Finding, Extracting, and Integrating Data from Maps

Craig Knoblock

University of Southern California and Geosemble Technologies
Acknowledgements

- **Finding Maps**
  - Joint work with Matthew Michelson, Vipul Verma (IIT Kharagpur), Aman Goel, and Sneha Desai

- **Extracting Data From Maps**
  - Joint work with Yao-Yi Chiang, Jason Chen (Geosemble), Cyrus Shahabi

- **Integrating Maps with Satellite Imagery**
  - Joint work with Jason Chen (Geosemble) and Cyrus Shahabi

- **Research Sponsors:**
  - Air Force Office of Scientific Research
  - Microsoft
Problem: How to Find, Extract, and Align Maps and Imagery

Lat / Long

Lat / Long
Why This is Important

- There is lots of map data available in mapping systems, such as GoogleMaps, Mapquest, etc., but
  - These systems don’t cover the world (no coverage for Iraq in GoogleMaps)
  - There are many types of maps that are not available in these systems (parcel maps, oil field maps, utility maps)
- There is a great deal of information locked up in raster maps
  - Road networks, utility lines, locations of abandoned oil wells, etc.
  - Names of features: roads, buildings, rivers, parks, etc.
Washington DC Transportation Map
USGS Topographic Map
Rand McNally Map
Outline

- Finding Maps
- Aligning Maps with Imagery
  - Extracting intersections
  - Point pattern matching
- Extracting Separate Layers from Maps
- Conclusions
Introduction and Motivation
Introduction and Motivation

Street Maps
Introduction and Motivation

Street Maps

Scanned Documents
Introduction and Motivation

Street Maps

Scanned Documents

Photographs
Introduction and Motivation

Street Maps

Scanned Documents

Photographs

Political, state, area maps
Overall Approach

Module 1: Automatic classification of street maps

Phase 1: Retrieving images from different sources
- Google Images
- Yahoo images

Phase 2: Identifying street maps
- Map Filter

Street maps of the city queried

Module 2: Automatic extraction of intersections

Intersections on the street Maps

Module 3: Automatic georeferencing street maps

Geocoordinates and scales of the street maps
Identifying Street Maps

- Law’s Texture Classification Algorithm
  - (K. Laws. 1980)
- Street maps have unique textures
  - lines, labels, characters
- Generate 75 different attributes (25R, 25G, 25B) to distinguish these textures on the images
Law’s Texture Classification Algorithm
(K. Laws. 1980)

Use different types of masks on the image to identify different textures for example,

- For horizontal lines:
Law’s Texture Classification Algorithm
(K. Laws. 1980)

Use different types of masks on the image to identify different textures for example,

- For horizontal lines:

```
-1 0 -1
2 0 2
-1 0 -1
```

Apply mask

Original Image
Law’s Texture Classification Algorithm
(K. Laws. 1980)

Use different types of masks on the image to identify different textures for example,

- For horizontal lines:

\[
\begin{array}{ccc}
-1 & 0 & -1 \\
2 & 0 & 2 \\
-1 & 0 & -1
\end{array}
\]

Apply mask

Original Image

Resulting Image with horizontal lines
Identifying Street Maps

- Support Vector Machine (Joachims, 1999)
  - Machine learning classification
  - Given training examples labeled either "yes" or "no", SVM creates a hyperplane to separate data into two classes
  - The dimension of the hyperplane is the number of attributes
Identifying Street Maps

- We train on one set of images and test on a separate set of images

**Training:**
- We provided 1150 different positive and negative examples of images
- 75 attributes per image

**Classification:**
- Using the trained SVM model to classify test images
Initial Results
[Desai et al, 2005 ACMGIR]

- Worked well for identifying specific types of maps with 93% recall and 100% precision, but...
- When we trained on a wider range of maps the recall went up and the precision went down
- So we are moving to a new classification approach...
Content-Based Information Retrieval

Get a query Image

Find most similar 9 images from database

No of maps $\geq 5$

Non Map $\rightarrow$ YES $\rightarrow$ Map

Non Map $\leftarrow$ NO
Outline

- Finding Maps
- Aligning Maps with Imagery
  - Extracting intersections
  - Point pattern matching
- Extracting Separate Layers from Maps
- Conclusions
Aligning Maps with Imagery

Utilize vector data as “glue” to automatically conflate imagery with maps
Aligning Maps with Imagery

- Utilize vector data as “glue” to automatically conflate imagery with maps

Geo-referenced Satellite Imagery → Vector-Imagery Conflation → Points On the Satellite Imagery/Vector Data

Map with Unknown Coordinates
Aligning Maps with Imagery

Utilize vector data as “glue” to automatically conflate imagery with maps

Map with Unknown Coordinates

Detect Intersection Points On the Map

Geo-referenced Satellite Imagery

Vector-Imagery Conflation

Vector Data TIGERLine

Points On the Satellite Imagery/ Vector Data
Aligning Maps with Imagery

- Utilize vector data as “glue” to automatically conflate imagery with maps

Map with Unknown Coordinates → Detect Intersection Points On the Map → Point Pattern Matching & Map-Imagery Conflation

Geo-referenced Satellite Imagery → Vector-Imagery Conflation → Points On the Satellite Imagery/ Vector Data

Vector Data TIGERLine
Aligning Maps with Imagery

- Utilize vector data as “glue” to automatically conflate imagery with maps

1. Map with Unknown Coordinates
2. Detect Intersection Points On the Map
3. Geo-referenced Satellite Imagery
4. Vector-Imagery Conflation
5. Points On the Satellite Imagery/Vector Data
6. Vector Data TIGERLine
7. Point Pattern Matching & Map-Imagery Conflation
Outline

- Finding Maps
- Aligning Maps with Imagery
  - Extracting intersections
  - Point pattern matching
- Extracting Separate Layers from Maps
- Conclusions
Module 1: Automatic Segmentation

Module 2: Pre-Processing: Extract and Rebuild Road Layer

- Double-Line Map Detection
- Parallel-Pattern Tracing
- Text/Graphic Separation
- Morphological Operations

Module 3: Determine Road Intersections and Extract Connectivity with Road Orientation

- Detect Road Intersection Candidates
- Extract Connectivity of Road Intersection Candidates

Road Intersection Candidates with Connectivity > 2

Extract Orientation

Road Intersection Points with Connectivity and Orientation
Module 1: Automatic Segmentation

- Raster Maps

Module 2: Pre-Processing: Extract and Rebuild Road Layer

- Double-Line Map Detection
- Parallel-Pattern Tracing
- Text/Graphic Separation
- Single-line map
- Morphological

Remove Noise and Rebuild Road Layer

Module 3: Determine Road Intersections and Extract Connectivity with Road Orientation

- Detect Road Intersection Candidates
- Extract Connectivity of Road Intersection Candidates
- Road Intersection Candidates with Connectivity > 2
- Extract Orientation

Road Intersection Points with Connectivity and Orientation
Module 1: Automatic Segmentation

Module 2: Pre-Processing: Extract and Rebuild Road Layer

Module 3: Determine Road Intersections and Extract Connectivity with Road Orientation

Road Intersection Points with Connectivity and Orientation
Remove Background

- Use Triangle method (Zack, 1977) to locate luminosity clusters in the histogram
- Remove the dominate cluster
Remove Background

- Use Triangle method (Zack, 1977) to locate luminosity clusters in the histogram
- Remove the dominate cluster
Remove Background

- Use Triangle method (Zack, 1977) to locate luminosity clusters in the histogram
- Remove the dominate cluster
Remove Background

- Use Triangle method (Zack, 1977) to locate luminosity clusters in the histogram
- Remove the dominate cluster

Background color should have dominate number of pixels
Remove Background

- Use Triangle method (Zack, 1977) to locate luminosity clusters in the histogram
- Remove the dominate cluster
Remove Noise & Rebuild Road Layer

Before we extract the intersections, we separate the road layer

Double-line road layer

Single-line road layer
Remove Noise & Rebuild Road Layer

Before we extract the intersections, we separate the road layer

- Double-line road layer
- Single-line road layer
Before we extract the intersections, we separate the road layer.

- Double-line road layer
- Single-line road layer
Remove Noise & Rebuild Road Layer

- Double-line road layers provide more information to separate the road layer from other linear structures.

- We utilize Parallel Pattern Tracing to find parallel road lines.
Parallel Pattern Tracing

Zoom in to pixel level:
- 8 directions connect to one pixel
- 4 possible straight lines

If a pixel is on a double line layer with road width=3 pixels, we should be able to find:
- At least 1 pixel on the original road line
- At least 1 corresponding pixel on the other road line

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Parallel Pattern Tracing

Zoom in to pixel level:

- 8 directions connect to one pixel
- 4 possible straight lines

If a pixel is on a double line layer with road width=3 pixels, we should be able to find:

- At least 1 pixel on the original road line
- At least 1 corresponding pixel on the other road line
Parallel Pattern Tracing

- Zoom in to pixel level:
  - 8 directions connect to one pixel
  - 4 possible straight lines

- If a pixel is on a double line layer with road width=3 pixels, we should be able to find:
  - At least 1 pixel on the original road line
  - At least 1 corresponding pixel on the other road line
Parallel Pattern Tracing

Zoom in to pixel level:
- 8 directions connect to one pixel
- 4 possible straight lines

If a pixel is on a double line layer with road width=3 pixels, we should be able to find:
- At least 1 pixel on the original road line
- At least 1 corresponding pixel on the other road line

Construct the first line
Parallel Pattern Tracing

Zoom in to pixel level:

- 8 directions connect to one pixel
- 4 possible straight lines

If a pixel is on a double line layer with *road width=3 pixels*, we should be able to find:

- At least 1 pixel on the original road line
- At least 1 corresponding pixel on the other road line
Parallel Pattern Tracing
Detect the type of road layer, the road width
Remove linear structures other than parallel roads
Remove Noise & Rebuild Road Layer

- Text/Graphics Separation (Cao et al. 2001)
  - Separate linear structures from other objects
Remove Noise & Rebuild Road Layer

- **Text/Graphics Separation** (Cao et al. 2001)
  - Separate linear structures from other objects

Find small connected objects - character
Remove Noise & Rebuild Road Layer

- **Text/Graphics Separation** (Cao et al. 2001)
  - Separate linear structures from other objects

- Find small connected objects - character
- Group small connected objects - string
Remove Noise & Rebuild Road Layer

- **Text/Graphics Separation** (Cao et al. 2001)
  - Separate linear structures from other objects

Find small connected objects - character

Group small connected objects - string

Remove small connected object groups
Remove Noise & Rebuild Road Layer

- **Text/Graphics Separation** (Cao et al. 2001)
  - Separate linear structures from other objects

Find small connected objects - character

Group small connected objects - string

Remove small connected object groups

After the removal of objects touching road lines, the road network is broken
Rebuild Road Layer

- General Dilation operator
  - Reconnect the broken road layer

Generalized Dilation
Rebuild Road Layer

- General Dilation operator
  - Reconnect the broken road layer

Generalized Dilation

For every foreground pixel, fill up its eight neighboring pixels.

1st iteration
Rebuild Road Layer

- General Dilation operator
- Reconnect the broken road layer

Generalized Dilation

For every foreground pixel, fill up its eight neighboring pixels.

After 2 iterations

2nd iteration
Rebuild Road Layer

- General Erosion operator
  - Thin road lines and maintain the original orientation

Generalized Erosion
Rebuild Road Layer

- General Erosion operator
  - Thin road lines and maintain the original orientation

Generalized Erosion

For every foreground pixel, erase it if any neighboring pixel is white.

1st iteration
Rebuild Road Layer

- General Erosion operator
  - Thin road lines and maintain the original orientation

Generalized Erosion

For every foreground pixel, erase it if any neighboring pixel is white.
Rebuild Road Layer

- **Thinning operator**
  - Produce one pixel width road lines

Thin each road line until they are all one pixel width.
Rebuild Road Layer

- Thinning operator
  - Produce one pixel width road lines

Thin each road line until they are all one pixel width.
Identify Road Intersections and Extract Road Information

Corner detector (OpenCV)
  - Find intersection candidates

Compute the *connectivity* and *orientation* to determine correct intersections
Identify Road Intersections and Extract Road Information

Corner detector (OpenCV)
- Find intersection candidates
Compute the **connectivity** and **orientation** to determine correct intersections
Identify Road Intersections and Extract Road Information

Corner detector (OpenCV)
- Find intersection candidates

Compute the **connectivity** and **orientation** to determine correct intersections
Identify Road Intersections and Extract Road Information

Corner detector (OpenCV)
- Find intersection candidates

Compute the connectivity and orientation to determine correct intersections

Connectivity<3, discard
Identify Road Intersections and Extract Road Information

Corner detector (OpenCV)

- Find intersection candidates

Compute the **connectivity** and **orientation** to determine correct intersections

Connectivity < 3, discard

Connectivity ≥ 3, compute road orientations
Experimental Results – Precision and Recall

Total 56 raster maps from 6 different sources with various resolution.

- Precision (%)
- Recall (%)

Bar chart showing precision and recall for different map sources:
- ESRI Map
- MapQuest Map
- TIGER/Line Map
- USGS Topographic Map
- Yahoo Map
- Thomas Brother Map
Experimental Results – Positional Accuracy

Total 56 raster maps from 6 different sources with various resolution.
Experimental Results - Performance

- **Computation time:**
  - Platform/Machine: Windows 2000 Server, Intel Xeon 1.8 GHZ Dual-Processor with 1 GB memory
  - 800x600 topographic map with resolution 2m/pixel: less than 1 minutes
  - Other simpler maps: less than 20 seconds
Outline

- Finding Maps
- Aligning Maps with Imagery
  - Extracting intersections
  - Point pattern matching
- Extracting Separate Layers from Maps
- Conclusions
Point Pattern Matching: Overview

Find the mapping between these points

Why? To generate a set of control point pairs

80 points

Example: \((x, y) = (83, 22)\)

400 points

Example: \((\text{lon, lat}) = (-118.407088, 33.92993)\)
Find the mapping between these points

Why? To generate a set of control point pairs

How to solve the point sets matching problem:

- A geometric point sets matching problem
- Find the transformation $T$ between the layout (with relative distances) of the two point sets
Point Pattern Matching: Finding the Transformation [Chen et al., 2004, ACMGIS]
Point Pattern Matching: Finding the Transformation [Chen et al., 2004, ACMGIS]
Point Pattern Matching: Finding the Transformation [Chen et al., 2004, ACMGIS]
Point Pattern Matching: Finding the Transformation [Chen et al., 2004, ACMGIS]

Transformation = Scaling + Translation
- Transforms most points on map to points on imagery
- Find matching point pairs to solve this transformation
Point Pattern Matching: A Brute-Force Algorithm

- Iterate all point pair in M, and for each chosen point pair in M examining all point pairs in S
- Time-consuming: $O(m^3 n^2 \log n)$
- Can we improve it by randomization? Not always!
  - Noisy points on maps
  - Some missing points on imagery
Point Pattern Matching: A Brute-Force Algorithm

- Iterate all point pair in M, and for each chosen point pair in M examining all point pairs in S

  - Time-consuming: $O(m^3 n^2 \log n)$
  - Can we improve it by randomization? Not always!
    - Noisy points on maps
    - Some missing points on imagery
Point Pattern Matching: A Brute-Force Algorithm

- Iterate all point pair in M, and for each chosen point pair in M examining all point pairs in S

  - Time-consuming: $O(m^3 n^2 \log n)$

  - Can we improve it by randomization? Not always!
    - Noisy points on maps
    - Some missing points on imagery
Point Pattern Matching: A Brute-Force Algorithm

- Iterate all point pair in M, and for each chosen point pair in M examining all point pairs in S
  - Time-consuming: $O(m^3 n^2 \log n)$
  - Can we improve it by randomization? Not always!
    - Noisy points on maps
    - Some missing points on imagery

Apply $T$

Check all pairs on S

m Points on Map M

n Points on Image S
Geospatial Point Pattern Matching (GeoPPM):
Exploit Geometric Info. Associated with Each Intersection

- Intersection degree: the number of intersected roads
- Directions of Intersected road segments

Degree: 3; Directions: 0, 90, 270
Geospatial Point Pattern Matching (GeoPPM):
Exploit Geometric Info. Associated with Each Intersection

- Intersection degree: the number of intersected roads
- Directions of Intersected road segments

Degree: 3; Directions: 0, 90, 270
We need to consider translation only

\[ O(m^3 n^2 \log n) \rightarrow O(m^2 n \log n) \]
Exploiting Point Density and Localized Distribution of Points

Assumption: we focus on medium to high resolution maps

- We are conflating maps with high resolution imagery!

Geospatial Point Pattern Matching (GeoPPM): For Map with Unknown Map Scale

Level 1: 1.2 m/pixel  Level 2: 4.25 m/pixel  Level 3: 14.08 m/pixel  Level 4: 35 m/pixel

Coarse level map: map with smaller map-scale (low resolution)
Geospatial Point Pattern Matching (GeoPPM): Exploiting Point Density

55 points

1059 points
Geospatial Point Pattern Matching (GeoPPM):
Exploit Localized Distribution of Points

The points are in a cluster!
Geospatial Point Pattern Matching (GeoPPM):
Exploit Localized Distribution of Points

The points are in a cluster!
Geospatial Point Pattern Matching (GeoPPM): Exploit Localized Distribution of Points

The points are in a cluster!

57 detected map points

1059 points
Geospatial Point Pattern Matching (GeoPPM): Exploit Localized Distribution of Points
## Geospatial Point Pattern Matching (GeoPPM):
Exploit Localized Distribution of Points

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Geospatial Point Pattern Matching (GeoPPM): Exploit Localized Distribution of Points
Geospatial Point Pattern Matching (GeoPPM): Exploit Localized Distribution of Points
Geospatial Point Pattern Matching (GeoPPM): Exploit Localized Distribution of Points
Geospatial Point Pattern Matching (GeoPPM): Exploit Localized Distribution of Points
Geospatial Point Pattern Matching (GeoPPM): Exploit Localized Distribution of Points
Geospatial Point Pattern Matching (GeoPPM): Current Implementation

- Utilizing these exploited information simultaneously to prune search space
  - Road directions
  - Map scale
  - Point density
  - Localized distribution of points

Map scale is known?
- yes
- Transforming
  - Image points
  - Map points

- no
- Sub-dividing image space into small sub-space

Utilizing
- road directions
- point density
  to filter potential matching points

For each sub-space,
Utilizing
- road directions
  to filter potential matching point pairs
Aligning Maps and Imagery

- Using matched point pattern to align maps with imagery by Delaunary triangulation and rubber-sheeting [Saalfeld’88]
  - Space partition to build influence regions: Delaunary triangulation
  - Warping maps’ pixels within each triangle to the corresponding pixels on imagery: based on Delaunary triangles and rubber-sheeting
Results
Results
Results
Results
Conflated DC Transportation Map
## Evaluation:
The performance of GeoPPM in Precision/Recall

<table>
<thead>
<tr>
<th></th>
<th>ESRI map</th>
<th>MapQuest map</th>
<th>Yahoo map</th>
<th>TIGER map</th>
<th>Topographic map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>96.0%</td>
<td>95.2%</td>
<td>94.0%</td>
<td>84.2%</td>
<td>93.9%</td>
</tr>
<tr>
<td>Recall</td>
<td>80.2%</td>
<td>84.8%</td>
<td>88.3%</td>
<td>75.6%</td>
<td>80.94%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test data set 1 (El Segundo, CA)</th>
<th>Test data set 2 (St. Louis, MO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>Precision</td>
</tr>
<tr>
<td>91.9%</td>
<td>93.4%</td>
</tr>
<tr>
<td>Recall</td>
<td>Recall</td>
</tr>
<tr>
<td>84.6%</td>
<td>77.4%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Res ≤ 2m/pixel (38%)</td>
<td>87.4%</td>
<td>78.2%</td>
</tr>
<tr>
<td>2 &lt; Res ≤ 4 (18%)</td>
<td>92.9%</td>
<td>84.0%</td>
</tr>
<tr>
<td>4 &lt; Res &lt; 7 (33%)</td>
<td>96.4%</td>
<td>88.6%</td>
</tr>
<tr>
<td>Res &gt; 7 (13%)</td>
<td>91.6%</td>
<td>77.1%</td>
</tr>
</tbody>
</table>
Evaluation: The performance of GeoPPM

One of our 50 tested maps where the intersection point set is not accurately aligned with the corresponding point pattern on the image.
## Evaluation:

### The running time of GeoPPM

**Platform:** Windows 2000; CPU Xeon 1.8GHz with 1GMB memory

Test on a Yahoo map with 57 points with varying number of image points

<table>
<thead>
<tr>
<th>Imagery Points</th>
<th>Brute force algorithm</th>
<th>Using map scale only</th>
<th>Using map scale and road directions</th>
<th>Using road directions</th>
<th>Using HiGrid and road directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>402 points</td>
<td>5 hours 58 minutes</td>
<td>171 seconds</td>
<td>16 seconds</td>
<td>503 seconds</td>
<td>11 seconds</td>
</tr>
<tr>
<td>591 points</td>
<td>N/A</td>
<td>317 seconds</td>
<td>26 seconds</td>
<td>1049 seconds</td>
<td>17 seconds</td>
</tr>
<tr>
<td>800 points</td>
<td>N/A</td>
<td>540 seconds</td>
<td>42 seconds</td>
<td>2449 seconds</td>
<td>26 seconds</td>
</tr>
</tbody>
</table>
### Evaluation:
The running time of GeoPPM

- **Platform:** Windows 2000; CPU Xeon 1.8GHz with 1GMB memory
- **Test on a Yahoo map with 57 points with varying number of image points**

<table>
<thead>
<tr>
<th></th>
<th>Brute force algorithm</th>
<th>Using map scale only</th>
<th>Using map scale and road directions</th>
<th>Using road directions</th>
<th>Using HiGrid and road directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>402 imagery points</td>
<td>5 hours 58 minutes</td>
<td>171 seconds</td>
<td>16 seconds</td>
<td>503 seconds</td>
<td>11 seconds</td>
</tr>
<tr>
<td>591 imagery points</td>
<td>N/A</td>
<td>317 seconds</td>
<td>26 seconds</td>
<td>1049 seconds</td>
<td>17 seconds</td>
</tr>
<tr>
<td>800 imagery points</td>
<td>N/A</td>
<td>540 seconds</td>
<td>42 seconds</td>
<td>2449 seconds</td>
<td>26 seconds</td>
</tr>
<tr>
<td>1059 imagery points</td>
<td>N/A</td>
<td>934 seconds</td>
<td>70 seconds</td>
<td>5298 seconds</td>
<td>38 seconds</td>
</tr>
</tbody>
</table>
Evaluation:
The running time of GeoPPM

Platform: Windows 2000; CPU Xeon 1.8GHz with 1GMB memory

Test on a Yahoo map with 57 points with varying number of image points

<table>
<thead>
<tr>
<th>Imagery Points</th>
<th>Brute Force Algorithm</th>
<th>Using Map Scale Only</th>
<th>Using Map Scale and Road Directions</th>
<th>Using Road Directions</th>
<th>Using HiGrid and Road Directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>402</td>
<td>5 hours 58 minutes</td>
<td>171 seconds</td>
<td>16 seconds</td>
<td>503 seconds</td>
<td>11 seconds</td>
</tr>
<tr>
<td>591</td>
<td>N/A</td>
<td>317 seconds</td>
<td>26 seconds</td>
<td>1049 seconds</td>
<td>17 seconds</td>
</tr>
<tr>
<td>800</td>
<td>N/A</td>
<td>540 seconds</td>
<td>42 seconds</td>
<td>2449 seconds</td>
<td>26 seconds</td>
</tr>
<tr>
<td>1059</td>
<td>N/A</td>
<td>934 seconds</td>
<td>70 seconds</td>
<td>5298 seconds</td>
<td>38 seconds</td>
</tr>
</tbody>
</table>
Outline

- Finding Maps
- Aligning Maps with Imagery
  - Extracting intersections
  - Point pattern matching
- Extracting Separate Layers from Maps
- Conclusions
Extracting the Layers of a Map

- Separate the road and text layers on a map
Previous Work

• Bixler 00’, Fletcher 88’, and Velazquez 03’ assume that the line and character pixels are not overlapping
• Li et al. work in local areas to separate the characters from lines
• Cao et al. use the different length of line segments to separate characters from line arts
• We do not use geometric properties as the previous work
Approach

- We use texture classification approach to classify pixels
- Features:
  - Discrete Cosine Transformation (DCT) coefficients
- Classifier:
  - Support vector machine
Discrete Cosine Transformation

- The DCT transformation gives us the strength of each component in the frequency domain for an image.

- The DCT coefficients represent the variation around each pixel.
Remove Background

- Pixels with low variation around them are background pixels
- A color C in a map can only represent either background or foreground
- For a color C to be a background color: more than 50% pixels of color C are background pixels
Classify Line and Character pixels

- SVM Training
  - One MapQuest map for character sample; one Google map and one Viamichiline map for line samples
  - The testing maps are disjoint from the training samples
- Example of classification result
Results

<table>
<thead>
<tr>
<th>Map Source</th>
<th>Precision/Recall of Classification</th>
<th>Line Pixels</th>
<th>Character Pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ours</td>
<td>Cao’s</td>
</tr>
<tr>
<td>A9</td>
<td></td>
<td>99/91%</td>
<td>95/91%</td>
</tr>
<tr>
<td>MSN</td>
<td></td>
<td>99/79%</td>
<td>91/87%</td>
</tr>
<tr>
<td>Google</td>
<td></td>
<td>99/99%</td>
<td>95/99%</td>
</tr>
<tr>
<td>Yahoo</td>
<td></td>
<td>95/91%</td>
<td>70/96%</td>
</tr>
<tr>
<td>Mapquest</td>
<td></td>
<td>99/78%</td>
<td>88/73%</td>
</tr>
<tr>
<td>Map24</td>
<td></td>
<td>95/74%</td>
<td>97/70%</td>
</tr>
<tr>
<td>ViaMichelin</td>
<td></td>
<td>83/34%</td>
<td>44/57%</td>
</tr>
<tr>
<td>Multimap</td>
<td></td>
<td>89/82%</td>
<td>98/64%</td>
</tr>
<tr>
<td>TIGER/Line</td>
<td></td>
<td>99/94%</td>
<td>97/89%</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>98/85%</td>
<td>85/82%</td>
</tr>
</tbody>
</table>

Computation time:
- For a 400x400 Google Map:
  - 2 seconds to remove background
  - 4 seconds to classify line and character pixels
- No threshold needed
Outline

- Finding Maps
- Aligning Maps with Imagery
  - Extracting intersections
  - Point pattern matching
- Extracting Separate Layers from Maps
- Conclusions
Conclusions

- We have developed the capabilities to:
  - Automatically find maps for a given region
  - Extract the intersection features from a given map
  - Determine the exact location of the map
  - Integrate the amp with satellite imagery
- Now building a system that given a location will generate a library of maps aligned with satellite imagery
Future Work

Remaining challenges include:

- Building a knowledge base of intersections to support the processing of maps anywhere in the world
- Supporting poor quality scanned maps and compressed jpg maps
- Scaling the matching algorithms to support larger areas
- Building vector layers directly from raster maps
- Developing OCR techniques to recognize the text
Publications

Available online: http://www.isi.edu/~knoblock

Finding maps

Extracting intersections

Aligning maps with imagery

Extracting Road Layers