A Framework to Access Handwritten Information within Large Digitized Paper Collections

Liana Diesendruck, Luigi Marini, Rob Kooper, Mayank Kejriwal, Kenton McHenry
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Email: {ldiesend, lmarini, kooper, kejriwa1, mchenry}@illinois.edu

***Abstract*—We describe our efforts with the National Archives and Records Administration (NARA) to provide a form of automated search of handwritten content within large digitized document archives. With a growing push towards the digitization of paper archives there is an imminent need to develop tools capable of searching the resulting unstructured image data as data from such collections offer valuable historical records that can be mined for information pertinent a number of fields from the geosciences to the humanities. To carry out the search we use a Computer Vision technique called Word Spotting. A form of content based image retrieval, it avoids the still difficult task of directly recognizing the text by allowing a user to search using a query image containing handwritten text and ranking a database of images in terms of those that contain more similar looking content. In order to make this search capability available on an archive three computationally expensive pre-processing steps are required. We describe these steps, the open source framework we have developed, and how it can be used not only on the recently released 1940 Census data containing nearly 4 million high resolution scanned forms, but also on other collections of forms. With a growing demand to digitize our wealth of paper archives we see this type of automated search as a low cost scalable alternative to the costly manual transcription that would otherwise be required.**

I. Introduction

Innumerous digital archives of scanned documents are being made available to the public. While the information contained in small to middle sized collections can be retrieved manually in a reasonable amount of time, this is not the case for large collections composed of terabytes of data spanning millions of images. In particular, digital archives of handwritten forms currently possess no viable or practical means of searchable access. The possibility to search images of handwritten text is fundamental for providing genuine access to the digitized data such as the 1940 Census forms released by the US Census Bureau and the National Archives and Records Administration (NARA), a collection containing 3.25 million images and approximately 18 terabytes of raw image data. Information from collections such as these are valuable historical data sources that could potentially be used in a variety of eScience efforts if the contained images of handwritten data were searchable. Census data for example can offer insights into climate change and population movement a century or more ago. The current means of providing searchable access to such data is through manual transcription, a process involving thousands of people and months of time. We focus here on developing an automated, low cost, and scalable alternative.

Multiple-author handwritten text recognition is still an open and actively researched topic within Computer Vision [1], and therefore an alternative approach based on the increasingly popular approach of content-based image retrieval (CBIR) was considered [2, 3]. Here, partially using the Word Spotting methodology [4, 5, 6, 7, 8, 9, 10, 11], a distance metric between images containing handwritten content is defined. Given a query image, multiple results consisting of its top matches are returned. Although the user still needs to search within these results, which might include bad matches, the search is downscaled from billions to a small number of images. The human involvement in this final stage of the search is then utilized to improve future query results through a passive crowd sourcing element.

In order to provide this image-based searchable access to the collections' content, we use High Performance Computing (HPC) resources to handle fundamental pre-processing steps that are computationally expensive (Figure 1). First, forms are segmented into cells based on the forms' columns and rows. Each form of the 1930 Census, our test collection, contains 1900 cells producing roughly 7 billion cell sub-images for the entire collection. Next, 30-dimensional feature vectors are extracted from each cell image generating an equal number of signature vectors. Finally, these signature vectors are clustered using Complete-Linkage Hierarchical Agglomerative Clustering and searchable trees are built based on the resulting hierarchy; these are later used as indices to speed up the searching process.

The sections that follow provide a brief background on handwritten recognition, describe these pre-processing steps in more detail, focus on the framework capabilities, and discuss results obtained with the proposed framework for searchable access to handwritten data within digitized archives.

II. Background

The scientific literature focusing on handwriting recognition is extensive. Nevertheless, most papers focus on constrained instances of the problem and leverage these constraints to achieve the desired results. Many studies assume a known language model (e.g., probabilities of word pairs) and use it during the retrieval process [12, 13, 14]. This approach however is usually applied to single author writings and can only be used in free-text documents, which exclude forms in general. Srihari et al. [15] concentrated in a narrow vocabulary

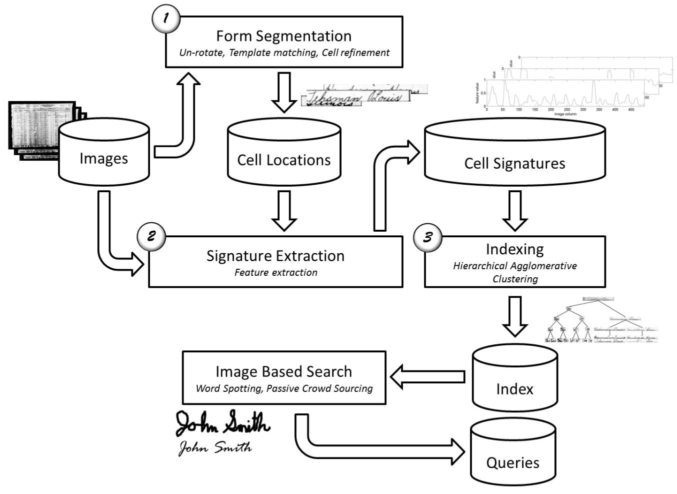


Figure 1. *A flow chart of the 3 pre-processing steps required to provide the image based search on the 1940 Census data. 1: The spreadsheet-like Census forms are segmented into individual cells. 2: A numerical signature is constructed to represent the handwritten contents of each cell. 3: A hierarchical index is constructed over the cell signatures.*

and successfully achieved recognition of handwritten text by various writers. This system, which starts by recognizing a numerals-only ZIP code and then matches the text to a restricted group of possible addresses within the specific zone, is used by the United States Postal Services. Manmatha and colleagues [9] achieved 75% recognition precision but viable datasets need large amounts of training samples per category. Although Milewski et al. [16] focus on forms, their research deals with a very small lexicon of medical terms.

Manmatha et al. [4, 6] suggested that word images could be retrieved from handwritten manuscripts without recognition. The idea was that these images could be described by a set of word features, among them the upper profile, lower profile, ink/background transitions and projection profile (amount of ink per column) of word images. These features are then combined into a feature vector and represent an image "signature" or descriptor. The similarity between two signature vectors can be judge by comparing them according to some predefined method. Clearly, small variations in style, as character spacing for example, do not change the meaning of a word and thus the comparison method should take possible variations into account or alternatively the signature vectors should be invariant to these changes. Dynamic Time Warping (DTW) [17] is commonly used as a signature comparison method to overcome some possible variations in handwriting with the downside of being relatively time consuming. Rath et al. [8] obtained signature vectors invariant to some styling discrepancies by applying Fourier Transform to each of the profiles and using their first coefficients to compose the signature vectors. With the same intent, we apply the cosine transform to each of the upper, lower and projection profiles and use the first 10 coefficients of each to build the feature vectors which can be then successfully compared using a simple Euclidian distance.

Typically, once feature vectors are extracted from all images, they are used to cluster the images into k clusters. The most interesting clusters are then annotated and are used as indices for the searching procedure, thus enabling ASCII search. The amount of clusters (k) is usually calculated using Heap's law [18] which estimates the vocabulary size based on the size of the corpus. However, in our case k cannot be estimated from Heap's law since prose and forms have different word frequencies. Moreover, the users are likely to require searchable access to unusual or obscure words, such as less common private and family names. Finally, even if k was known and the searchable terms were only very frequent ones, the multiple handwritten styles are likely to generate very heterogeneous clusters.

As an alternative to this classical Word Spotting approach, the described system is queried by images from which feature vectors are extracted. The searching process can then be based solely on comparing the query vector to each of the signature vectors in the database. Nevertheless, for large collections such as the 1940 US Census it is not a practical approach and an alternative type of index is needed. This is accomplished by using a Hierarchical Agglomerative Clustering method to build a hierarchy between the signature vectors, and thus the images; cluster trees are built based on the resulting cluster dendogram and used later during the searching process. The clustering and indexing steps will be described in details in the sections below.

III. Pre-processing

As previously mentioned, three computationally expensive pre-processing steps are required in order to provide searchable access to archives using the framework presented. These steps consist of segmenting the image forms, extracting the signature vectors that are used to describe each cell in an image, and clustering the vectors into cluster trees which will be used as indices to the system.

A. Form Segmentation

This first pre-processing step aims to accurately segment the scanned images of the spreadsheet like Census forms into sub-images of cells. These cells are the units of information we wish to search over. The layout of the 1930 US Census forms resembles a grid mostly containing 50 rows and 38 columns of data. The scanned forms usually contain many physical imperfections, e.g. tears, smudges, and bleed-throughs, and are often slightly rotated due to imperfect alignment during the scanning process. Previous work in form segmentation [19, 20] was adapted to deal with the above mentioned problems, and paying particular attention to the resource demands required in terms of processing the 3.6 million high resolution (20-60 megapixel) Census form images.

Initially, an individual form is loaded as a grayscale image into memory. Each image is down-sampled to 25% of its original size so to improve the speed of processing it. The down-sampled image is then binarized so that each pixel is assigned a value of 1 (black pixels such as text and lines) or 0 (white pixels such as the ones present in the image background). The binarization process makes the image easier to deal with while also partially removing noise, such as smudges and CCD noise. A morphological thinning process is then applied to erode the white regions around the forms. Next, all lines and handwritten text, likely several pixels thick in the original image, have their thickness reduced to one single pixel. Care must be taken when thinning an image as it can lead to new artifacts [21] and thus we utilize a thinning approach that was specifically designed for character recognition [22]. Finally, any rotation generated during the scanning process is corrected so to facilitate the cells extraction (see Figure 2). This is done by first detecting long horizontal lines in the image using a Hough transform [23]. The transform is costly to carry out on large images and is one of the main reasons for the previous down-sampling of the images. In general, the majority of the lines found are horizontal form lines with a few additional long vertical lines. A histogram is built with the angles of these lines, the largest bin is identified, and the angles that contributed to the chosen bin are averaged to identify the rotation of the form. The image is then un-rotate while assuming the rotation was around the center of the image.

Subsequently, straight lines within the rotation-corrected form are detected by summing the ink pixels along the horizontal and vertical axis of the image. These sums are used to identify locations above a certain threshold, which is based on the dimensions of the image, as locations of a form line. Though more robust than the Hough transform, the resulting lines will still have extra and missing form lines (see Figure reffig-segmentation). To deal with these missing and extra lines, a form template was constructed by hand labelling all horizontal and vertical form lines from one single form among the approximately 3.6 million Census images. The template stores the number of form lines along with their relative locations. After the basic lines are detected in each rotation-corrected image, we search through the space of 2D rigid transformations, limited to scaling and translation, that best aligns the lines found in the image to the ones in the template. We search this space of transformations in an efficient manner taking into account known constraints such as the number of lines in the form. Each possible alignment is evaluated using either a dot product of the transformed template with the binarized image or using dynamic time warping [17] between the template lines and the image lines detected. The dynamic time warp allows for comparisons to be made despite missing or extra lines between the two sets. Both methods give comparable results and run at nearly the same speed. In order to improve the speed of this step, the form matching of horizontal and vertical lines can be performed separately, converting a 2-dimensional problem into two much smaller 1-dimensional ones.

Once the template matching step is finished, the cell contents can be extracted using the fitted template line positions to delineate a particular row/column (see Figure 2). The retrieved contents of a cell might be chopped on the sides or include portions from neighboring cells as a result of the image down-sampling, the precision of the estimated angle of rotation, and the underlying assumption that the form was rotated about the center. Thus, a refinement step, which consists of a small-scale version of the previous steps, is carried out on the retrieved cell to correct the possible imperfections. Finally, the location of each cell within the image is then stored in the database as individual grayscale images sliced from the original form image.

B. *Signature Extraction*

In order to apply the Word Spotting approach to our dataset, feature vectors need to be extracted to represent each form cell. This pre-processing step focuses on extracting the relevant features of the handwritten words.

Wordspotting techniques obtain best results when the feature vector is extracted from clean text devoid of noises such as borders, small spots/smudges, extra padding, etc. These image artifacts are reduced through additional refinement steps performed on each cell's image. First, it is rescaled to have a fixed canonical height since some Word Spotting techniques are height-dependent. Then the resulting image is binarized so that the ink strokes, our focus of interest, are among the non-zero values in the image. The next step is to attempt to remove white pixels that are not part of the text but are remains of the cell's border. In addition, small isolated areas of white pixels, which are not part of the desired text, are also removed. Finally, white ink pixels are centered within the cell's image.

With the refined cell image at hand, three features described by Rath et al [8] are extracted from the cell's content. The upper profile is found by scanning vertically the cell image until getting to the first ink pixel of each vertical pixel line. The bottom profile is found by a similar process starting at the bottom of the sub-image. The projection profile is calculated by summing the pixel values along the vertical axis of the cell image.

A discrete cosine transform is computed for each of the profiles and the first 10 coefficients of each are used to build the cell's signature vector. Since only the lower coefficients are preserved, high- frequency components are discarded helping to ignore undesired details that result from handwriting variations. This approach also guarantees that every vector will be exactly the same size, independently of the sub-image dimensions, which facilitates future comparisons between different feature vectors. The resulting feature vector for that particular cell is then added to the database as an array of doubles.

C. *Signature Indexing*

Altogether roughly 7 billion cell images compose the 1930's Census data. This is the number of image cells that would be compared to a query image if the system should be searched in a linear manner. Assuming that single comparison takes a very generous 1 millisecond to be executed, it would take 81 days just to carry out all the comparisons needed for one single query.

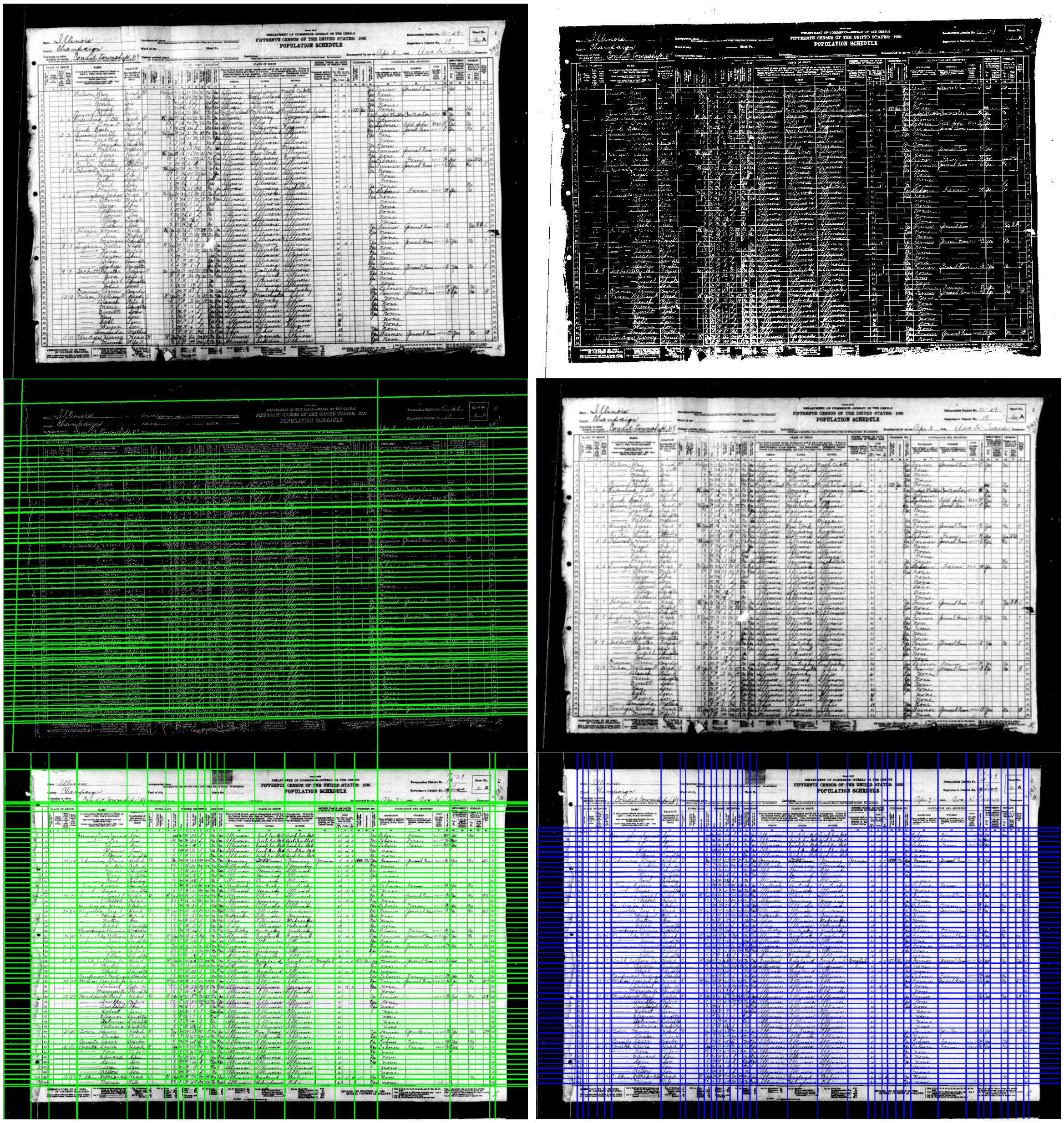


Figure 2. *Top left: An example 1930 Census form image containing gray values between 0 (black) and 1 (white). Top right: The thinned threshold image with inverted values for black and white. Middle left: Lines found with a Hough transform. Middle right: The un-rotated image with horizontal/vertical form lines aligned with the rows/columns of the image. Bottom left: The form lines found within the rotation corrected image. Note the existence of missing form lines as well as non-form lines at the borders of the form. Bottom right: The form lines obtained by matching a template to the lines obtained from the rotation-corrected image.*

Usually a sort of index of the entries within the database is constructed to speed up the search. The index is a list, possibly hierarchical or otherwise structured, of signature vectors representing each image in the database. Here, the index built is similar to a binary search tree, and the search procedure used takes advantage of its structure so not to search through every entry in the database aiming for a logarithmic number of comparisons per query.

Hierarchical Agglomerative Clustering with Complete Linkage is used to cluster the signature vectors in a bottom-up procedure until one single cluster remains. The merging steps of the clustering process are used to build a binary tree of clusters, where each cluster is composed by signature vectors that were grouped together by the algorithm and is represented by an average signature vector. When searching the data, a cluster tree can be descended by comparing the query vector to average cluster signature vectors until arriving at a cluster of suitable size. The query vector will then be compared in a linear manner to all the signatures contained in the chosen cluster.

Since the Hierarchical Agglomerative Clustering with Complete Linkage is very resource demanding, alternative clustering and hierarchy-building algorithms were considered. Due to its faster running time, tests on using Single Linkage instead of Complete Linkage were performed, but yielded unsatisfactory results. Kd-trees were also considered as a possible solution, but the large high-dimensionality of the signature vectors make the search in the tree very inefficient. K-means was not adequate here since, as explained before, k could not be estimated. Approximations to any clustering algorithms or kd-trees were deemed to be prejudicial due to the complexity of the task at hand.

The naive Hierarchical Agglomerative Clustering with Complete Linkage implementation has $O(n^3)$ time complexity and possibly $O(n)$ space complexity. The approach used here [24] is bound by $O(n^2 logn)$ time and $O(n^2)$ space complexity. Due to the quadratic element of both time and space complexity is not practical to build one single cluster tree for the entire vector set. Instead, the vectors were divided by states, reels within states and categories (columns in the Census forms). Therefore the system's index is composed by multiple cluster trees. North Carolina's index, for example, is composed by 2280 cluster trees (60 reels x 38 categories) each requiring 50 GB of memory during its building process. Since most queries are performed within a specific category, usually only 60 cluster trees are searched per query.

The cluster trees are represented by arrays, which contain information such as the number of elements each cluster node has, the average signature vector of each cluster node, and which cluster is the right/left child of every node. A 50,000 signature vector cluster tree needs around 25MB of memory (or storage space). Based on the simplicity of these data structures and their relatively small sizes, the trees are serialized to binary arrays. These arrays can then be stored in the file system or in a database.

IV. A Framework for Image Based Search of Handwritten Content

The image based search discussed in previous sections is only one part of the proposed method for providing searchable access to archives of handwritten forms. While Word Spotting provides automated low-cost searchable image-based access to the archive information, it is not perfect. Here, the image-based search is supplemented with both active and passive crowdsourcing. Specifically, as users utilize the system they will improve its search capabilities. While the active crowdsourcing through a reCAPTCHA-like [25] step is fairly straight forward, we focus on passive crowdsourcing, where users are unaware of their contributions to the system. Potential transcriptions are obtained through a Cursive Text Panel interface, which allows the users to type queries rendered with fonts that look like handwriting or directly draw the query as handwritten text with the mouse or hand (see Figure 3). Since the query input is an integral part of the search procedure regardless of the acquiring of transcriptions, this crowdsourcing element is unobtrusive and transparent to the user. Once the text is obtained from the panel, it will be associated with suitable entries in the archive's data based on the user's usage of the system.

The web interface, written with the Google Web Toolkit (GWT), provides the users with access to all the functionalities outlined so far. In the following subsections we describe the web front end, the crowdsourcing strategy and address the system's ease of use in terms of finding data and carrying out searches, the presentation of results to the user, the system's design so to be robust and fast, and how to make the system maintainable and flexible for possible use with new datasets.

A. *Web Front End*

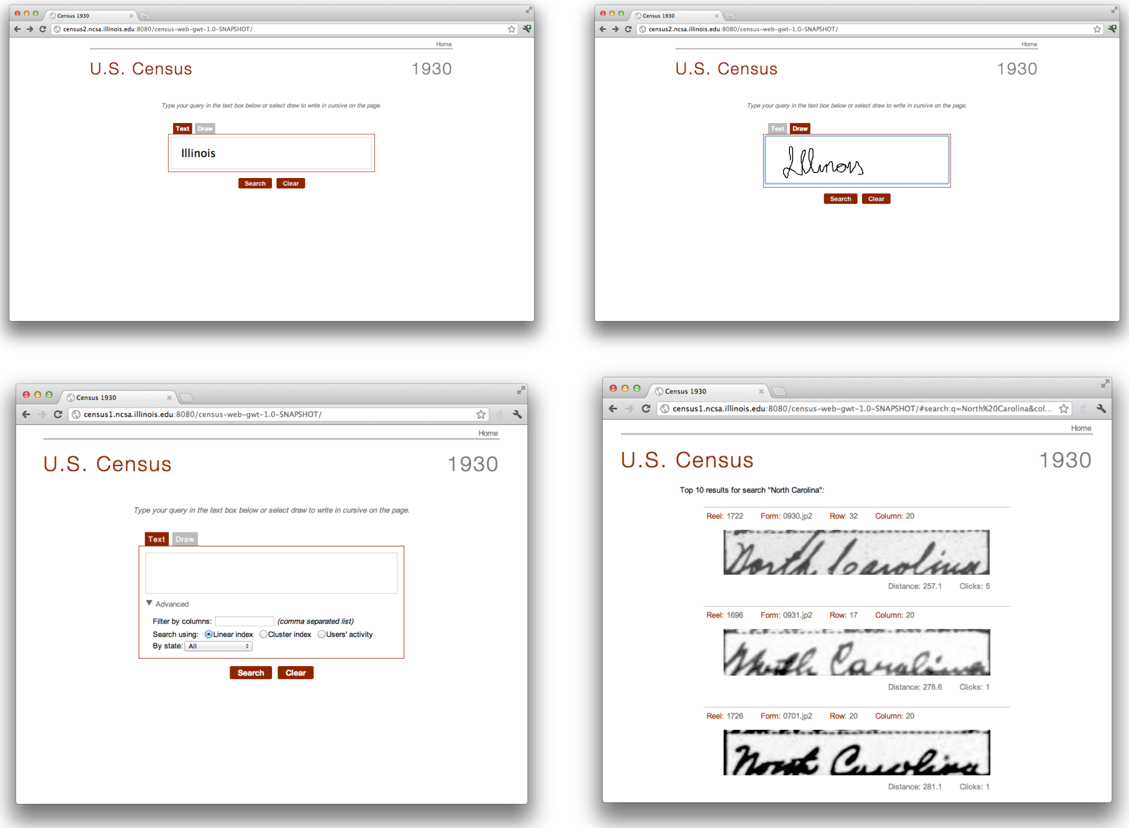
The web front end was developed using the Google Web Toolkit, the Apache Tomcat web server, and an HTML5 canvas element. All information is stored in a MongoDB database [1].

As described before, users are allowed to query the system from the web interface in two ways. By default a text box is presented where the user can type a string to search for. Since an image-based search approach is used in the system, the query must be converted to an image containing handwritten text. When the user clicks on the search button, the contents of the text box are sent to the server using a simple Ajax call against a Search Service implemented as a Java servlet. There the entered text is rendered in ”Rage Italic”, a handwriting-like font. Optionally, the user can use the computer's mouse to draw on the HTML5 canvas the text to be searched. The user should draw the text in cursive handwriting style since the resulting image will be saved in the server and used as the query to the system. (See Figure 3)

Prior to submitting their query, the users are allowed to select which columns of the Census forms they wish to search. The users are warned that more columns entail more indices and thus require more time. Once a query is submitted and saved in the server, its signature vector is computed on the fly. The search can be performed both linearly, i.e. comparing the query against every element in the database, or through the cluster trees computed during the pre-processing.

In the linear search, the query's signature vector is compared using Euclidean distance to all elements contained in the columns being searched. The MongoDB java drivers and server are optimized for this kind of query, with all feature vectors written to disk in a sequential manner. Nevertheless, this approach proved to be too slow when dealing with very large collections and thus the clustering proved essential to the system's success. In the cluster-based search, Java cluster tree objects for the chosen columns are created in memory from the bytes stored in the database. The cluster trees are then queried and return a list of possible matches, i.e. the contents of the final cluster in each cluster tree searched, identified by their MongoDB ids. The results from the query of every cluster tree are merged and a linear search is performed comparing the query against all the resulting feature vectors using Euclidean distance as metric.

[1] http://www.mongodb.org



census2.ncsa.illinois.edu:8080/census-web-gwt-1.0-SNAPSHOT/

Type your query in the text box below or select draw to write in cursive on the page.

Illinois

census1.ncsa.illinois.edu:8080/census-web-gwt-1.0-SNAPSHOT/

Filter by columns: (comma separated list)

Search using: ● Linear index ○ Cluster index ○ Users' activity

By state: All

census1.ncsa.illinois.edu:8080/census-web-gwt-1.0-SNAPSHOT/#search:q=North%20Carolina&col...

Top 10 results for search "North Carolina":

Reel: 1722 Form: 0930.jp2 Row: 32 Column: 20

Distance: 257.1 Clicks: 5

Reel: 1696 Form: 0931.jp2 Row: 17 Column: 20

Distance: 278.6 Clicks: 1

Reel: 1726 Form: 0701.jp2 Row: 20 Column: 20

Distance: 281.1 Clicks: 1

Figure 3. *The main page of the 1930 Census portal. The interface provides two types of search inputs, plain text and drawing. Top left: A user types in a query word within the text box. This entered text is rendered on the server using a font that mimics handwriting. An image generated from this rendered text is used as the query image for the Word Spotting. Top right: A user uses the mouse to write the text to be searched for; the handwritten text results in an image that will be used as the query image for Word Spotting. Bottom left: Search options including the type of search procedure to choose among linear search, cluster search and previous users' results.Bottom right: Partial results returned for a search on the term 'North Carolina' using the text interface including statistics about each item retrieved.*

Finally, the top N closest matching cell images, determined via the smallest Euclidean distances, are returned. Each result is visualized on the page with information about its reel, form, row and column numbers shown above the image of the cell entry and the Euclidean distance between the query and the cell is shown below it. A user can then scroll through these results and click on any of them to view additional information, such as the entire form with the relevant row highlighted as well as metadata associated with the form. The high resolution images are presented as image pyramids [21] through Microsoft's SeaDragon interface to allow for efficient viewing over the web. These image pyramids are generated on demand causing the first user requesting a specific form to wait for its pyramid to be constructed. The resulting pyramid is cached however so that future viewers of the form will be able to view it instantly. The image pyramid creation was observed to take roughly ten seconds.

B. System Scalability

The image processing algorithms involved in this framework require a great deal of CPU resources. Fortunately, as noted before, the majority of this computation can be carried out a single time as a pre-processing step before the system goes live. In addition, this pre-processing can be carried out within and take advantage of high performance computing environments since the segmentation and signature extraction pre-processing steps are both naively parallelizable. In other words, these tasks which act on individual images or sub-images respectively are independent of one another and, thus, can be potentially executed simultaneously. The final pre-processing step, the indexing of the signatures into cluster trees, is not parallelizable if the entire database of cell signatures is to be combined into one single cluster tree. However, the computational and memory costs for clustering a large collection would be so high, that would be deemed impractical. Instead, as mentioned previously, the database is split into multiple indexed collections according to state, column, and reel. The building of an index for each of these smaller collections can be done independently from one another and therefore can be processed in parallel as well.

When dealing with archives, big or small, which could take a considerable amount of time to pre-process and have invaluable information, some kind of system backup is necessary; otherwise, in case of hardware failure, the dataset would have to be reprocessed and the additional information contained in the system, such as information harvested through the crowdsourcing element, would be lost. MongoDB provides replication, high availability, and auto-sharding out of the box. This becomes fundamental when scaling out the system to deal with even larger collections. In addition to the metadata, all the cell files are handled by MongoDB, which by using its sharding capabilities transparently distributes the data among multiple nodes. Since the sharding includes replication of the data on multiple nodes, it has the advantage of functioning as failsafe in case one node goes down.

The no-schema nature of MongoDB proves itself much helpful when expanding the system. It can be used to add new feature descriptors to the image cells or even extend the system to support data contained in new forms of similar or different formats. If a form of different format is added to the collection, the this schema-less property is even more useful since the metadata files will likely contain different sets of fields.

Finally, when scaling from a small collection to a larger one, the query time for linear search will naturally increase significantly. Here, MongoDB provides the advantage of storing data into collections; each collection is a list of memory-mapped files on disk in which individual documents are stored one after the other. Queries are optimized in such a way that a client (for example the front end web application) opens a cursor on the server and results are streamed to the client in a batch of a predefined size. Since the files are memory-mapped and the access is sequential, browsing large collections, if not fast, can be done in a reasonable amount of time.

C. Passive Crowd Sourcing

One of the driving motivations for using the Word Spotting methodology was that by returning the top N closest matches, a human is kept in the loop since any user must look through the presented results to find the desired matches. This, combined with the two query interfaces provided, allow for a passive crowd sourcing element which is aimed to improve search results as the system is used. Usually, active crowd sourcing, such as the popular reCAPTCHA [25], present the user with something unrelated to the site being visited. As such, it is seen as a necessary inconvenience obstructing the access to what the user is truly interested in. For example, in the case of reCAPTCHA two scrambled words are presented to a user to type in. One word is computer generated and used to verify whether the user is a computer script attempting to abuse the system. The other word however is a difficult one to be recognized by OCR, typically extracted from a book. The user, not knowing which is which, types in both words correctly so to proceed, and by doing so helps to OCR a book. Passive crowd sourcing, on the other hand, is designed in such a way that it is invisible to the user.

In our system, the text box widget offers the means to conduct passive crowd sourcing and gain human transcriptions of text. When the user types in text, the text entered is stored and later associated with the results that the user clicks on. The underlying assumption is that users will tend to click on results that match their query. Regardless, the system does not rely on the behaviour of a single user, but records this information for all users. When multiple users entering the same text selected the same image, the system assumes with some level of confidence that the entered text is the same as the one displayed in the image.

In addition to the text entries, the mouse drawn entries also allow for this type of passive crowd sourcing. By allowing the user to draw the text with their mouse, the native offline handwritten recognition problem (based on pixels) is converted to an online handwritten recognition problem (based on point paths) [1]. A significantly more manageable problem than the offline one, transcriptions to the queries can be obtained with roughly 80% accuracy and still have this text later associated with results.

Employing these two passive crowdsourcing approaches, the system provides searchable access to handwritten image data while also allowing the system to improve through usage.

D. Framework Adaptability

Some collections cannot be completely transcribed by hand for many reasons, such as lack of funds or personnel. The presented framework, being easily adaptable, could be potentially used to provide searchable access to archives that would otherwise remain inaccessible. Large personal collections, for example, are less likely to be transcribed for lack of public interest. This framework nevertheless offers interesting possibilities for this kind of collections. For example, drawing the query to be searched is a useful feature when dealing with personal archives. It allows the users to enter queries using their actual handwriting which would be extremely valuable if they wanted to find something they personally wrote in a large archive of scanned documents. Examples might include collections of scientific notebooks, letters, envelopes/postcards, and checks. Even within the US Census data, searching based on handwriting could be used to identify the individual Census takers if this information was no longer available.

Currently, this open source framework can be easily adapted to be used with any kind of form collections. Apart from supplying new form templates to be used in the pre-processing steps, no changes are required. This framework could also be adapted to collections of free-style text. In this case, no templates would be available and the document segmentation could be performed using existing image processing algorithms ([26, 27], for example). All other subsequent preprocessing steps and system usage would remain the same.

V. Experimental Results

We evaluate the performance of the proposed framework both in terms of accuracy of the Word Spotting as well as computational requirements on HPC resources.

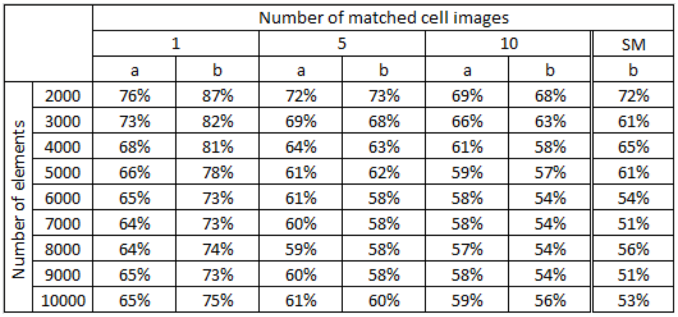
A. Word Spotting

We evaluate the performance of the proposed Word Spotting framework on a small annotated dataset of Champaign County from the 1930s Census. The availability of 10,000 annotated cell images divided into four different categories or columns: marital status, gender, home state and yes/no; allowed the execution of several experiments that contributed to a better understanding of the data set and as an evaluation of the Word Spotting.

In order to verify whether the extracted signatures were indeed representative of the word images, each image in the data set was used as a query to be matched to similar images from the collection. The queried image was excluded from the search space. Two search methods were employed: linear search and hierarchical cluster search. Comparing the query's signature against other images' signatures produces a ranked list of retrieved images based on Euclidean distance. A successful match was defined as one that returns a word image with the same label as the query.

For linear search, the accuracies were 65%, 61% and 59% for the top 1, 5, and 10 matches respectively. The cluster search, where elements were clustered based on their categories, resulted in 75%, 60% and 56%. The precision of the results seems compatible with the ranking of the match, meaning that the average precision of the first ranked match is higher than the precision of the second ranked matched image and so on. Gradual results for sets containing between 2000 and 1000 images are shown in Table 1. Interesting enough, when testing with different sizes of datasets, the hierarchical search seems to improve the accuracy of the first ranked image independently of the dataset size, this advantage is true up to the first four ranked images, but not shown however due to space constraints. In general, the results indicate that similar images can indeed be found based on their signature vectors and that the usage of the cluster trees do not significantly impair the results received.

As described before, the hierarchical search performed over a cluster tree aims to reduce the searching time. If the cluster trees were balanced, which we cannot guarantee due to the clustering process, the search time would be logarithmic instead of linearly proportional to the set size. Indeed, our results for the annotated dataset show that the cluster tree height will have an upper-bound of $2 * logN$ and the search time is logarithmic while the linear search time grows with N (See Figure 4). Nevertheless, there is the question of whether the best possible match to the query is indeed in the final cluster reached by the search procedure. In order to investigate this issue, all cell images were used to query the system, but this time the query was not removed from the possible result set. In successful trials, the final cluster reached contained the original image; otherwise, the trial was classified as unsuccessful. When using the complete dataset (1000 cell images), a 53% accuracy was obtained. Detailed results can be seen in Table 1.



		Number of matched cell images						
		1		5		10		SM
		a	b	a	b	a	b	b
Number of elements	2000	76%	87%	72%	73%	69%	68%	72%
	3000	73%	82%	69%	68%	66%	63%	61%
	4000	68%	81%	64%	63%	61%	58%	65%
	5000	66%	78%	61%	62%	59%	57%	61%
	6000	65%	73%	61%	58%	58%	54%	54%
	7000	64%	73%	60%	58%	58%	54%	51%
	8000	64%	74%	59%	58%	57%	54%	56%
	9000	65%	73%	60%	58%	58%	54%	51%
	10000	65%	75%	61%	60%	59%	56%	53%

Table 1. *Search accuracy results. The data set was composed by images from 4 different categories. The results in columns 1, 5 and 10 represent the average accuracy of the 1st, 5th, and 10th highest-ranked matching images, respectively. A correct match image is defined as one with the same transcription as the query. The results in column SM show the accuracies for the self-matching task, where a correct match occurs when the queried cell is present in the final cluster retrieved. The methods used were: (a) linear search, and (b) hierarchical cluster search with different categories composing different cluster trees.*

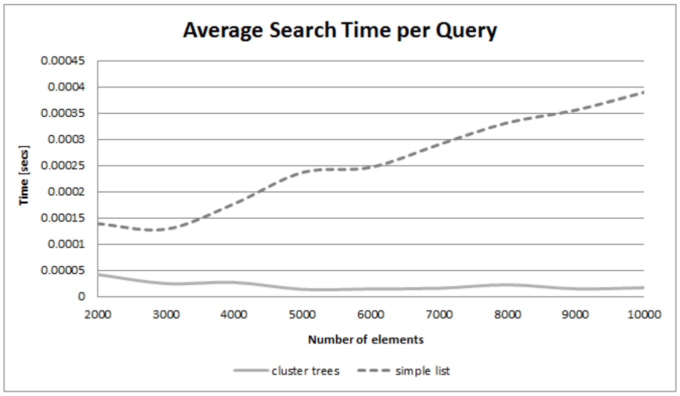


Figure 4. *Average search time for linear and cluster search methods. Notice how the average linear search time grows linearly as N grows, while the search in the clustered procedure is logarithmic.*

These results establish that the best possible match to a query image is not always present in the final retrieved cluster. The hierarchical search clearly loses a significant amount of relevant results during the cluster tree descent. Although considerable, this loss is not surprising considering the significant reduction of the search space. Nevertheless, the hierarchical cluster search seems as the most practical option considering the substantial speed-up of the searching process and its general accuracy rates (which are similar to the linear search results).

B. HPC Benchmarking

The three pre-processing steps of the framework were benchmarked on Steele at University of Purdue and NCSA's Ember at the University of Illinois. Both systems are very different in architecture and capabilities. The Steele cluster consists of around 900 systems with 8 core processors and between 16 GB and 32 GB of memory. When submitting a job on Steele, it is queued based on the required resources; thus we limited ourselves to 16 GB of memory usage on this system. Ember is a shared memory system that consists of 4 systems each with 384 cores and 2 TB of memory. Unlike Steele, the resources at Ember are split in groups of 6 cores and 30 GB of memory. The process used 12 cores and 60 GB, the equivalent to a single node board. Both systems have some local disk space that can be accessed as well as a large parallel file system which hosts the data. The results described here focus specifically on the state of North Carolina which consists of approximately 60 thousand images divided in 60 reels which were processed on Ember and on the District of Columbia consisting of 14 reels mainly processed on Steele.

The segmentation computation requires a little less than 3 GB of memory per image that needs to be processed. Both Steele and Ember were used for the segmentation step. In the case of Steele, the jobs were split into smaller sub-jobs each running 5 threads and segmenting only 250 images, which limits the computation time and guarantees access to the system's faster queue. When using Ember, a single reel (around one thousand images) was processed per job using 12 threads and requesting a total of 8 hours of runtime. The segmentation of a reel took between 5 and 6 hours on Steele and between 1 and 5 hours on Ember. The average time to process the one single image is 132 seconds on Ember and 91 seconds on Steele.

The next pre-processing step is the signature extraction, which was also executed both on Ember and Steele. When using Ember, a single reel was processed per job; on Steele, the jobs were once more divided into sub-jobs of 250 images each. Again, 12 and 5 threads per job were used in Ember and Steele, respectively. The processing took per reel between 1.5 hours and 5 hours on Ember and between 4 and 9.5 hours on Steele. Some of the timing difference can be explained due to overhead of transferring the resulting cell images from Steele to NCSA. An average of 151 seconds was needed to process one image on Ember; the average for Steele was 145 seconds per image.

The final, and more costly, pre-processing step is the indexing of the signature vectors. Due to the high amount of memory needed during the clustering process, this third pre-processing step run solely on Ember. Each of the resulting 2280 North Carolina's cluster trees was processed on average in 3 hours; an average of 123 hours were required per reel and a total of 7401 hours was needed to create the complete index for the state's data. Several CPUs needed to be allocated to the indexing step due to the large amount of memory required. Currently, the indexing computation utilizes one single CPU. Some optimizations can be done to exploit these idle cores. For example, it is possible to create a pipeline where segmentation and signature extraction jobs can be run simultaneously with the resources requested for an indexing job. In this case, the time required for the first two pre-processing steps would be assimilated in the total time of the much time consuming indexing processing. Another significant possible improvement would be to modify the implementation of the Hierarchical Agglomerative Clustering such as to utilize some of the idle cores during the clustering process to 'clean' the priority queue that controls which clusters should be merged.

VI. Conclusions

The need for automated search solutions for digital archives of handwritten documents will only grow as more paper archives go through the digitalization process. Most large collections among the existing centuries of handwritten form archives do not have the wide appeal needed to attract the investment capital required for manual transcription. Nonetheless, these legacy collections can often offer valuable historical data that can be used in a variety eScience efforts [28] if their contents were accessible within the resulting scanned image data. The reported open source framework can be utilized to offer searchable access to other form collections provided very few modifications such as compatible form templates. The system's code consists of the pre-processing code to be run on HPC resources as well as a GWT front end for web access. Though we discussed largely human access of such data in terms of a search engine like interface, automated access is also possible for applications that would attempt to mine this data or gather statistics. The benchmarking performed on the state of North Carolina and the District of Columbia's 1930 Census data helps to estimate the amount of time needed to process the entire 1930 Census collection. Approximately 0.1 days of computation would be required per reel, with roughly a total of 288 days to process the 2878 reels of the entire 1930 Census. By incorporating the pipeline described in the previous section for the idle CPU's during indexing, the time estimated for the complete 1930 Census collection would be a shorter 213 days, assuming the other two pre-processing steps can execute within the unused resources of the indexing.

Acknowledgments

This research has been funded through the National Science Foundation Cooperative Agreement NSF OCI 05-25308 and Cooperative Support Agreement NSF OCI 05-04064 by the National Archives and Records Administration (NARA). This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

References

[1] R. Plamondon and S. Srihari, "On-line and off-line handwriting recogntion: A comprehensive survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[2] R. Veltkamp and M. Tanase, "Content-based image retrieval systems: A survey," 2000.

[3] R. Holley, "How good can it get? analyzing and improving ocr accuracy in large scale historic newspaper digitization programs," *D-Lib Magazine*, 2009.

[4] R. Manmatha, C. Han, and E. Riseman, "Word spotting: A new approach to indexing handwriting," *IEEE Conference on Computer Vision and Pattern Recognition*, 1996.

[5] T. Rath and R. Manmatha, "Word image matching using dynamic time warping," *IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[6] ——, "Features for word spotting in historical manuscripts," *International Conference on Document Analysis and Recognition*, 2003.

[7] J. Rothfeder, S. Feng, and T. Rath, "Using corner feature correspondences to rank word images by similarity," *Workshop on Document Image Analysis and Retrieval*, 2003.

[8] T. Rath and R. Manmatha, "A search engine for historical manuscript images," *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2004.

[9] N. Howe, T. Rath, and R. Manmatha, "Boosted decision trees for word recognition in handwritten document retrieval," *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2005.

[10] A. Bhardwaj, D. Jose, and V. Govindaraju, "Script independent word spotting in multilingual documents," *International Workshop on Cross Lingual Information Access Addressing the Information Need of Multilingual Societies*, 2008.

[11] J. Rodriguez-Serrano and F. Perronnin, "A similarity measure between vector sequences with application to handwritten word image retrieval," *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[12] V. Lavrenko, T. Rath, and R. Manmatha, "Holistic word recognition for handwritten historical documents," *Document Image Analysis for Libraries*, 2004.

[13] T. Rath, V. Lavrenko, and R. Manmatha, "A statistical approach to retrieving historical manuscript images without recognition," *Technical Report*, 2003.

[14] ——, "Retrieving historical manuscripts using shape," *Technical Report*, 2003.

[15] S. Srihari, V. Govindaraju, and A. Shelihawat, "Interpretation of handwritten addresses in us mailstream," *Document Analysis and Recognition*, 1993.

[16] R. Milewski, V. Govindaraju, and A. Bhardwaj, "Automatic recognition of handwritten medical forms for search engines," *International Journal on Document Analysis and Recognition*, 2009.

[17] H. Sakoe and S. Chiba, "Dynamic programming algorthm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1978.

[18] T. Rath and R. Manmatha, "Word spotting for historical documents," *International Journal on Document Analysis and Recognition*, 2007.

[19] R. Casey and D. Ferguson, "Intelligent forms processing," *IBM Systems Journal*, 1990.

[20] J. Liu, X. Ding, and Y. Wu, "Description and recogntion of form and automated for data entry," *ICDAR*, 1995.

[21] D. Forsyth and J. Ponce, "Computer vision and modern approach," *Prentice Hall*, 2002.

[22] L. Lam and Y. Suen, "An evaluation of parallel thinning algorithms for character recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1995.

[23] R. Duda and P. Hart, "Use of the hough transformation to detect lines and curves in pictures," *ACM Communications*, 1972.

[24] W. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of Classification*, 1984.

[25] L. Ahn, B. Maurer, C. McMillen, D. Abraham, and M. blum, "recaptcha: Human-based character recognition via web security measures," *Science*, 2008.

[26] G. Louloudis, B. Gatos, I. Pratikakis, and C. Halatsis, "Text line and word segmentation of handwritten documents," *Pattern Recognition*, 2009.

[27] V. Papavassiliou, T. Stafylakis, V. Katsouros, and G. Carayannis, "Handwritten document image segmentation into text lines and words," *Pattern Recognition*, 2010.

[28] M. Dietze, D. LeBauer, C. Davidson, A. Desai, R. Kooper, K. McHenry, and P. Mulrooney, "The pecan project: Model-data ecoinformatics for the observatory era," *AGU*, 2011.