An Analysis of Write-in Marks on Optical Scan Ballots

Theron Ji *Eric Kim* *Raji Srikantan* *Alan Tsai* *Arel Cordero* *David Wagner*
University of California, Berkeley

Abstract

Optical scan ballot systems are widely used in elections today. However, deployed optical scan systems may not always interpret write-in votes correctly. For instance, if a voter writes in a name but forgets to shade in the corresponding voting target, an optical scanner may not detect the write-in, which could lead to a lost vote. In this paper, we study methods for automatic recognition of write-in marks. We then apply these methods to ballots from an election in Leon County, Florida and study the kinds of write-in marks that are seen in practice. Our results from this election show that voters frequently (about 49% of the time) do not fill in the write-in bubble when entering a write-in vote. Consequently, votes may be lost in current voting systems.

1 Introduction

Historically, write-in candidates have always been an overlooked part of elections. The dark horses who run write-in campaigns are often—though certainly not always—less well-known and under-funded compared to their competitors. Thus, write-ins tend to be an overshadowed topic.

However, there are still strong reasons for allowing write-in candidates. For example, candidates who lose a party primary or candidates who file their paperwork late may want to run a write-in campaign. Also, in some countries write-ins are an important defense against "name-blocking", in which an incumbent manages to prevent his or her opponents from having their names printed on the ballot [1]. These situations make it important that voters have a usable way to enter write-in candidates, and for voting systems to count them accurately. While write-in candidates face long odds, it is not unheard of for write-in candidates to win. A notable list of figures have won primaries through write-ins, including Dwight Eisenhower, Richard Nixon, and John F. Kennedy. Most recently was the 2010 Alaska Senate election, where incumbent Lisa Murkowski won a write-in campaign after losing the Republican primaries [3]. Write-ins are also more common in smaller elections, such as city council or school board elections.

Current optical scan systems support write-in candidates by providing a blank space on the paper ballot where the voter can write in a candidate name of their choice. To enable the scanner to detect write-ins, voters are instructed to fill in a bubble next to the write-in region, indicating their intention to cast write-in vote. Scanners separate out ballots with a write-in bubble filled, so they can be inspected manually by election officials. Therefore, *write-in marks are detected by current optical scanners only if the voter marks the corresponding voting target*. This system works well if voters remember to fill in the write-in bubble. However, it falls apart if the voter writes in a name, but forgets to mark the voting target: such votes are not detected by the scanner and thus are not counted.

A recent instance in which this protocol directly affected the outcome of an election was the 2004 San Diego mayoral election, where Democratic candidate Donna Frye lost to incumbent Richard Murphy because about 4000 write-in votes for Frye did not have the voting target filled in [6]. If these lost votes had been accepted, then Donna Frye would have won the election.

One approach to solving this problem is to put more effort into instructing voters on how to mark their ballots, or to design the ballots so the usage is more obvious. However, this cannot guarantee that every voter will follow the instructions. In this paper we design automated techniques to detect these types of errors. We also apply these methods to a large real-world data set and use the results to gain new understanding about the prevalence of various kinds of problems associated with write-ins.

The contributions of this paper are:

- We develop automated methods for detecting and

analyzing write-in votes, if scanned images of ballots are available for analysis. These methods are suitable for use in next-generation voting systems.

- We analyze the write-in votes and marks found on over 100,000 ballots cast in the 2008 Leon County election. Based upon this, we are able to provide a quantitative analysis of the prevalence of various sorts of voter marks, including the rate at which write-in votes are lost due to limitations of current technology.
- We demonstrate that a large fraction of write-in votes are not detected or counted by current optical scan technology, motivating the benefit of better technology for processing write-in votes.

2 Problem statement

We have two goals in our work: to develop automated methods for detecting write-in votes, and to understand how voters use write-in portions of the ballot in practice.

Current optical scan technology has a significant limitation: it cannot detect write-in votes unless the corresponding voting target has been filled. We wish to create new methods for write-in vote detection that eliminate this limitation. This is of importance because the limitations of current technology have the potential to shape the laws and regulations that govern public elections. For instance, current technology cannot automatically detect a write-in for which the corresponding voting target was not filled, and some states declare those kinds of votes invalid as a matter of law. As a result, even if all ballots are manually recounted by hand (in a setting where those votes can be detected), state law may require that those votes not be counted. To the extent that technologists can eliminate the limitations of current technology, policymakers and election officials may be given additional options for better capturing the will of the populace.

Our second goal is to understand voter behavior with respect to write-ins. In particular, we want to better understand the failure modes that exist for write-in votes. For instance, to what degree do people actually fill out write-in votes in accordance with assumptions made by optical scan systems? To what extent does voter behavior differ from the assumptions made by designers of optical scan systems? Can such disparities lead to lost or misinterpreted votes, and if so, how prevalent are such cases? A specific question we raise is, how often do voters write in a candidate name but fail to mark the corresponding voting target? This situation is especially relevant because it can lead to lost votes in deployed optical scan systems.

To answer these questions, we analyze a large data set of scanned images of all of the paper ballots cast in the 2008 Leon County general election. Individually inspecting ballot images for a large election is time consuming and expensive. (Our data set for one election involves close to a quarter-million images.) Therefore, our exploratory analysis of voter use of write-ins relies upon the methods we develop for automating—*accurately!*—much of the processing of write-in votes.

Our work on automating the detection and processing of write-in votes requires us to automate a number of steps that are currently handled manually. Some of the image processing tasks we wish to automate include:

- Registering all the ballot images to a uniform coordinate system.
- Grouping ballots by ballot style (or layout of contests).
- Automatically identifying all the write-in regions for every ballot.
- Detecting every mark that occurs in a write-in region, whether or not the corresponding voting target has been filled.
- Initially categorizing write-in marks (e.g., as actual write-in votes, emphasis votes, or non-serious votes).

For each of the above tasks, we require a high degree of accuracy, so we also need methods for verifying the accuracy of each step we automate. In other words, we want to be sure at the end of the day that our analysis has fully accounted for every write-in vote in the data.

We make several assumptions in our analysis:

- We assume that we are given scans of the front and back side of each voted ballot. We assume these represent the complete set of ballots.[1]
- We assume that we are given a set of blank ballot images, or *template* images, that consists of all ballot styles in the data set. We do not assume any prior knowledge of which voted ballots correspond to which template.
- We assume that each contest on the ballot is surrounded by a clearly demarcated box.

There are several assumptions we do not make:

[1] We do not verify that our results match up with the actual physical ballots cast by voters. We consider this a separable problem that is out of scope for this work. Prior research has demonstrated both the value of investigating ballot images, as well as methods in which the ballot images may themselves be audited [5].

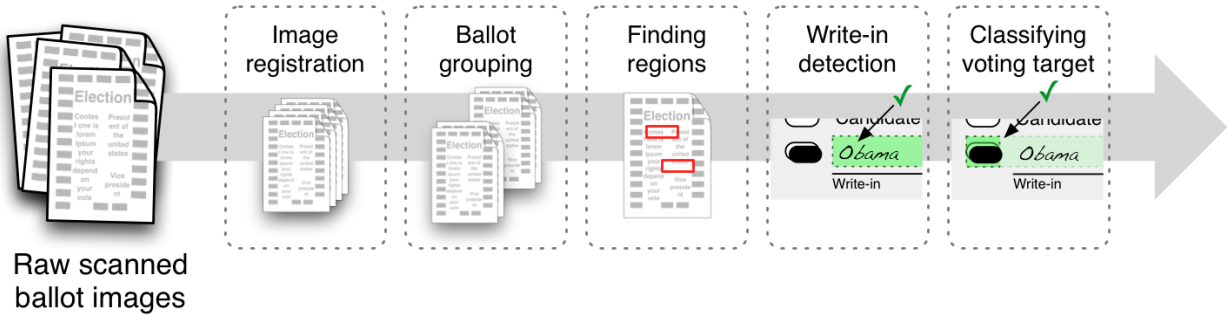

Figure 1: Overview of our approach to automatically detecting write-ins in a set of scanned ballot images.

- We do not assume the images are free from natural scanning errors. For instance, the scanner may introduce variations in alignment or artifacts from creases, folds, or tears. The automated image processing techniques described in this paper are responsible for dealing with these issues.

- We also do not assume any prior knowledge of the election configuration of the ballots. For instance, we do not assume we are given the coordinates of each voting target or the candidate that each voting target corresponds to.

To maximize the applicability of our technique, we adopt several goals:

- Our approach should be generally applicable to a wide variety of jurisdictions and optical scan systems. It should be clear how to generalize our approach to new elections and to the ballot designs of other vendors.

- The methods should be designed to minimize the amount of manual labor required. We do not try to completely eliminate all human interaction. Rather, we think of the system as a system comprising a human and a machine. We use the human operator for their knowledge of the problem domain (e.g., specifics of the local ballot layout), and use the machine for mechanical tasks. For example, in certain cases, we may ask for a limited amount of human assistance, e.g., to identify an example of a voting target or registration mark. Our goal, of course, is to reduce the amount of human involvement required to a minimum.

By developing automated methods for write-in detection, we hope to make it easier to investigate how people write in votes in practice. We can use this knowledge to design more robust optical scan ballots and systems.

3 Automated Write-in Detection

Our process for detecting write-in votes in scanned ballot images involves several steps (see Figure 1).

3.1 Registration

We start with a set of images of marked ballots, and a set of unvoted "template" images, one per ballot layout. Each marked ballot image will correspond to a template image, however the images may not be perfectly aligned: each image may be slightly rotated, scaled, or shifted by a different amount. In addition, some ballot images may be upside-down by virtue of how they were scanned. Before we analyze the content of the ballots, we must first *register* each scanned ballot image so it aligns with its corresponding template image. We use feature-based registration, which is often used in computer vision, by making use of the hashmarks on the border of the ballot image. Our registration algorithm is designed to be robust to common errors, such as artifacts introduced by low-resolution scans, lossy compression, or folded/torn ballots.

The algorithm involves three steps: detection of registration marks, identification of the ballot bounding box, and computation of an affine transformation that aligns the ballot to a common coordinate system.[2]

3.1.1 Locating registration marks

Most optical scan ballots include *registration marks* for the purpose of aligning the image so that it can be interpreted correctly. The exact shape of these marks may vary from one vendor to another. The Diebold/Premier ballot style used in our data set has a border of *hashmarks* (solid black rectangles) along the edges of the bal-

[2] An affine transformation (which accounts for rotation, scaling, skewing, and shifting) is appropriate because the ballots are scanned flat, pressed down inside the scanner. This would not be a safe assumption to make if the ballots might be curled, as is the case in the OpenScan system [15].

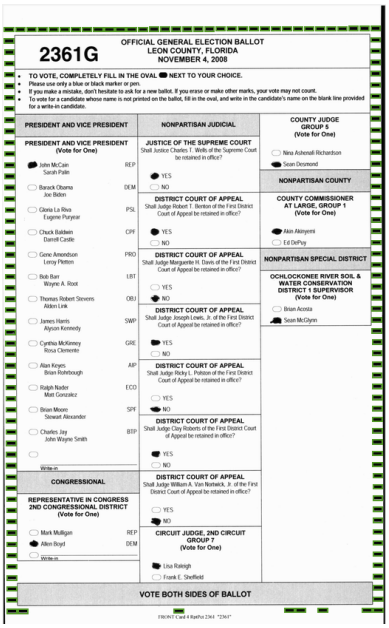

Figure 2: The location of every hashmark in the ballot image is identified through template matching. The detected hashmarks are highlighted in green.

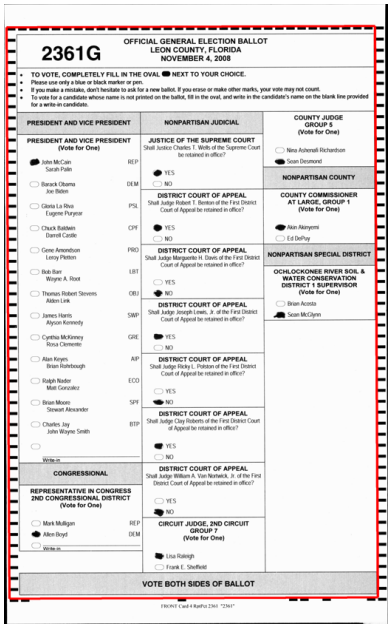

Figure 3: Linear regression is performed on each side of the hashmarks we just identified, to give us four best-fit lines. Notice the slight leftwards skew of the image.

lot which identify the boundary of the ballot and determine the positions where voting targets may appear. Our registration process is specialized to Diebold ballots, but it would be possible to extend it to handle other kinds of ballots as well.

Given an example of a hashmark, we use *template matching* to find the locations of all such hashmarks on each ballot image (see Figure 2). Template matching is a well-established image processing technique that, given a template T and an image I, finds all locations in I where the template T occurs. Template matching can be performed by a straightforward algorithm that tries to place T at each possible offset within the image, and then scores each candidate location using a pixel-wise difference between the pixels of T and the pixels of the corresponding region in I. The best-scoring matches are retained. In our experience, template matching produces highly accurate results due to the uniform nature of the hashmarks. Efficient implementations of template matching are available as libraries. In our work we used the open source computer vision library, OpenCV.

3.1.2 Computing the bounding box

To more robustly register the ballot images, we compute the bounding box of each ballot by detecting the horizontal and vertical lines formed by the hashmarks that surround the ballot. We separate the set of hashmarks into four groups: the two horizontal sets of hashmarks (at the top/bottom edges of the ballot), and the two vertical sets of hashmarks (at the left/right edges of the ballot). Then, we perform linear regression four times, once on each set of points.[3] This finds a best-fit line for each set of hashmarks, which ultimately gives us a bounding box of the ballot with corners formed by the intersection of these lines.

This method is designed to be robust against damaged ballots, which are common in our data set.[4] Sometimes the corner of the ballot is folded, causing us to miss several hashmarks (this is one reason why we did not simply use the corner hashmarks to generate the bounding box). Even if a few hashmarks are missing, linear regression will still find the best fit through the remaining hashmarks. This method is even robust against ballots that are missing an entire edge: as we shall see next, a minimum of two intersecting lines of hashmarks suffices.

[3] For the top and bottom edge, we use simple linear regression to find the line that best predicts the y-coordinate of each hashmark as a function of its x-coordinate. To avoid singularities, for the left and right edge we flip the role of the x and y coordinate: we find the line that best predicts the x-coordinate of each hashmark as a function of its y-coordinate. This step is necessary because the line formed by the left/right edges is nearly vertical.

[4] Our data set included hundreds of ballot images that showed some sort of damage.

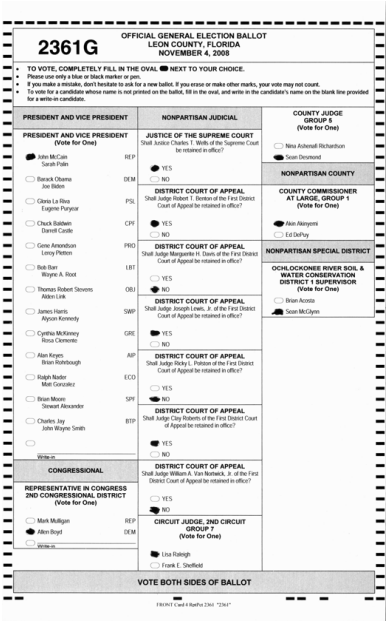

Figure 4: The ballot image, after it applying an affine transformation that maps its hashmarks to their canonical locations in a template image.

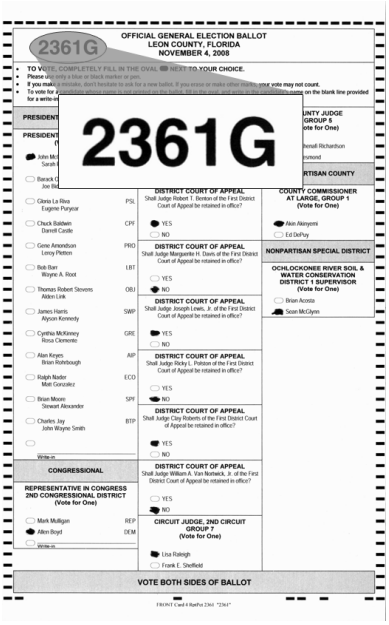

Figure 5: The precinct number is at the top left of every ballot. We OCR this to group the ballots.

3.1.3 Affine transformation

In this final step of the registration process, we use the location of each hashmark to compute the best-fit alignment. We assume we have a template showing the canonical location of each hashmark. For each hashmark in the unaligned ballot image, we associate it with its corresponding hashmark in a template. In particular, we pick one corner of the bounding box (found in §3.1.2), find the hashmark at that corner in the ballot image, and then locate the corresponding corner hashmark in the template. Starting from the corner hashmark, we then scan in both the clockwise and counterclockwise direction along the lines detected in the previous step, in both the ballot image and the template image. This scan process allows us to build a correspondence between hashmarks in the ballot image and hashmarks in the template image.

This correspondence is robust against skewed images since it uses relative positions instead of absolute positions. It is also robust against ballots with missing sides, as it only needs one corner to be able to find the relative positions of the hashmarks. As long as the ballot image is missing no more than one consecutive segment of hashmarks, we can map all of the hashmarks present on the ballot image to the template image.

Last, we solve for the best affine transformation that maps the ballot hashmarks to the template hashmarks, using least-squares regression to find the transformation that minimizes the sum of squared residuals in the result.

The ballot image can thus be brought to the universal coordinate system. The registered ballot image as a result of these steps is shown in Figure 4.

3.2 Ballot grouping

Once the ballots are aligned, we identify the blank ballot template image that each voted ballot corresponds to, and group all the ballots accordingly. Within a group, all ballots have the same layout and the same set of contests. Moreover, each contest and its corresponding write-in region is at the same location in the image on all ballots within the group. This enables us to optimize the process of finding write-in regions because once we have found the location of all write-in regions within a single ballot, we can infer the location of the write-in regions on all the other ballots in the group.

We group ballots by their precinct number. We take advantage of the fact that in our election, ballots in the same precinct also share the same ballot style. The ballots in our data set contain a four-digit precinct number printed in the upper-left corner of the ballot, at the same location on all ballots, as shown in Figure 5. Looking only in the region containing the precinct number, we apply OCR (optical character recognition) to obtain the text. Our implementation uses the Tesseract OCR engine.

Next, we identify the template (blank ballot) for each precinct using *form extraction* [4]. Form extraction is an image processing method that identifies the horizontal

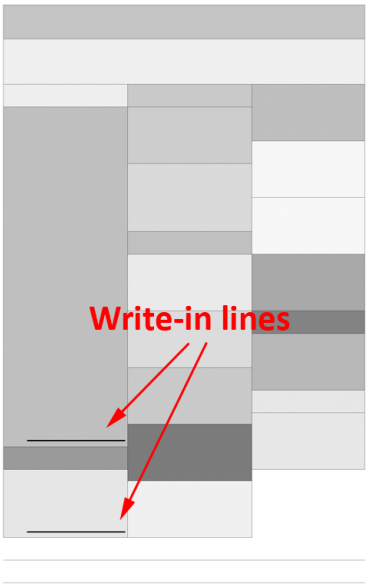

Figure 6: The result of form extraction. It detects the location of all boxed contests. It also finds write-in lines (narrower horizontal lines contained within a box).

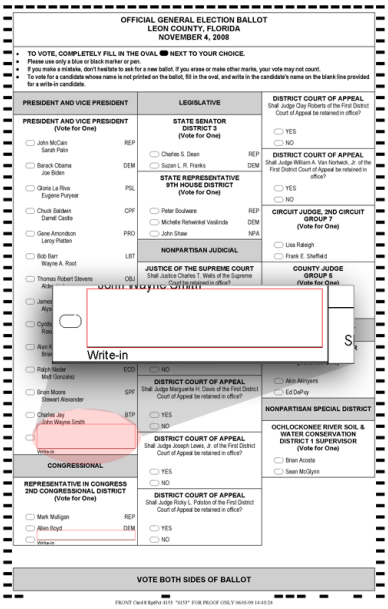

Figure 7: Our algorithm automatically infers the region for write-in marks.

and vertical lines in the image and then finds all of the rectangular "boxes". In this case, the boxes generally surround contests or headers. See Figure 6 for the result of form extraction applied to a Leon County ballot.

We use the set of box locations as a fingerprint of the ballot style. To identify the template corresponding to a particular precinct, we apply form extraction to a single ballot image from that precinct, identify the ballot style fingerprint, and then associate that precinct with the template with the same fingerprint. (We assume no two templates share the same fingerprint; if a collision is detected, operator assistance may be necessary.)

3.3 Finding write-in regions

Next, we find the region of the ballot where voters may write in a name. We list several possible approaches:

1. Manual: The simplest method would simply have a human operator specify the location of each write-in region. This would be the most general in the sense that it would work for any type of ballot style. This would only need to be done once for each ballot template, as the previous grouping step would have already grouped each marked-ballot with its corresponding template image. However, this would become time-intensive if there were a large number of ballot templates. Also, any slight inaccuracy in selecting the region might pose a problem for classification in the next step.

2. Template Matching: In this approach, we show the operator a sample ballot and ask him/her to select a region containing the text "Write-in" (or any other label that identifies a write-in region) and use this as a template. We use template matching to find all instances of it, and then look for a rectangular empty space above it. Assuming that all write-in regions on the ballot are labelled in this way, this allows us to find all write-in regions efficiently, with minimal operator assistance. This method is useful is useful for ballots that do not surround each contest with a black box or do not contain differentiated write-in lines. In these cases, if the write-in spaces are labeled, we can still identify where each write-in region is located.

3. Form Extraction: Alternatively, we can eliminate the need for operator assistance by inferring the location of each contest on the ballot. Form extraction identifies the horizontal and vertical lines on the ballot image. We find the intersections of these lines; the boxes that result are interpreted as the contests on the ballot. Then, we note that write-in regions often contain a horizontal write-in line, which is contained entirely within a contest box (so the line is shorter than the width of the box). We find all lines of this form (see Figure 6).

   Next, given the location of a write-in line, we find

the empty region above it (which is left empty so the voter can write in the name of a candidate) by scanning row-by-row upwards until we reach a row containing several black pixels. The empty region above the write-in line is interpreted as the write-in region (see Figure 7).

We apply write-in region detection to the template (blank ballot) images. Since each marked ballot is already grouped by ballot style, and each ballot group is associated with a template, we only need to extract the write-in regions for each template to know their location on every ballot. Also, if the templates were generated from the original files (our templates were given to us as PDF files), they may already be aligned and free from noise, improving accuracy when working with templates.

In our implementation, we use form extraction. The other two approaches are more general, making fewer assumptions on the ballot style. Appendix A analyzes a variety of ballots to determine how many ballots can be handled by one of these approaches.

3.4 Write-in detection

Given the location of all write-in regions, we look for ones that appear to contain handwritten marks. We first convert the grayscale images to black and white, using Otsu's thresholding algorithm [12]. We then count the number of black pixels within each cropped-out write-in region and conclude that a write-in mark exists if this count exceeds some threshold. The count of black pixels effectively estimates the amount of area covered by ink in a region.[5] A threshold, which could be specified in terms of pixels or area, gives us a buffer against stray noise introduced by the scanner or registration.

We selected the threshold by constructing a histogram of the set of all counts. Experimentally, we found a clear separation between marked and non-marked regions, making it easy to choose a reasonable threshold. In selecting the threshold we chose to err on the side of caution and accept more false positives (which can be thrown away on inspection), over false negatives that could lead to missing a handwritten mark.

In our experiments, we used a threshold of 500 black pixels: regions with at least 500 black pixels (approximately 8 mm^2 of dark ink) were classified as containing a handwritten mark, while regions with fewer were classified as empty. The average write-in region is 31,311 pixels large (441×71), so this threshold number is around 1.6% of the area of the region.

[5]The area of a pixel can be determined given the DPI of an image. For instance, our ballot images were scanned at 200 DPI, which works out to about 62 pixels per mm^2.

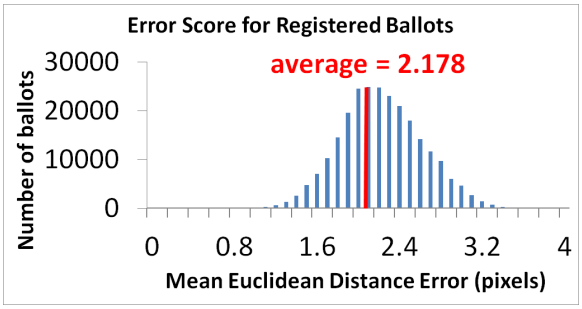

Figure 8: Evaluation of the accuracy of our registration procedure. The x-axis represents the average distance each voting target is from the corresponding voting target in the canonical template image. Most images are registered to within at most 3 pixels error.

3.5 Voting target association

Lastly, we want to determine whether the voting target associated with this write-in was marked or not. To find the voting target, we search for the voting target closest to the center of the write-in region. We can then determine if the corresponding target is actually filled by template matching with an image of an *empty* voting target: if a match is found, then the write-in voting target is not filled in. This implies that any mark the voter makes in the target, even if it is ambiguous, will be classified as "filled".

At this point, our pipeline is now complete, and we have counts of the number of write-ins, as well as the number of filled and unfilled write-in voting targets for the write-in votes.

4 Evaluation of our approach to finding write-ins

We have presented techniques to automate the detection of write-ins given a large set of scanned ballot images. In this section we evaluate the performance of our techniques on a real data set. Since our process involves a series of steps, we must have a way to establish confidence in each step.

4.1 Registration evaluation

Accurate image registration is essential for the kind of image analysis we propose in this paper and is a prerequisite for the remainder of the steps in our process. An alignment error of more than a few pixels may introduce noise that can lead to misclassification down the line. In prior work, we showed how image superposition can be used to visually verify the correctness of ballot image registration [5]. One question that remained open was

how to quantify the accuracy of the registration procedure. For that, we developed the following technique.

We evaluate the accuracy of our alignment by measuring the distribution of locations of *empty* voting targets across all ballots:

1. First, we use our set of blank template ballot images (which we know to be registered correctly), and find all the possible regions in which a voting target might occur. We will use this information to group together voting targets appearing near the same region.

2. Next, for every individual ballot image, we use *template matching* (see Section 3.1.1) to locate all instances of empty voting targets occurring in each image. Note, we do not attempt to locate filled voting targets, as they may vary significantly by voter.

3. Last, for every empty voting target found, we associate it with the nearest voting target in our template ballot images (from step one). This lets us determine a distribution of voting target locations per region.

Empty voting targets are useful for measuring alignment because:

- Template matching works accurately and efficiently on the distinct and identical patterns typified by empty voting targets.
- Voting target locations are of critical importance to optical scan ballots, so it makes sense to evaluate registration at those regions.
- Voting targets are generally spread out across the page, letting us measure registration on many parts of the ballot.

We evaluated our registration method on our dataset of Leon County ballots. For each ballot, we computed the average Euclidean pixel distance of all the detected empty voting targets to their corresponding locations in the template images. For most ballots the average error was 3 pixels or less (see Figure 8). In contrast, the unregistered images showed an average error of 9.480 pixels.

Our method has a couple limitations. First, our method relies on template matching to locate empty voting targets, which itself may introduce some slight noise. Second, if a ballot is very poorly registered, an empty voting target in the image may by chance associate with an incorrect voting target in the template. This would lead us to underestimate the alignment error. However, from our experience the scanned ballot images were initially closely aligned enough that we did not run into this problem. Despite these limitations, this method proved to be a useful measure of registration accuracy for this data set.

Cause	Write-ins	Ballots
Registration error	17	9
Grouping error	2	1
Scanning error	2	2
Total	21	12

Table 1: False positives of our algorithm. The second column shows the number of write-in regions affected, and the third column shows the number of ballots affected. Sometimes errors affect multiple write-in regions on a single ballot.

4.2 Evaluating ballot grouping

We evaluated our ballot grouping algorithm by forming one overlay image [5] per group from all the ballots in that group. By carefully inspecting each overlay image, we confirmed that all ballots were grouped properly. Because there was a relatively small number of groups, this was a fairly quick task to complete.

4.3 Evaluating write-in detection

We would like to know how well our method detects write-ins. Generally, two things can go wrong: we could detect a write-in mark where none actually exists (a false positive); or we could fail to detect a write-in mark when one actually does exist (a false negative).

4.3.1 False positives

A false positive in our case is a write-in region that has no *voter-made* mark, but was still classified as a write-in. We do not try to make judgement calls about the nature of the mark—rather, we just want to know if *something* is marked in there or not. Therefore, even voter marks such as lines, scribbles, or stray marks should be detected by our system, and would not be considered false positives. Given this definition, we found a total of 21 false positive *write-in regions*, across 12 ballots, which accounts for 0.005% of total write-in regions (see Table 1). The sources for these errors fell into three categories, examples of which are shown in Figure 9:

1. Registration errors. These are caused by a failure in the registration step that left the registered ballot misaligned. As a result, the write-in regions capture some extraneous ink from the printed ballot.

2. Grouping errors. These are caused by putting the ballot into the wrong group, so its write-in regions are associated with the wrong location. This makes us look at an incorrect location on the ballot for write-in marks, so we may mistake printed text for

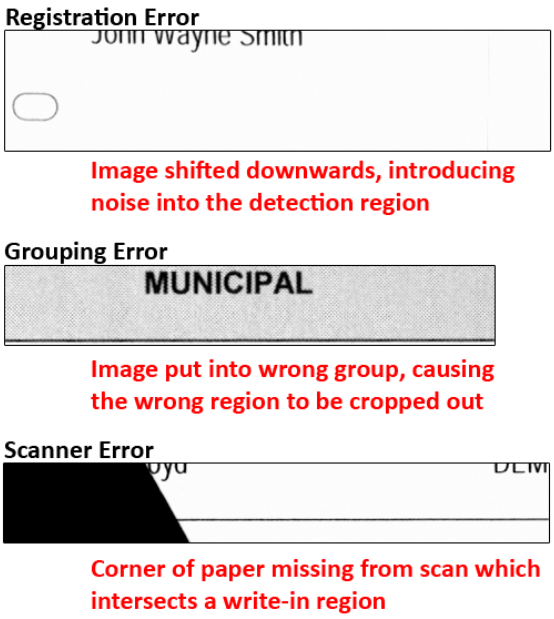

Figure 9: Examples of false positives: images classified as having a write-in mark when in fact there was no *voter-made mark* in the region.

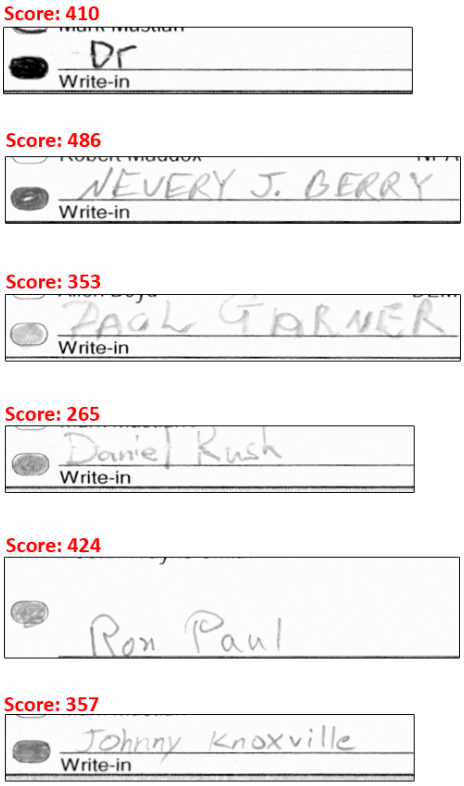

Figure 10: Examples of false negatives (write-in marks not detected by our algorithm). The score is the number of black pixels in the region. Our algorithm classified any image with a score above 500 as a write-in mark.

voter-made write-in marks. This kind of failure is caused by OCR errors during grouping.

3. Scanning errors. These are due to a fold in the scanned ballot, which just happens to intersect a write-in region[6]. Folds like this appear black in the scanned image and are mistaken for voter marks.

The first two cases are errors in our pre-processing steps, and show that they need to be improved in order to prevent these occurrences. Scanning errors can only be prevented through changes to the process for scanning ballots.

We did not have an automated way to detect false positives. Instead, we manually searched for false positives by individually and carefully inspecting each ballot flagged as having a write-in. This did not take a long time — around 30 minutes to inspect about 1600 relatively small images — but this could be an expensive task for larger data sets or ones with a higher prevalence of write-ins. This is a fundamentally hard problem because it can be ambiguous whether a mark counts as a "write-in" or not.

[6]These folds or tears occur in several ballots (which is one reason that the registration algorithm takes several steps), but this is the only one that happens to interfere with a write-in region—the rest are harmless for our analysis.

4.3.2 False negatives

False negatives are write-in marks that were missed by our analysis. These include marks that were small or light enough to pass underneath the detection threshold. We found a total of 6 write-in regions across 6 ballots that were false negatives, accounting for 0.002% of the total number of write-in regions.

We detected these failure by superimposing images of the write-in regions classified as not containing a voter mark, to create an overlay image [5]. We visually inspected the resulting overlay image for marks. If the location of the write-in regions are detected accurately and there is a mark in any of these regions, it will also be in the overlay. However, this method does not reliably detect false negatives due to registration or grouping error. Our analysis suggests that the detection threshold is important, and an alternative scheme is described in the future work section (Section 7).

Figure 11: We used overlay images to verify that all voting targets were classified correctly. Above is an example showing the result of overlaying 100 voting targets.

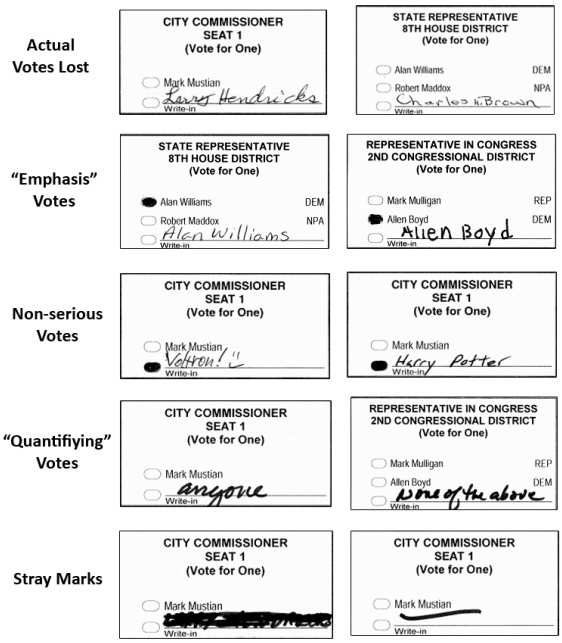

Figure 12: Examples of several kinds of write-ins.

4.4 Evaluating the classification of voting targets

Lastly, we verified that we categorized the voting targets for each write-in region correctly. We crop all voting targets and classify them as filled or unfilled. We generate an overlay image superimposing all targets classified as unfilled, and another overlay image superimposing all targets classified as filled (see Figure 11). This confirms that we have classified all voting targets correctly.

5 Analysis of Leon County Election

Next, we use our techniques to examine how voters actually filled out write-ins during a recent real election. We obtained scans of the ballots from an election held in Leon County, analyzed them using our approach (from Section 3) to find all write-in marks made by voters in that election, and then analyzed these write-in marks using Amazon's Mechanical Turk (MTurk).

	Write-in region	
	marked	unmarked
Target filled	833	78
Target not filled	785	366,981
Total	1,618	367,059

Table 2: Summary of write-in regions, according to whether the voter marked the write-in region or not and whether the corresponding voting target is filled or not.

Question	Number of errors
Question 1	85
Question 2	49
Question 3	63
Total	197

Table 3: The number of cases where the MTurk majority answer was incorrect.

Data Set. The data set consists of scans of the ballots from the Leon County, November 4, 2008 general election. There were a total of 124,167 double-sided ballots, each 8.5×14 inch in size and formatted for a Premier (Diebold) optical scan system. The ballots were independently scanned by the Clear Ballot Group and given to us as 200 DPI, 8-bit grayscale uncompressed Bitmap Image (BMP) files. Clear Ballot also provided us 19 blank templates in PDF format, for the blank ballots.

Analysis. We applied our automated write-in detection methods to this data set. Then, we identified all false positives and false negatives (as described in Section 4.3) and corrected them manually. For the purposes of this analysis, we count any marginal marks (marks inside the voting target) as being *filled*, meaning that only voting targets free of noise will be counted as unfilled.[7]

Table 2 shows the results of this analysis. We show the number of write-in regions, not ballots, as there are usually multiple write-in regions per ballot. These 368,677 write-in regions are across all 124,167 ballots, and the 1,618 write-in marks are across 1,280 ballots.

Amazon Mechanical Turk. We used Amazon's Mechanical Turk (MTurk) to carry out further analysis of the Leon County write-in marks. MTurk allows a 'requester' to submit tasks, which are then completed by

[7] In an actual election, these marginal marks may be ambiguous and need to be reviewed by an election official.

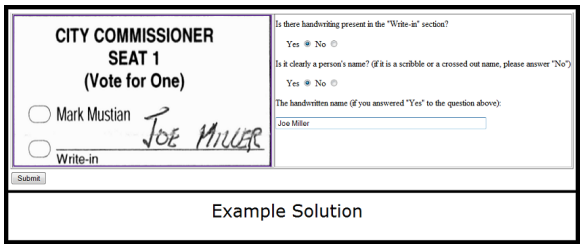

Figure 13: We asked MTurk workers: "Is there handwriting present in the "Write-in" section? Is it clearly a person's name? (if it's a scribble or a crossed-out name, please answer "no") The handwritten name (if you answered "yes" to the previous question)."

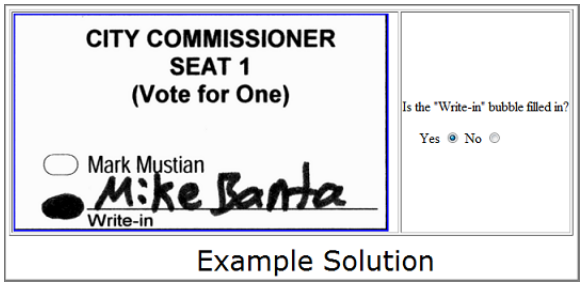

Figure 14: We asked MTurk workers: "Is the "Write-in" bubble filled in?"

online workers for a small monetary reward. We submitted three tasks. In each task, a write-in contest was presented to the worker, along with the questions shown in Figures 13, 14, 15. We used MTurk to classify voting targets as filled or unfilled, detect handwriting in write-in regions, and recover the name written in (if any). Our primary motivation for using MTurk was to identify the name written in, which would be hard to automate.

In order to establish confidence in the workers' results, we had three workers answer each question, and we took the majority answer in our analysis. Despite these measures, there were still some errors. To address the remaining errors, we verified the MTurk results by manually inspecting and correcting every majority answer. As Table 3 shows, there were 197 errors where the majority answer was incorrect (out of 14,670 tasks), which is an error rate of 1.3%.

Common Categories of Write-In Votes. We observed several kinds of write-in votes appeared frequently:

1. Emphasis Votes. An emphasis vote is a write-in contest in which the voter has both filled in a candidate's voting target and wrote down the candidate's name in the write-in region. This behavior is presumably a way for the voter to emphasize his/her voting intention (hence, an "emphasis" vote).

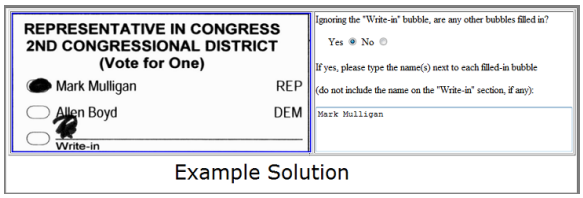

Figure 15: We asked MTurk workers: "Ignoring the "Write-in" bubble, are any other bubble's filled in? If yes, please type the name(s) next to each filled-in bubble (do not include the name on the "write-in" section, if any)."

2. Conflict Votes. A conflict vote is a write-in contest in which the voter filled in one candidate's voting bubble, but wrote down a different candidate's name in the write-in region.

3. "Non-serious" Names. A "non-serious" name occurs in a write-in region when the name written down clearly refers to a fictional character. A popular example is Mickey Mouse, of Disney-cartoon fame. We detected these by manually inspecting all write-in marks with handwriting and flagging marks with non-serious names. We do not consider names of celebrities as "non-serious" (such as 'Elvis' or 'Stephen Colbert').

5.1 Results

The MTurk results enabled us to answer many questions, such as: How many voters actually filled in the write-in bubble? How many did not? How many bubbled in for other candidates and also made a mark in the write-in region? More often than one might expect, we found that voters do not behave in the way that designers of optical scan systems might assume.

Out of the 1,618 cases of handwriting in the write-in region, 833 had the corresponding voting target filled in, and alarmingly 785 left the voting target unfilled. In other words, nearly half (49%) of voters who wrote something in the write-in region did not fill in the bubble. This highlights the importance of not relying on the presence of a filled voting target to detect write-in marks.

There were many cases where the voter filled in a certain candidate's bubble, and also wrote that same candidate's name in the write-in region (an "emphasis" vote). This might be simply because the voter wanted to reiterate which candidate they prefer, or because the voter was unclear on how they were supposed to mark the ballot. In our data set, 453 out of 1,618 (28%) write-in marks were emphasis votes. Of these, there were two types of emphasis votes: ones in which the voter also filled in the write-in bubble, and ones in which the voter did not fill

"Non-serious" name	Count
Mickey Mouse	17
Jesus	9
Donald Duck	3
Daffy Duck	3
God	3
Babe the Pig	1
Bingo the Dog	1
Bubba the Frog	1
Dr "V"	1
Me	1
Your Mom	1
Tasmanian Devil	1
Timmy the Turnip	1
Dr. Horrible	1
Ronald McDonald	1
Captain America	1
Santa Clause	1
Slim Pickens	1
Doe	1
Voltron	1
Harry Potter	1
Wedge Antilles	1
Ham Sandwich	1
Dr	1

Table 4: "Non-serious" names written in by voters, and the number of times they were written in.

in the write-in bubble. There were 3 instances of the former, and 450 instances of the latter. It is important to consider this type of voter behavior when designing our write-in detection system, since emphasis votes are not the same as other write-ins.

We also found 17 conflict votes, in which the voter filled in a candidate's bubble, but also wrote down a different candidate's name in the write-in region. The write-in bubble was not filled in any of these 17 votes. At the top of each of these contests, the instructions state "Vote for one", so it is not immediately clear if this is a valid vote. This case is intriguing, because current scanners will count it as a vote for the candidate who is bubbled in, and ignore the un-bubbled write-in. However, one could plausibly argue that votes like these should be deemed *overvotes* (an invalid vote in which the voter picked more than one candidate). In fact, the State of Minnesota takes this stance [11], and resolved the famous "Lizard People" ballot accordingly.[8]

In addition, we found non-serious votes. We classify a write-in vote as "non-serious" when it is clear that it refers to a fictional person (e.g, "Mickey Mouse"). We also classified some obvious cases where the voter is trying to be humorous (writing "me", for instance) as "non-serious". However, if it does sound like a real person's name, or it is unclear, we classify it as "serious". While non-serious votes don't have a significant impact on election results, these types of votes do occur: we found 54 non-serious votes (3% of the 1,618 write-ins). See Table 4 for examples of non-serious names written in.

[8]The "Lizard People" ballot was a highly-publicized ballot where a voter voted for Al Franklin, but wrote down "Lizard People" in the write-in region. During the manual recount, the state decided that the "Lizard People" ballot (and other similar ballots) are overvotes [2].

"Non-serious" name	Count
none, n/a, neither	37
anyone, anyone else	10
yes	3

Table 5: "Quantifying" names written in by voters, and the number of times they appeared.

Finally, we found 35 write-in marks (2%) containing quantifiers such as "none," "neither," or "anyone else" (see Table 5).

Lost Votes. In current optical-scan systems, a write-in vote is counted only if the write-in bubble is filled in. A question that we'd like to answer is: how many write-in votes are lost (i.e., not counted) in current systems? In particular, we count the number of votes with a legible, handwritten name in the write-in region, where the corresponding bubble is not filled in. We exclude emphasis votes, conflict-votes, and overvotes from this count.

In the Leon County election, there were 266 votes that would be lost under this definition. This is a surprisingly large number: 16% of all write-in marks would not be counted by the current optical-scan systems. Filtering out all "non-serious" names leave 252 votes (16% of write-in marks). In Leon County, votes for write-in candidates are only counted for candidates who file forms in advance ("qualified candidates"). We did not analyze how many of these write-ins were for qualified candidates; in some jurisdictions, there is no need for write-in candidates to register in advance. Overall, these numbers imply that the current method of detecting write-in votes (by detecting a filled-in voting target) misses many votes.

Lost votes could be avoided by better technology and revised policies. One simple solution would be to not require the voter to fill in the write-in bubble. Several jurisdictions allow this, including the City of Massachusetts [13] and the State of Minnesota [11]. As we've shown, next-generation optical scan systems could automatically and reliably detect write-in votes without having to rely on a write-in bubble being filled in.

Cost. Our use of MTurk was relatively inexpensive. We paid workers \$0.04, \$0.03, \$0.04 per task for the questions shown in Figures 13, 14, 15, respectively. After submitting 1630 write-in contests to MTurk, the total cost was \$612.00 (including tax). While MTurk was useful to expedite our analysis, it is not essential, and we anticipate that election officials might prefer to take responsibility for interpreting write-in marks themselves.

6 Related work

Prior work has established the value of using scanned images for auditing and investigating elections. The Humboldt County Election Transparency Project is an important pioneer of this approach, and for several elections they have rescanned all ballots cast in that county with a commercial off-the-shelf document scanner [7]. Other open-source software projects for analyzing ballot images include Mitch Trachtenberg's BallotBrowser [7] and VotoScope [8]. We were inspired and motivated by VotoScope and BallotBrowser, and we see our work on automating write-in detection as complementary.

The PERFECT project at Lehigh University pioneered the application of document analysis techniques to automating the interpretation of voter marks on optical scan ballots [16, 14, 9], though they do not look at detecting write-ins.

We were also influenced by OpenScan [15], which analyzes a video of ballots to extract and interpret ballot images. OpenScan uses the SIFT algorithm for alignment [10]. However, it is not clear how to apply SIFT in our setting. OpenScan's implementation assumes that all ballots have the same ballot style. Under that assumption, one can use SIFT to align each ballot to a template for that ballot style. However, when there are multiple ballot styles, as in our data sets, it is not clear how to perform alignment: registration using SIFT requires knowledge of the appropriate template and thus knowledge of the ballot style before registration, and it is not easy to infer the ballot style of unregistered ballots.

7 Future work

As shown by the examples of false positives and false negatives, there is room for improvement to the write-in detection algorithm. We used hard-coded thresholds on the number of black pixels to determine whether a write-in mark was present, and also when converting gray-scale images into black and white. However, hard-coded thresholds are brittle; it would be preferable to automatically infer the appropriate threshold from the environment. One idea to set the threshold would be to use an expectation-maximization algorithm[9] on the write-in regions, to cluster them into one cluster with empty write-in regions and a second cluster with marks.

Several aspects of our implementation were tailored to Premier (Diebold) ballots. Several steps in the process make certain assumptions about the ballot that may not be valid for other ballot styles (for instance, the presence of registration hashmarks along the border). One promising direction might be to use algorithms like SIFT[10] in the registration and grouping stages, to build an algorithm that applies to many different ballot styles.

8 Conclusion

This paper develops techniques to detect write-in marks that could otherwise be missed by optical scan systems. Our approach involves processing scanned images of ballots to detect whether or not they contain marks in the write-in regions. We demonstrated the feasibility of our approach on a large data set from Leon County. The analysis from this data set showed surprising results, including 252 write-in marks that would not be detected by existing optical scan systems. This suggests that our methods may be useful for improving future optical scan systems.

9 Acknowledgments

We thank the Clear Ballot Group, Larry Moore, and Ion Sancho, Supervisor of Elections in Leon County, for sharing a large collection of scanned ballot images with us. This research would not have been possible without their generous assistance. We thank Kai Wang and the anonymous reviewers for helpful comments. This work was supported by National Science Foundation grant CNS-0524745. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Write in candidates, 2009. http://www.citizenmodel.net/?p=184.

[2] BOB COLLINS. The Lizard People ballot solved. Minnesota Public Radio, 2008. `http://minnesota.publicradio.org/collections/special/columns/news_cut/archive/2008/11/the_lizard_people_ballot_solve.shtml`.

[9] Expectation-maximization (EM) is one approach to clustering. It may be especially applicable in our setting: the distribution of number of black pixels in an empty region may be Gaussian, and similarly for the number in a marked region, so the task is to separate a mixture of two Gaussians—something EM is especially well-suited for.

[10] SIFT (Scale Invariant Feature Transform) is an algorithm that identifies robust features in images, regardless of how they are rotated or translated [10].

[3] BOHRER, B. Murkowski becomes 1st write-in senator since '54. The Boston Globe, 2010. http://www.boston.com/news/politics/articles/2010/11/17/murkowski_emerges_as_winner_in_alaska_senate_race/.

[4] CHEN, J.-L., AND LEE, H.-J. An efficient algorithm for form structure extraction using strip projection. *Pattern Recognition* (September 1998).

[5] CORDERO, A., JI, T., TSAI, A., MOWERY, K., AND WAGNER, D. Efficient user-guided ballot image verification. In *Proceedings of EVT/WOTE 2010*.

[6] FOUNDATION, C. M. San Diego mayoral election revives memories of the 2000 Florida count, 2004. http://www.citymayors.com/politics/usmayoral_elections.html.

[7] Humboldt County Election Transparency Project. http://humtp.com.

[8] HURSTI, H. Votoscope software, October 2005. http://vote.nist.gov/comment_harri_hursti.pdf.

[9] LOPRESTI, D., NAGY, G., AND SMITH, E. B. Document Analysis Issues in Reading Optical Scan Ballots. In *DAS '10: Proceedings of the 8th IAPR International Workshop on Document Analysis Systems* (2010).

[10] LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2 (2004), 91–110.

[11] MINNESOTA OFFICE OF THE REVISOR OF STATUTES. 2010 Minnesota Statutes - 204C.22 Determining Voter's Intent , 2010. https://www.revisor.mn.gov/statutes/?id=204c.22.

[12] OTSU, N. A threshold selection method from gray-level histograms. In *IEEE Transactions on Systems, Man and Cybernetics, Vol. 9, No. 1.* (1979), pp. 62–66.

[13] SECRETARY OF THE COMMONWEALTH, ELECTIONS DIVISION. Running for Office as a Sticker of Write-in Candidate, Unknown. http://www.sec.state.ma.us/ele/elestkr/stkridx.htm.

[14] SMITH, E. H. B., LOPRESTI, D., AND NAGY, G. Ballot Mark Detection. In *19th International Conference on Pattern Recognition* (2008).

[15] WANG, K., RESCORLA, E., SHACHAM, H., AND BELONGIE, S. OpenScan: a fully transparent optical scan voting system. In *Proceedings of EVT/WOTE 2010*.

[16] XIU, P., LOPRESTI, D., BAIRD, H., NAGY, G., AND SMITH, E. B. Style-Based Ballot Mark Recognition. In *10th International Conference on Document Analysis and Recognition* (2009).

A Generalization to other elections

Our experiments used ballots from a single county. To assess the applicability of our techniques to other ballot styles and formats, we examined a subset of the sample state and local ballots collected by the National Institute of Standards and Technology (NIST).[11] Some of these examples came from very similar versions of the same ballot (for example, Republican and Democratic versions of a ballot for a primary election). For analysis purposes, we combined these cases into a single entry so that it was only counted once. We also excluded ballots that did not appear to be optical scan ballots, such as touchscreen, punch card, and lever-machine ballots.

There are four main optical scan ballot vendors in the United States: Diebold/Premier Election Solutions, Election Systems & Software (ES&S), Sequoia Voting Systems, and Hart InterCivic. The NIST collection includes examples of ballots from all four vendors. Each vendor's ballots tend to follow a uniform format and layout, so we analyzed examples to see if our methods for registration/alignment and automated write-in detection could be extended to these ballot formats.

- **Premier/Diebold:** Premier ballots have uniform hashmarks spaced evenly along the top and down the left and right sides. The bottom border has unevenly spaced hashmarks that may indicate information about the ballot style or page. Each contest is typically enclosed in a box, and write-in lines are usually labeled and lined up with the voting target.

- **ES&S:** ES&S ballots usually have many points that could be used for registration. There are three hashmarks spaced across the top and bottom of each ballot, a uniform column of hashmarks on one side of the ballot, and a few other different types of hashmarks unevenly spread along the border. The central area of the ballot that contains all the contests is enclosed in a rectangle, with the corners marked by an "L"-shaped indicator. Each contest is typically enclosed in a box. The write-in lines sometimes connect all the way across to the side, which can make line detection a bit more difficult. Also, while most ES&S ballots with write-ins had the write-in lines labeled, some of them were not labeled.

- **Sequoia:** Sequoia ballots typically have a solid bar spanning both the top and bottom of the ballot, as well as regularly spaced hashmarks on the left and right sides. In addition, there are hashmark formations at the top and a few hashmarks at the bottom of the ballot that seem to be used for identifying the ballot format. Each candidate within a contest is separated with a horizontal line, with a write-in label or blank space left for write-ins. Write-in lines are not differentiated from those used for listed candidates. The write-in space is, however, matched up with the voting target.

- **Hart:** Hart ballots typically do not have hashmarks along the border. However, there are barcodes in three corners. Also, the entire contents of the ballot are bound in a rectangle, the corners of which could be used for alignment. Hart ballots have contests

[11] NIST curates a collection of scanned paper ballots used in elections between 1998 and 2006. It is available at http://www.nist.gov/itl/vote/sample-ballots.cfm

Vendor	Number of ballots
Premier/Diebold	29
ES&S	26
Hart	4
Sequoia	25
Unknown/Other	20

Table 6: Classification of sample ballots in the NIST collection by vendor.

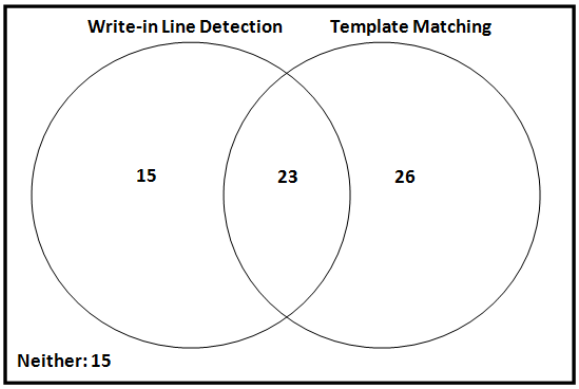

Figure 16: Applicability of two write-in region detection methods on the sample ballots in the NIST collection.

that are enclosed in rectangles, with write-in lines within. The write-in lines are not always labeled.

Of the 104 ballots we analyzed, 84 appeared to be associated with one of the four main vendors. The breakdown is shown in Table 6. Out of the 104 ballots, there were only 3 where we could not identify any obvious hashmarks or other indicators that could be used for alignment. 25 of the 104 ballots did not contain any write-in regions so were excluded from our analysis below.

We examined what proportion of the remaining 79 sample ballots our write-in region detection methods could be applied to. If the ballot contains write-in lines within a contest enclosed in a box, the form extraction method is applicable. If write-in regions are indicated by a labelled write-in line, the alternate template matching method is applicable. We found that it should be possible to identify the write-in regions for 64 out of the 79 ballots using one of these methods (see Figure 16).

There were 15 ballots that cannot be handled by the form extraction or template matching write-in detection techniques. These ballots typically left an unlabeled blank space next to a voting target for the write-in (see Figure 17 for an example). In order to identify write-in locations for these ballots, other techniques could be used. For example, we could find voting targets that are

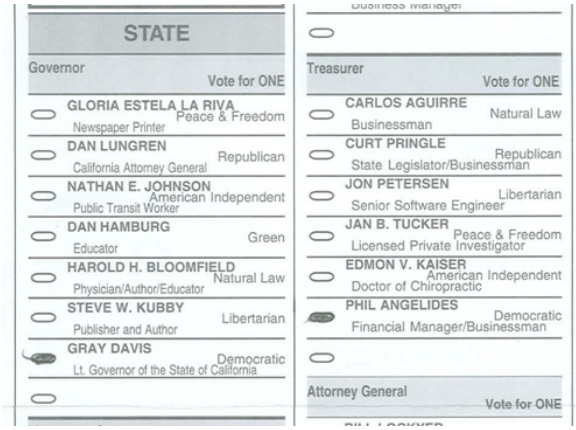

Figure 17: An example of a ballot where our write-in region detection techniques would be insufficient. On this ballot, write-in regions are not differentiated or labeled.

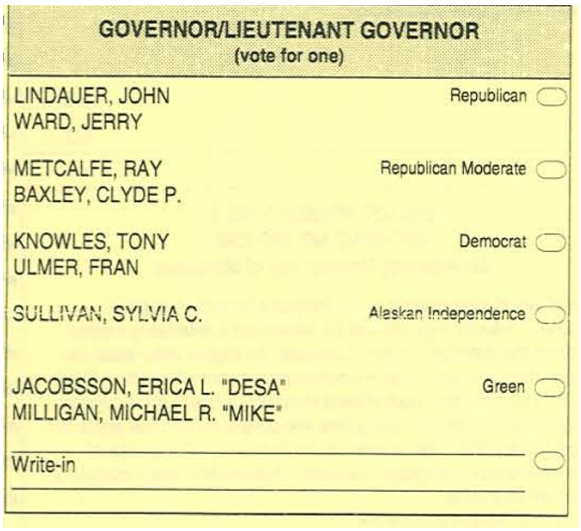

Figure 18: An example of a contest where a single voting target and write-in label correspond to two write-in lines. This can be problematic when trying to automatically match the label or target with a write-in.

situated next to a certain amount of blank space. A default method if none of these work is to ask the operator to specify the write-in locations on each ballot template.

We also studied the ease of associating each write-in region with its corresponding voting target. 74 of the 79 ballots had the voting target lined up with the write-in region/line, so associating a write-in region with its corresponding voting target should not be difficult for most ballots. However, see Figure 18 for a difficult case.