Kartta Labs: Unrendering Historical Maps

Sasan Tavakkol*
tavakkol@google.com
Google Research
New York, NY

Yao-Yi Chiang*
yaoyic@google.com
Google Research
New York, NY

Tim Waters*
tim@watersgeospatial.co.uk
Waters Geospatial
Leeds, W. Yorks, UK

Feng Han*
bladehan@google.com
Google Research
New York, NY

Kisalaya Prasad*
kisalaya@google.com
Google
New York, NY

Raimondas Kiveris*
rkiveris@google.com
Google Research
New York, NY

ABSTRACT

This paper introduces the Kartta Labs project, an ongoing open-source and open-data project aiming at organizing the world's historical maps and making them universally accessible and useful. Kartta Labs' framework is designed as a composition of multiple modules. Each module has a crowdsourcing implementation and an artificial intelligence based implementation. The framework takes images of historical maps, registers them in space and time, generates a vector version of the map content, and allows the users to query for the vector content and recreate the historical maps in various cartographic styles. We refer to this process as unrendering. The resulting machine readable map data can support a variety of scientific studies and applications that require long-term, detailed geographic information in the past, while opening up opportunities in other areas such as entertainment. This paper also presents the preliminary results from one automated module to geolocalize a given historical map.

CCS CONCEPTS

• **Information systems** → **Content analysis and feature selection**; *Open source software*; *Spatial-temporal systems*; *Collaborative search*.

KEYWORDS

historical maps, digital map processing, Kartta Labs, open source tools, open data

ACM Reference Format:

Sasan Tavakkol, Yao-Yi Chiang, Tim Waters, Feng Han, Kisalaya Prasad, and Raimondas Kiveris. 2019. Kartta Labs: Unrendering Historical Maps. In *3rd ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery (GeoAI'19)*, November 5, 2019, Chicago, IL, USA. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3356471.3365236>

*All authors contributed equally to this research.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
GeoAI'19, November 5, 2019, Chicago, IL, USA
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6957-2/19/11.
<https://doi.org/10.1145/3356471.3365236>

1 INTRODUCTION

Kartta Labs is an open-source¹ and open-data project with crowdsourcing and artificial intelligence modules. These modules are defined to collect and process wide varieties and numbers of historical maps to collectively build and serve an extensive database of geographic features with a time dimension (we envision OpenStreetMap², OSM, with a time slider). Kartta Labs' mission is to organize the world's historical maps and make them universally accessible and useful.

While there exist several efforts to collect historical maps in online digital archives [2], Kartta Labs goes beyond data collection to also 1) register the maps in space and time, 2) extract geographic features from the maps, and 3) provide the mechanisms for querying the extracted geographic features from a map server. There is a large community of researchers and engineers working on developing these processes with various crowdsourcing or automated tools [3, 5]. For example, the New York Public Library (NYPL) uses an open-source software called MapWarper³ to crowdsource the georectification of historical map scans, and another open-source software named Building Inspector [1], to vectorize them. While we leverage these open-source tools, our goal is to increase the level of automation and scale. Similar automated tools exist, but they typically only handle specific types of maps or map processing steps, work in silos, and do not scale to process large numbers of heterogeneous maps [5]. In contrast, Kartta Labs defines and integrates independent map processing modules, each with both crowdsourcing and artificial intelligence components, to fully process heterogeneous historical maps. Kartta Labs' pipeline receives a raster map⁴ and produces a semantically labeled vector map.

Kartta Labs' final product is a map server containing historical map contents in a vector format, registered in time and space, and rendered in tiles. Kartta Labs will produce a stack of open-source tools to enable both crowdsourcing and automated processing of historical maps, allowing users to request historical geographic datasets for a particular location (e.g., in latitude and longitude or by names) and a period (e.g., 1942 - 1945).

¹<http://github.com/kartta-labs>

²<http://openstreetmap.org/>

³<http://mapwarper.net/>

⁴A map in a raster data format, which can be a scanned image, a screenshot of a map, or other types of map images

2 FRAMEWORK

The Kartta Labs' framework for the processing of historical maps includes six major modules, each defined by its data input and output. The implementation may rely on crowdsourcing, machine learning, etc. The six modules are:

Georectification: This module finds the correspondence between a raster map and a reference geographic dataset (e.g., OSM maps). Then, it aligns the map to the reference dataset to obtain the precise location of the map content on Earth. This module needs to find control points (points on the raster map with a known latitude and longitude) and then apply a transformation function to warp the raster map such that if the map overlays on the reference dataset, each control point is on top of its corresponding geographic location. The input is a raster map and a georeferenced dataset, and the output is an array of pairs of control points in image coordinates and geographic coordinates.

Semantic Segmentation: This module assigns a semantic label to each pixel of the raster map. For example, each pixel can have labels such as "road", "river", or "building". The input is a raster map, and the output is an image of the same image dimensions as the input map with a predefined number of channels. Each channel represents the probability of a pixel belonging to a class of geographic features. The number of channels represents the number of geographic feature classes (plus one class for "others"). For each pixel, the sum of its probability from all channels is 1.

Vectorization: This module gives a vector representation for each geographic feature on the raster map, expressing them as either points, lines, or polygons. The input is a semantically segmented image of a raster map, and the output is the same map in a vector format (e.g., in GeoJSON⁵).

Labeling: This module recognizes and assigns toponyms to each vector feature on the raster map. For example, if the map contains the polygon boundary and text label of the Metropolitan Museum of Art, this module will recognize and assign the text label to the polygon feature. The input is a vector map, and the output is the same vector map augmented with text labels (toponyms).

Temporal Localization: This module associates a raster map and its extracted map content with a snapshot in time. This information might come explicitly as in a metadata file, might be available in a raster format on the image (e.g., on map collar), or might be missing and must be implicitly inferred. The input is a map and the output is a year range.

Synthetic Map Generation: For either crowdsourcing or automated techniques, Kartta Labs needs data to design, train, and test the framework. Real data are usually manually georeferenced and/or digitized from scanned historical maps and hence in small amounts. These real data are useful in training machine learning components and validating the results from both crowdsourcing and automated components. For text recognition and detection, Chiang et al. at the University of Southern California⁶ and Weinman et al. at Grinnell College⁷ have released a small number of ground truth data (less than 100 maps) including the original map

scans, text labels in machine-readable strings, and the text bounding boxes. GB1900⁸ has released about 2 million crowdsourced text labels from historical Ordnance Survey maps, but the dataset only contains point locations of these labels (instead of bounding boxes). NYPL⁹ has a large number of georeferenced and georectified maps. USGS hosts a large collection of georeferenced historical topographic maps covering the entire United States¹⁰. OldMapsOnline¹¹ provides a wide variety of georeferenced historical maps covering many places around the globe. These maps can be used to generate training data for geographic feature detection models for historical maps [4].

Synthetic data often prove to be a viable option for training and validation of models in lack of real data (e.g., using transfer learning [2]). Although there is no guarantee that a model trained on synthetic data will handle real data with similar performance, synthetic data can help to rapidly explore the solution space, test the feasibility of different algorithms and implementations, and choose the most promising one. This module generates metadata-rich raster maps from contemporary vector data. The input is contemporary geographic datasets, and the output is metadata-rich synthetic and semantic map pairs. Figure 1 shows a preliminary result of a raster map paired with its color coded semantic map. The raster and semantic map pairs can serve as training and validation data for the modules described in this section. For example, we can directly generate such synthetic map pairs in various map styles (or use style transfer to generate them in historical map styles [6]) to train and test the Georectification module.

Once a raster map goes through the Kartta Labs modules in a pipeline, the final output is stored in a spatiotemporal database and can be used for recreating the historical maps using a map server. It is not strictly necessary to perform these modules in the given order. For example, the Vectorization module can process the raster map before the Georectification module or vice versa. Finally, Kartta Labs will also host a map drawing tool to let users draw maps for areas or time periods without any available historical maps. These drawings may rely on users' local information, old documents and gazetteers, etc.

Figure 1: Raster map (left) paired with a semantically segmented map (right) generated using OpenMapTiles (<https://openmaptiles.org/>) and Mapbox GL JS (<https://docs.mapbox.com/mapbox-gl-js/api/>).

⁵<http://geojson.org/geojson-spec.html>

⁶<https://github.com/spatial-computing/map-ocr-ground-truth>

⁷<https://www.cs.grinnell.edu/~weinman/research/maps.shtml>

⁸<http://geo.nls.uk/maps/gb1900/>

⁹<http://spacetime.nypl.org/>

¹⁰<https://ngmdb.usgs.gov/topoview/viewer>

¹¹<https://www.oldmapsonline.org/>

3 CASE STUDY: AUTOMATION OF THE GEORECTIFICATION MODULE

This section discusses the automation of the geolocalization component in the Georectification module in more details. The geolocalization component takes a raster map as an input and roughly locates the map on Earth (e.g., within a few street blocks away from the true map location).

For geolocating a map, the component needs to find one or more known locations on the map. For example, if the component can recognize the text label of “The Metropolitan Museum of Art” on the map, it can roughly position the map in Manhattan. However, such location information from the map labels can be ambiguous. For instance, if the component recognizes the text label of “Hudson River” on the map, it can only conclude that the map is somewhere along a river named Hudson, which is not enough information for geolocalization. As another example, the place name “Madison Square” exists in many different cities and countries and does not provide enough information to geolocalize the map.

One possible solution for geolocalization is to use all toponyms collectively to narrow down candidate locations (e.g., [8]). For example, having “Hudson River” and “Madison Square” on the same map implies that the map is most likely in Manhattan, New York. The geolocalization module includes the following tasks: 1) Recognizing Text, 2) Matching and Filtering Words, 3) Finding Candidates, and 4) Ranking Candidates.

Recognizing Text: A common first step to find a set of known locations on a map is to look for textual information. The input to this task is the raster map. The output is a set of words, their minimum bounding boxes (or polygons), and the recognition confidence scores. The output words do not necessarily represent complete place phrases. For example, the output might contain “Hudson” and “River”, but there is no explicit link between these two words.

Matching and Filtering Words: The output from the previous text recognition task is a set of words and may contain incorrectly recognized words. The goal for this task is to remove/correct the incorrect words and link individual words to generate place phrases, e.g., link the separate words “Hudson” and “River” into a phrase “Hudson River”. Here machine learning methods can help to learn the representative features for correctly generating complete place phrases, provided enough training data are available. The representative features might include similarity metrics based on the distance, font style and size, capitalization, and orientation difference between any two words [7]. Furthermore, if several maps from the same approximate location (e.g., London) are available, cross-validating the recognized words from each map can further improve the results [9]. The input to this task is a set of recognized words, their location on the map, and their recognition confidence, and the output is an array of place phrases.

Finding Candidates: Once several place phrases are available, a geocoding process that compares the place phrases with a reference dataset can determine their geocoordinates. The geocoding process might return multiple candidate locations for each phrase. The input to this task is a set of place phrases, and the output is a set of candidate locations for each phrase.

Ranking Candidates: The input to this task is a set of phrases and a set of candidate locations for each phrase, and the output is

the approximate location of the raster map. This task exploits the spatial consistency among the candidate locations for each phrase and ranks them based on the level of consistency. In other words, the task needs to detect and remove the outlier candidate locations and summarize a set of “inlier” locations to geolocalize the map. The summarization process can be a simple clustering algorithm that groups nearby candidates and uses the centroid of the largest cluster as the geolocation of the map.

In summary, each task in the geolocalization component is sufficiently modularized such that at runtime, multiple instances of the same task can run in parallel as well as can have more than one implementation. For example, for the first task, if an automatic text recognition job does not produce sufficient words from a particular low-resolution map, a crowdsourcing job to transcribe the map text can augment the results or replace the automatic job entirely and then feed the results to the next task. As another example, some maps may come with companion text about their location (e.g., “A map from New York City, south of Central Park”). Such descriptive metadata can also provide us with candidate locations without the text recognition step. Furthermore, the map collar usually contains a substantial amount of metadata (map titles such as Los Angeles, California), which can help to limit the search scope of finding candidates and therefore improve the results. The modularized design enables the injection of such information without creating a separate processing pipeline.

3.1 Preliminary Implementation & Results

This section presents the preliminary implementation and experimental results of the geolocalization module using Google APIs for text recognition and geocoding¹². We used the Google Vision API¹³ with the Document Text Detection mode on the input raster map to recognize the words along with their locations and recognition confidence. Then we grouped all the recognized words with a confidence score higher than 90% together into a text corpus. Next, we used the Google Maps Geocoding API to parse and geocode the corpus and obtain candidates for the approximate locations of the input map. The benefit of using the Geocoding API¹⁴ is that the API may account for minor misspellings in the recognition results and is often able to return candidate locations from the large corpus of words. We only consider the first candidate, if more than one candidates are returned. It is important to note that the Geocoding API utilizes contemporary data, so we implicitly assumed that at least some of the recognized toponyms on the historical map still exist today.

We tested our preliminary implementation with 500 randomly selected scanned maps from the NYPL georectified map collection¹⁵. To evaluate the results, we assumed the geolocation of each map is the average of its control points and calculated the distance between our results and the benchmark geolocations. Figure 2 shows a histogram of the distance for all historical maps in our test dataset. The experimental results are promising: 37% of the automatically

¹²Open source tools for these purposes are also available, such as the USC Strabo for text recognition from maps and the Texas A&M geocoder

¹³<https://cloud.google.com/vision/>

¹⁴<https://developers.google.com/maps/documentation/geocoding/start>

¹⁵<http://spacetime.nypl.org/#data-mapwarper>

identified geolocations are within 15 km of the benchmark geolocation, and 28% within only 5 km. This approximate geolocation could allow both crowdsourced users and automated procedures to more efficiently find reference points for georectifying the maps.

We manually investigated the first 50 maps in our randomly selected list to learn more about the strengths and shortcomings of our implementation. In several cases, the Geocoding API returned more than one candidate, but the first one was not the closest candidate to the benchmark. As mentioned before, we selected the first candidate from the list of the returned candidates to calculate the distance. A better strategy to choose the candidate is expected to improve the results. In our crowdsourcing web application, we let the users to see all the candidates and choose one themselves. In some cases, the Geocoding API picked up a street name from the text corpus as a city name, and therefore returned a candidate that might be far away from the benchmark location. For example, in one case the word “Amsterdam” was recognized in the map, which is the name of a street in New York City, however the Geocoding API returned the city of Amsterdam in Netherlands. In cases where no candidate was returned, the text corpus was either too small (i.e., very few words were recognized from the map), or too noisy (e.g., too many street numbers were recognized). We plan to examine natural language processing techniques to prune and organize the text corpus before sending it to the Geocoding API.

Figure 2: Histogram of the distance between the automatically identified geolocations of the maps and their benchmark geolocations. All of the bins, except for the rightmost one, show the number of maps geolocated within the distance range given by the bin label from their benchmark locations. The rightmost bin, labeled as “No Results”, shows the number of maps for which no candidate geolocation was returned by the geolocalization module.

4 DISCUSSION AND OUTLOOK

This paper presented Kartta Labs project, an ongoing open-source and open-data project aiming at organizing the world’s historical maps and making them universally accessible and useful. The paper outlined the projects technical modules and provided some preliminary results of the module for finding approximate locations of a raster map. We plan to continue developing each module with

both crowdsourcing and automated techniques, and aim to identify collaborators to develop and test the modules.

ACKNOWLEDGMENTS

We thank Amol J. Kapoor and anonymous reviewers for their thorough review of this paper and the constructive comments.

REFERENCES

- [1] Mauricio Giraldo Arteaga. 2013. Historical map polygon and feature extractor. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on MapInteraction*. ACM, 66–71.
- [2] Yao-Yi Chiang, Weiwei Duan, Stefan Leyk, Johannes Uhl, and Craig A Knoblock. in press. *Using Historical Maps in Scientific Studies: Applications, Challenges, and Best Practices*. Springer Publishing Company, Incorporated.
- [3] Yao-Yi Chiang, Stefan Leyk, and Craig A Knoblock. 2014. A survey of digital map processing techniques. *ACM Computing Surveys (CSUR)* 47, 1 (2014), 1.
- [4] Weiwei Duan, Yao-Yi Chiang, Craig A Knoblock, Vinil Jain, Dan Feldman, Johannes H Uhl, and Stefan Leyk. 2017. Automatic alignment of geographic features in contemporary vector data and historical maps. In *Proceedings of the 1st workshop on artificial intelligence and deep learning for geographic knowledge discovery*. ACM, 45–54.
- [5] Christopher Fleet, Kimberly C Kowal, and Petr Pridal. 2012. Georeferencer: Crowdsourced georeferencing for map library collections. *D-Lib magazine* 18, 11/12 (2012).
- [6] Yuhao Kang, Song Gao, and Robert E Roth. 2019. Transferring multiscale map styles using generative adversarial networks. *International Journal of Cartography* (2019), 1–27.
- [7] Haowen Lin and Yao-Yi Chiang. 2018. SRC: automatic extraction of phrase-level map labels from historical maps. *SIGSPATIAL Special* 9, 3 (2018), 14–15.
- [8] Jerod Weinman. 2017. Geographic and style models for historical map alignment and toponym recognition. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 1. IEEE, 957–964.
- [9] Ronald Yu, Zexuan Luo, and Yao-Yi Chiang. 2016. Recognizing text in historical maps using maps from multiple time periods. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 3993–3998.