Colorado, IEEE Computer Society, June 23-25, 1999, pp. 1274-1280.

[29] M. Kass, A. Witkin, and D. Terzopoulos. "Snakes: Active contour models", *International Journal of Computer Vision*, Vol. 1(4): 321-331, 1988.

[30] E.M. Knorr, R.T. Ng, and V. Tucakov. "Distance-Based Outliers: Algorithms and Applications", *The VLDB Journal*, Vol. 8(3): 237-253, 2000.

[31] M.T. Musavi, M.V. Shirvaikar, E. Ramanathan, and A.R. Nekovei. "A Vision Based Method to Automate Map Processing", *Pattern Recognition*, Vol. 21(4): 319-326, 1988.

[32] R. Nevatia and K.R. Babu. "Linear Feature Extraction and Description", *Computer Graphics and Image Processing*, Vol. 13: 257-269, 1980.

[33] D. Nicklas, M. Grobmann, S. Thomas, S. Volz, and B. Mitschang. "A Model-Based, Open Architecture for Mobile, Spatially Aware Applications." In *Proceedings of the International Symposium on Spatial and Temporal Databases*, Redondo Beach, CA, 2001.

[34] K. Price. "Road Grid Extraction and Verification", *International archives of photogrammetry and remote sensing*, Vol. 32(3-2W5): 101-106, 1999.

[35] A. Saalfeld. *Conflation: Automated Map Compilation*, Ph.D. Dissertation, Computer Vision Laboratory, Center for Automation Research, University of Maryland, 1993.

[36] A. Saalfeld. "Conflation: Automated Map Compilation", *International Journal of Geographic Information Sciences*, Vol. 2(3): 217-228, 1988.

[37] T. Sato, Y. Sadahiro, and A. Okabe. "A Computational Procedure for Making Seamless Map Sheets", *Technical Report*, Center for Spatial Information Sciences, University of Tokyo, 2001.

[38] T.J. Sebok, L.E. Roemer, and J. Malindzak, G.S. "An Algorithm for Line Intersection Identification", *Pattern Recognition*, Vol. 13(2): 159-166, 1981.

[39] G. Seedahmed and L. Martucci. "Automated image registration using geometrically invariantparameter space clustering (GIPSC)", *International Archives of Photogrammetry, Remote Sensing andSpatial Information Sciences*, Vol. 34(3A): 318-323, 2002.

[40] S. Shekhar, C.T. Lu, and P. Zhang. "A Unified Approach to Spatial Outliers Detection", *GeoInformatica*, Vol. 7(2003): 139-166, 2003.

[41] C. Steger, H. Mayer, and B. Radig. "The Role of Grouping for Road Extraction", *Automatic Extraction of Man-Made Objects from Aerial and Space Images (II)*: 245-256, 1997.

[42] E.L. Usery, M.P. Finn, and M. Starbuck. "Data Integration of Layers and Features for The National Map", In *Proceedings of American Congress on Surveying and Mapping*, Phoenix, AZ, March 31- April 2, 2003.

[43] S. Venkatasubramanian. *Geometric Shape Matching and Drug Design*, Ph.D. Dissertation, Computer Science Deptartment, Stanford University, 1999.

[44] V. Walter and D. Fritsch. "Matching Spatial Data Sets: a Statistical Approach", *International Journal of Geographic Information Sciences*, Vol. 13(5): 445-473, 1999.

[45] M.S. White and P. Griffin. "Piecewise Linear Rubber-Sheet Map Transformation", *The American Cartographer*, Vol. 12(2): 123-131, 1985.

[46] C. Wiedemann, C. Heipke, and H. Mayer. "Empirical Evaluation of Automatically Extracted Road Axes", In *Proceedings of the IEEE Computer Society Workshop on Empirical Evaluation of Computer Vision Algorithms*, Santa Barbara, CA, IEEE Computer Society, June 21-22, 1998.

[47] S. Yuan and C. Tao. "Development of Conflation Components", In *Proceedings of Geoinformatics Conference*, Ann Arbor, Michigan, USA, June 19-21, 1999, pp. 1-13.

Appendix A

In this appendix, I present the pseudo code for vector and imagery conflation.

```
Input:
VectorData[], Imagery   /* VectorData[] stores the road vector segments */
Output:
ConflatedVectorData[]  /* ConflatedVectorData [] stores the conflated  road vector segments */
Descriptions:

        /* Step 1: Analyze vector data to get all road intersections */
        for each road-segment road_i in VectorData[]
            Add the end points of road_i to the array endpoints[]
        end for
        for each point point_i in endpoints[]
            if (there are more than three road segments in VectorData[] intersecting at point_i)
            Add point_i and the directions of intersected roads to the array VectorInt[]
        end for
        /* Step 2: For each intersections on vector data, use LTM to find the corresponding intersection in
           the imagery */
        for each intersection v in VectorInt[]
              iv = LTM(v, Imagery)                    /* LTM is presented in Figure A.2 */
                   if  ( iv != null ) Add the control point pair (v, iv ) to CPs[]
        end for

        /* Step 3: Filter out inaccurate control points (outliers)*/
        filteredCPs[] = Filtering (CPs[])              /* Filtering is presented in Figure A.3 */
        /*Step 4: Apply Delaunay triangulation (see reference [35]) based on filtered CPs to partition
           space*/
        DT[] = DelaunayTriangulation(filteredCPs[])
        /*Step 5: Apply rubber-sheeting (see reference [35]) to align other points*/
        ConflatedVectorData[]= RubberSheeting (VectorData[],DT[])


        return    ConflatedVectorData[]
```

Figure A.1: The pseudo code for vector and imagery conflation

```
Input:
VectInt, Imagery   /* VectInt stores an intersection on road vector data */
Output:
CorrespondingImagePoint=null   /* CorrespondingImagePoint is the identified corresponding intersection
                                  on  the imagery */
Descriptions:

        /* Step 1: Determine a localized area around the intersection on the vector dataset */
        Generate a bounding square Box around the center point VectInt

        /* Step 2: Pre-classifying the imagery pixels */
        Classify each pixel on the Box as on-road or off-road pixel, save the results to roadImg[]

        /* Step 3: Generate a road template from road vector data*/
        Generate a road template roadTemp[] around the point VectInt, based on road-widths and road
        directions

        /* Step 4: Match the road template with classified imagery, using normalized cross correlation*/
        /* The function NormalizedCrossCorrelation computes the best match between roadImg[] and
           roadTemp[] */
        Similarity =  NormalizedCrossCorrelation(roadImg[],roadTemp[] )
        if (Similarity > 0.5)
            CorrespondingImagePoint = the matched image point based on the calculated Similarity

        return    CorrespondingImagePoint
```

Figure A.2: The pseudo code of Localized Template Matching (LTM)

```
Input:
CPs[]                       /* CPs[] stores the unfiltered control point pairs*/
Output:
FilteredCPs[]               /* FilteredCPs [] stores the VMF-filtered control point pairs. That is, the
                               accurate control point pairs */

Descriptions:

        /* Step 1: Convert the control point pairs to control point pair vectors*/
        for each control point pair (v, iv) in CPs[]
            Add the vector (iv-v) to ControlPointPairVector[]
        end for

        /* Step 2: Perform VMF to control point pair vectors*/
        for each control point pair vector cpv in ControlPointPairVector[]
            Find all Control Point Pair Vectors in ControlPointPairVector[] that are close to cpv, store
                    these vectors as NearByControlPointPairVector[]
            VectorMedian= Median(NearByControlPointPairVector [])   /* Find the vector median */
            If cpv is within the k-th closest control point vector to VectorMedian
              Add cpv to FilteredCPs[]
        end for

        return    FilteredCPs[]
```

Figure A.3: The pseudo code of Vector Median Filter (VMF)

Appendix B

In this appendix, I present the pseudo code for map and imagery conflation.

```
Input:
VectorData[], Imagery, Map   /* VectorData[] stores the road vector segments */
Output:
ConflatedMapImagery
Descriptions:

        /* Step 1: Conflate vector data with imagery using the algorithm in Figure A.1 */
        /* S[]  stores the imagery aligned intersections */
        S[] = VectorImageryConflation(VectorData[], Imagery)

        /* Step 2: Find intersections on the map. The detailed algorithm can be found in [13] */
        /* M[] stores the intersections on the map*/
        PreprocessedMap = Thresholding(Map)            /* Remove background pixels from the map */
        PreprocessedMap = RemoveText(PreprocessedMap) /* Remove Text */
        PreprocessedMap = MophOperatior(PreprocessedMap)    /* Reconnect and enhance road
                                                               pixels*/
        PreprocessedMap = Thinning(PreprocessedMap)    /* Make single width road pixels */
        TempInt[] = DetectSalientPoints(PreprocessedMap) /* Find corner points as potential
                                                               intersections*/
        for each point point_i in TempInt[]
            if (there are more than three intersected road segments at point_i)
               Add point_i and the directions of intersected roads to the array M[]

        /* Step 3: Find the matched point pattern from S[] and M[] using GeoPPM described in Figure
        B.2 */
        /* CPs[] stores the matched point pairs */
        CPs[] = GeoPPM (M[],S[])


        /*Step 4: Apply Delaunay triangulation (see reference [35]) based on CPs[]  to partition space*/
        DT[] = DelaunayTriangulation(CPs[])
        /*Step 5: Apply rubber-sheeting (see reference [35]) to align other points*/
        ConflatedMapImagery = RubberSheeting (Map,Imagery,DT[])

        return    ConflatedMapImagery
```

Figure B.1: The pseudo code for map and imagery conflation

```
Input:
        M[], S[]      / * M means Map Points, S means Imagery Points */
Output:
        MatchedPointPattern
Descriptions:
        MatchedPointPattern = null
        UpperThBound = 2 * original setting of δ
        /* Step 1: Check whether map scale is known*/
        if (map –scale is known)
            M' =TransformMapPoints(M) /* transforms map points based on map-scale. See Figure B.3*/
            S'  =TransformImageryPoints(S) /* transforms image points based on image resolution */
          repeat
            Found = false
             /* For all the points in M' and S', generate the corresponding transformation then examine
               whether the transformation meets the threshold α and δ */
           for each map point m in M'
               for each image point s in S' and m has the same directions with s
                    if  ExamineTheTransformation(M', S', m, s, α, δ)   /* See Figure B.3 */
                        Found = true
           δ = δ+ Δ   /* automatic threshold adjustment and search again */
          until (Found or δ ≥ UpperThBound)
          if (Found)
                Pick one matching point to compute the transformation T and then generate
                MatchedPointPattern
        else  /* map –scale is unknown */
          /* Partition the imagery space S into sub-space PS using HiGrid*/
          PS=PartitionSpaceByHiGrid(S)                    /* See Figure B.3 */
          Set PS_i  as the set of partitioned space in the lowest level
          repeat
           repeat
            Found = false
           for each sub-space PS_i
             for each point pairs m_i and m_j in M
                for each s_i and s_j in the same sub-space PS_i and m_i(m_j) has the same directions with s_i(s_j)
                   /* examine whether the transformation formed by m_i ,m_j, s_i ,s_j meets the threshold α
                      and δ */
                    if  ExamineTheTransformation2(M , S, m_i ,m_j, s_i ,s_j ,α, δ)      /* See Figure B.3 */
                        Found = true
           δ = δ+ Δ   /* automatic threshold adjustment and search again */
          until (Found or δ ≥ UpperThBound)
          δ = original setting of δ
          Set PS_i as the sub-space of one level higher than current level
          until (Found or in the highest level of HiGrid)
          if  (Found )
            Pick one matching point to compute the transformation T and then generate
            MatchedPointPattern
      return MatchedPointPattern
```

Figure B.2: The pseudo code for GeoPPM

Function *TransformMapPoints* (input MapPointSet *M,* output TransformedMapPointSet *M'*)
Randomly pick a map point *pnt* from *M* as origin (0,0)
for each point *m* in *M*
Using map-scale, Find the real-world-distance between *m* and *pnt* in north-south and west-east directions, respectively. Then use these distance to determine the coordinate of *m*
Store the coordinate of *m* to *M'*
end for
End Function

Function *TransformImageryPoints* (input ImagePointSet *S,* output TransformedImagePointSet *S'*)
Randomly pick a image point *pnt* from *S* as origin (0,0)
for each point *s* in *S*
Using image-resolution, Find the real-world-distance between *s* and *pnt* in north-south and west-east directions, respectively. Then use these distance to determine the coordinate of *s*
Store the coordinate of *s* to *S'*
end for
End Function

Function Boolean ExamineTheTransformation (input mapPtSet, imagePtSet, m, s, threshold α, δ)
Matched = **false**
if (***m, s***) not in matching point pair record table
Compute the transformation *T* mapping $m \rightarrow s$, if one exists
Compute *T*(*mapPtSet*) /* apply *T* to all map points */
Store the point pair (***m, s***) and the matching results to the matching point pair record
if more than α% of the points in *T*(*mapPtSet*) match points in *S* within the threshold δ *Matched* = **true**
else
if (for the matching record entry with the potential matching point pair (***m, s***) that has more than α% matched points within the threshold δ)
Matched = **true**
end for
return *Matched*
End Function

Function Boolean ExamineTheTransformation2 (input mapPtSet, imagePtSet, m_1, s_1, m_2, s_2, threshold α,δ)
Matched = **false**
if ((m_1, s_1), (m_2, s_2)) not in matching point pair record table
Compute the transformation *T* mapping $m_1 \rightarrow s_1$ and $m_2 \rightarrow s_2$, if one exists
Compute *T*(*mapPtSet*) /* apply *T* to all map points */
Store the point pairs ((m_1, s_1), (m_2, s_2)) and the matching results to the matching point pair record
if more than α% of the points in *T*(*mapPtSet*) match points in *S* within the threshold δ
Matched = **true**
else
if (for the matching record entry with the potential matching point pairs ((m_1, s_1), (m_2, s_2)) that has more than α% matched points within the threshold δ)
Matched = **true**
end for
return *Matched*
End Function

Function *PartitionSpaceByHiGrid* (input ImageSpace *S,* output PartitionedSpace *PS*)
Let *b*= 4 /* Assume recursively the grid into 4 sub-grids */
Let *n*=10 /* Assume there are n points for each cell of the lowest level k */
k= $\left\lfloor \log_b \frac{|S|}{n} \right\rfloor$ /* k is the depth of HiGrid*/
PS = Partition the image space *S* into 4^k equi-seized cells and construct the hierarchical relationships between the space of adjacent levels
End Function

Figure B.3: The subroutines used in GeoPPM