Operator-Assisted Tabulation of Optical Scan Ballots

Kai Wang[*] Eric Kim[†] Nicholas Carlini[†] Ivan Motyashov[†] Daniel Nguyen[†]
David Wagner[†]
[*]*University of California, San Diego* [†]*University of California, Berkeley*

Abstract

We present OpenCount: a system that tabulates scanned ballots from an election by combining computer vision algorithms with focused operator assistance. OpenCount is designed to support risk-limiting audits and to be scalable to large elections, robust to conditions encountered using typical scanner hardware, and general to a wide class of ballot types—all without the need for integration with any vendor systems. To achieve these goals, we introduce a novel operator-in-the-loop computer vision pipeline for automatically processing scanned ballots while allowing the operator to intervene in a simple, intuitive manner. We evaluate our system on data collected from five risk-limiting audit pilots conducted in California in 2011.

1 Introduction

In this paper, we develop techniques to count optical scan ballots, based only upon scanned images of those ballots.

There are several reasons why it might be useful to be able to count the ballots using a system that is independent of the official, certified voting system. Some have suggested that, to gain confidence in the election results, all ballots should be scanned and the images should be published, so that anyone interested can count the ballots on their own [7] [16, § 3.3]. We build a tool that can be used to perform the counting process, and thus could be used for this purpose. Alternatively, our tool could be used by election officials to check the accuracy of official vote tallies before they are certified.

Perhaps most importantly, our tool can play an important role in supporting risk-limiting audits of elections [12]. Risk-limiting audits require the ability to export cast vote records (CVRs) and vote totals from the voting system, separated by precinct and other criteria. Unfortunately, many current voting systems cannot report vote tallies in this fashion, and provide no way to export cast vote records [20], posing a barrier to adoption of risk-limiting audits. Our tool provides a way to generate CVRs and makes it easy to generate vote tallies for arbitrary batch sizes, enabling risk-limiting audits to be more efficient. Thus, our tool eliminates a key barrier to adoption of risk-limiting auditing [6, § 4].

Finally, our tool can support ballot-level audits [4, 3]. For example, if the order of ballots is maintained after scanning, our tool enables ballot-level audits using the order of ballots [3, § 2]. Thus, our work eliminates a barrier to ballot-level auditing that was identified in prior work [19, § 6]. See Section 2.4 for further discussion.

We are not the first to articulate this vision. The Humboldt Election Transparency Project initially proposed parallel tabulation [7], and in support of this goal, the ground-breaking TEVS system [21] (the successor to Ballot Browser) was built to automate the process of tabulating an election from scanned ballot images. Our work was motivated by an attempt to use TEVS during several risk-limiting audit pilots in California in 2011. We found that at this point in its development, TEVS required some adaptations to the code and/or (due to its reliance upon OCR) manual post-processing of its output for each election [17, 18]. We design techniques to address these problems and build a tool, OpenCount, that provides software support for risk-limiting audits. Our early prototype was used in California pilots in 2011, and we plan to use the refined tool to assist with risk-limiting audits in 6 more counties in 2012.

This paper makes the following contributions:

- We design techniques to recognize and tabulate votes, given only scanned ballot images. Our methods require no support from the official voting system and do not rely upon election definition files.
- We implement these techniques and show that they scale to large elections and are robust enough to handle the cases that arise in practice.

- We demonstrate our techniques and our tool on five elections held in five California counties in 2011 and show that OpenCount is at least as accurate as currently deployed voting systems for this dataset.

2 Problem Statement

2.1 Terminology

We define some terms we will use throughout the paper. A *blank ballot* is a paper ballot as it was originally printed, with no voter marks on it. A *voted ballot* is the ballot after a voter has marked his/her ballot and cast it.

Each ballot contains a set of *contests*. A *contest* includes a list of *candidates*, with one voting target per candidate. A *voting target* is an empty oval, broken arrow, or other location on the ballot where the voter should mark her ballot, if she wants to indicate a vote for the associated candidate. A *cast vote record* (CVR) is a record of all selections made by the voter on a single voted ballot.

The *ballot style* is the set of contests found on the ballot as well as the visual organization of these contests on the ballot. For example, an English-language ballot may have the same set of contests as a Spanish-language ballot, but because their text is different, we consider them as two different ballot styles. Ballots may also contain a precinct number or a tally group for accumulation (e.g., absentee vs. polling-place). We do not distinguish between blank ballots whose content is visually identical.

The *grouping patch* is a region on the ballot that uniquely determines the ballot style and blank ballot. For example, the grouping patch might be a location on the ballot where the precinct number is printed. In some elections, we use multiple grouping patches: for example, one grouping patch might encode the precinct, another might encode the language, and (in a primary election) a third might encode the party affiliation.

2.2 Goal

Our goal is to count a set of paper ballots, given just two kinds of information: scans of all of the voted ballots, and scans of blank ballots. In particular, we use only the human-readable information that is found on the face of the ballots—the same information that voters see. We want to compute vote tallies and CVRs, given this set of scans.

Deployed voting systems typically use election definition files, generated by the ballot layout definition system, to determine where to look for voter marks and associate them with particular candidates. We deliberately avoid relying upon election definition files, both because the election definition file can be faulty (causing deployed systems to mis-count votes) and because we want to independently count the ballots without any dependencies on other voting software.

We assume that someone has scanned all of the voted ballots, and has scanned one instance of each visually-unique blank ballot, using a standard document scanner. The logistical details of the scanning process are out of scope for this paper.

2.3 Technical Challenges

There are several non-trivial challenges in building a system to count the ballots. First, our system must be *scalable*: it must be able to process all of the data associated with large elections, in a reasonable amount of time (hours or days, not weeks). This imposes severe restrictions on the kinds of algorithms that we can use: a system that takes 1 second of computation per ballot may be acceptable, but one that takes 100 seconds per ballot would be unworkable (for an election with 100,000 ballots, processing would take about 16 weeks).

Second, and closely related, the system must be *robust*. We have found that, when dealing with large numbers of ballots, unusual cases are common. There will always be some ballots that are scanned imperfectly – effects from transformation (rotation, skew), distortion (scanner noise, specks of dust), and illumination differences may be prominent. An extreme (yet observed) case includes the physical destruction of the paper ballot itself (tears, creases, and other damage). Our system must be able to handle all of these cases gracefully. A method that works on 99.9% of ballots is too fragile; in an election with 100,000 ballots, such a method would fail on 100 ballots. The robustness requirements rule out many techniques that may at first glance appear promising.

Third, the system should be *general*: it should not rely upon hard-coded assumptions that are specific to ballots from a particular voting vendor. For instance, it should not rely upon decoding barcodes or timing marks that may be found on the ballot, as those are vendor-specific. We do not want the system to be limited to handling ballots from a particular set of voting system vendors.

Lastly, the system must be *self-contained*: it must not require any additional data sources beyond the scanned images. This poses algorithmic challenges. For instance, when processing a voted ballot, we need to identify the corresponding blank ballot, so that we can identify the set of contests on the ballot, find the location of all voting targets, and then check for marks at those locations. However, when an election might have 100 or 1000 different types of blank ballots, it is not clear how to quickly map each voted ballot to its corresponding blank ballot, within the time budget. We devise algorithms to solve this problem efficiently.

2.4 Applications to Auditing

OpenCount could be used to support election auditing in three ways, elaborated below.

Parallel audits. If election officials scanned all paper ballots and made the scans available to the public, OpenCount could be used to count those scanned ballots. Because OpenCount is open source, any candidate, observer, or interested member of the public could use OpenCount to interpret and count the ballots. However, this procedure assumes that OpenCount is correct and that the published scans accurately reflect the paper ballots actually cast in the election [16], which limits the level of confidence attainable through such an approach.

Batch-level risk-limiting audits. OpenCount can be used to support risk-limiting audits, performed at a batch granularity. Generally speaking, the smaller the batch, the more efficient the audit is, so there are good reasons to want to use batches that are as small as possible. The audit procedure needs vote tallies for each batch. Unfortunately, many currently deployed voting systems cannot produce vote tallies for batches that are smaller than a precinct. This is a key barrier to broader use of small-batch risk-limiting audits [6, § 4].

OpenCount meets this need: it supports counting arbitrarily-defined batches. Batches can be defined by any combination of attributes printed on the ballot (e.g., precinct number, mode, ballot type, party affiliation, language), as well as any structure inherent in how the ballots are scanned (e.g., if ballots are scanned in batches, OpenCount can produce corresponding batch-level tallies). Thus, OpenCount satisfies a prerequisite for adoption of small-batch risk-limiting audits. Election officials can scan their ballots in batches using an ordinary document scanner, process the ballots using OpenCount to obtain batch-level tallies, and then verify those tallies using a batch-level risk-limiting audit.

OpenCount also solves a related challenge associated with auditing of absentee ballots. Some jurisdictions do not sort their absentee ballots by precinct; instead, they scan them in batches. The natural way to audit such ballots is using those same batches. Unfortunately, many deployed voting systems cannot export vote tallies for each batch, which makes it difficult to audit the absentee ballots in this situation. OpenCount solves this problem.

Ballot-level risk-limiting audits. Ballot-level audits have the potential to be especially efficient, because the batch size is a single ballot. However, most deployed voting systems do not provide the information that is needed to perform ballot-level audits [3, § 2].

Ballot-level audits require a cast vote record (CVR) for each ballot and some way to uniquely associate each paper ballot to its CVR. Most deployed voting systems cannot provide this information. OpenCount meets this need. Election officials can scan the ballots, maintain them in the order they were scanned, and process the scanned images using OpenCount. OpenCount can produce a CVR for each ballot, in the same order as the paper ballots. The ordering of the ballots provides a way to link each paper ballot to its CVR. Then, given these CVRs, one could apply SOBA [3] or another ballot-level risk-limiting audit method.

Alternatively, if ballots are stamped with a unique number while they are scanned (a feature that is supported by many commercial scanners), OpenCount can be used to enable ballot-level auditing without any support from or changes to the official certified voting system [4, 3].

Note that using OpenCount to facilitate risk-limiting audits does not require trusting OpenCount; the audit process verifies the accuracy of OpenCount's results. This is a transitive audit [12]: to address limitations in the official voting system, we count the ballots a second time using OpenCount, check that OpenCount reports the same winner as the official voting system, and then use a risk-limiting audit to verify the accuracy of OpenCount's results. The benefit of using OpenCount is that it can produce the information needed by risk-limiting audit procedures—something that is beyond the capabilities of many currently deployed voting systems.

3 Overview of Approach

3.1 Principles

Our design is influenced by several principles. First and foremost, OpenCount uses a hybrid of computer vision algorithms together with human assistance. Neither alone is sufficient; human classification does not scale, but computer vision cannot ensure that ballots are interpreted as a voter or election official would. Thus, we use vision-based algorithms for scalability, and we rely upon assistance from a human operator to guide the algorithms in edge cases or situations where voter intent is ambiguous. The technical challenge is to identify how to direct the computation so the operator's workload is minimized. In short, OpenCount can be thought of as a hybrid man/machine system; one of the novel contributions of OpenCount lies in the specific details of which tasks are performed by algorithms and which are performed by the operator.

Second, to ensure accuracy, OpenCount emphasizes verification. Because computer vision algorithms can make mistakes, we ensure that every computation performed has a simple operator-assisted verification step.

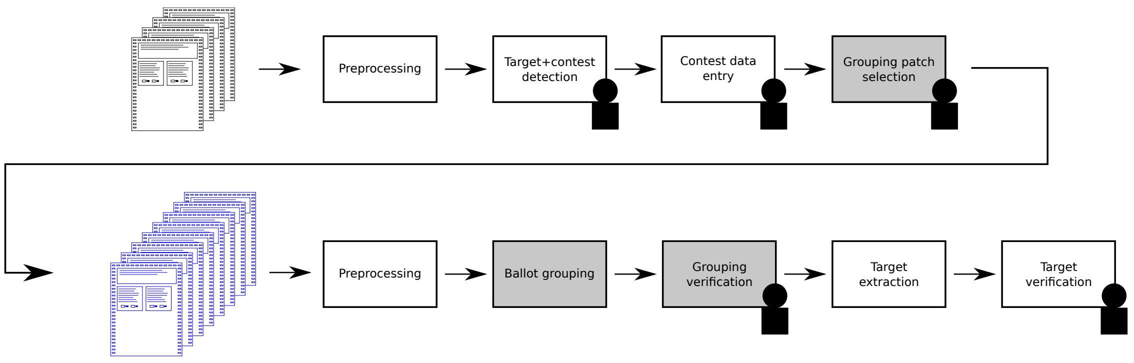


Figure 1: The OpenCount pipeline. Blank ballots are shown in black, and voted ballots in blue. Steps requiring operator assistance are labeled with the person icon; all others are automated. The steps shown in grey are only required when the election has multiple blank ballots. Section 3.2 describes each step.

3.2 Phases of computation

We now provide a brief overview of each phase of OpenCount. Figure 1 shows a diagram of the major steps in our system. Our pipeline is split up into two phases: blank ballot processing and voted ballot processing.

3.2.1 Blank ballot processing

The first phase involves interleaving computer vision and operator assistance in order to annotate the blank ballot images.

Preprocessing. The preprocessing stage transforms the scanned images (both the blank ballots and the voted ballots) into a normalized state which makes them easier to manipulate during the pipeline. In particular, preprocessing first straightens the images, and then resizes all images. These two steps simplify further computation.

Target and contest detection. In this step, the operator identifies the voting targets: the operator draws a bounding box around an example of a voting target, and OpenCount uses this to automatically find other voting targets. The operator can inspect the result and adjust the automatic results if necessary. Then, the operator collaborates with OpenCount to cluster the targets by contest.

Contest data entry. Next, the operator enters the title and candidate list for each contest. The purpose of this step is twofold: first, it allows OpenCount to correctly link all appearances of a contest on multiple ballots (e.g., the President contest may appear on many different ballot styles); and second, it allows the result of the election to be accumulated in a human-readable format. We do not rely on Optical Character Recognition (OCR); it is not reliable enough for our purposes.

Grouping patch selection. If the election has more than one blank ballot, the operator draws a bounding box around the grouping patch for each blank ballot. If the election has just a single blank ballot, this step is skipped. The grouping patch is a part of the scanned ballot that allows us to distinguish between blank ballots, based solely upon the contents of this part of the ballot. We assume elections have some grouping patch which can be used for this purpose.

3.2.2 Voted ballot processing

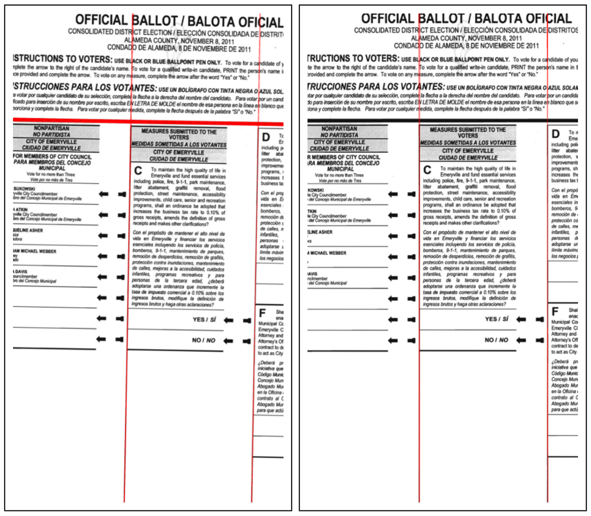
After the blank ballots have been annotated for an election, we perform processing on voted ballots for tabulation.

Ballot grouping. "Grouping" is the process of associating each voted ballot with its corresponding blank ballot. OpenCount examines the grouping patch of each voted ballot to find its corresponding blank ballot. This allows OpenCount to accurately locate all voting targets on each voted ballot.

Grouping verification. The operator then verifies that the grouping was accurate. We describe our methods for doing this efficiently in Section 4.6. Any mistakes found during verification can be corrected.

Target extraction. After all ballots have been grouped, we extract the voting targets from each voted ballot. For each voting target on its corresponding blank ballot, we extract the region at the same location on the voted ballot: this is the voting target as it appears on the voted ballot. The result is a collection of images of the voting targets (whether marked or not) on each voted ballot.

Target classification and verification. Next, OpenCount helps the operator classify each extracted voting target image as either marked or unmarked. OpenCount displays these images, sorted by average pixel intensity



(a) First pass (b) Second pass

Figure 2: Preprocessing detects and corrects for rotation by detecting long lines. We show the lines discovered by the first pass and second pass of our method.

and organized in a grid. The operator sets a threshold cut-off value that is used to separate marked and unmarked targets. The operator then corrects any classification errors that may remain. The operator can also view the entire ballot associated with a single target if more context is needed in order to determine the intent of a mark.

Postprocessing. Finally, OpenCount collects the target classifications and generates a CVR for each voted ballot.

4 Algorithms

4.1 Preprocessing

During the ballot scanning process, images typically experience small, random amounts of rotation and translation (i.e., a shift in some direction) between each scan. To better manage this variability, we perform preprocessing on all ballots to undo rotation at a coarse level.

In order to detect rotation, we use a linear Hough transform that is selective for near-vertical and near-horizontal lines, defined as being at some relatively small angle θ ($\theta \leq 4^\circ$) to either the vertical or the horizontal axis. We operate under the assumption that each ballot type will contain 2 or more relatively long vertical or horizontal lines. The parameters to the Hough transform are selected to detect lines whose length is at least $\frac{4}{5}$ of the ballot width; they are adjusted dynamically if no lines are found or if too many lines are found. The rotation detection occurs in two passes.

First pass. We first run a rough Hough transform, sensitive to 0.1° for the entire range of possibilities (e.g., 4° from the horizontal or the vertical in either direction). For each discovered line, we calculate its angle of rotation from the vertical axis. To filter out possible outliers, we take a trimmed mean of the discovered angles as our first-order approximation of the rotation angle.

Second pass. To improve precision, we run another Hough transform, this time over a narrower range of possible angles (0.1° in either direction from the estimate obtained in the first pass), sensitive to 0.01°. To estimate the overall rotation angle, we take the median of the resulting values. Figure 2 displays the lines found by both passes on a sample ballot.

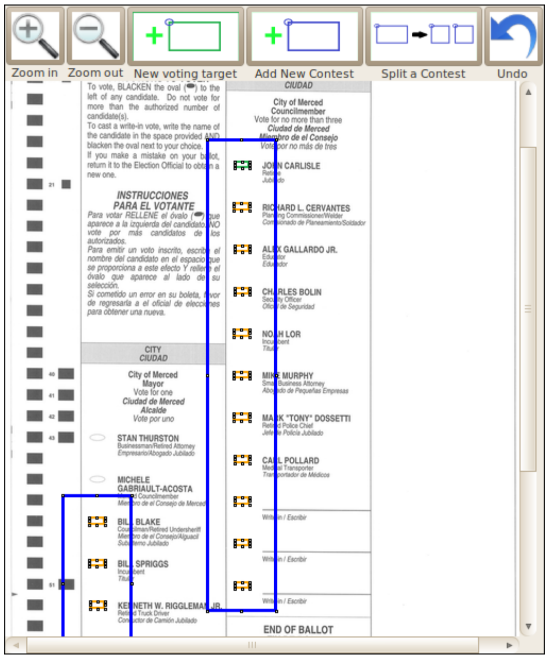


Figure 3: After the operator selected one voting target on this blank ballot from Merced County, OpenCount automatically detected the location of most of the voting targets. The green bounding-box is the exemplar target—the rest were auto-detected by OpenCount. Note the two undetected targets: the operator will have to identify another exemplar target to help OpenCount detect the rest.

4.2 Voting target detection

OpenCount needs to identify the locations of all voting targets on all ballots. This task is nontrivial. Different ballot vendors deploy widely-varying ballot styles, each with their own style of voting target. In addition, while empty voting targets (with no voting marks) all look identical, marked voting targets exhibit broad variation, as different voters may mark their ballot differently. This makes it difficult to identify the location of voting targets from the voted ballots.

Our solution is to find the location of all voting targets by analyzing the blank ballots. On the blank ballots, the voting targets are uniformly empty (unmarked) and look visually identical, so it is easy to recognize all targets if we are given information about what an empty voting target looks like. We ask the operator to identify an example of an empty voting target by drawing a bounding box around it. OpenCount then automatically identifies other targets using *template matching*, a process for finding other places where a template image appears in a second image. The operator can inspect the results; if this automatic process has missed any voting targets, the operator can indicate them and OpenCount will use template matching to find additional instances it missed in the first round, repeating until all targets have been found. Figure 3 shows the result after OpenCount has located voting targets using one round of template matching.

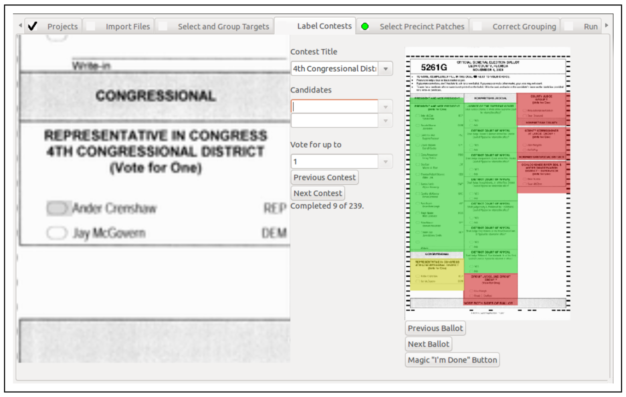
Template matching takes as input two images: an image I, and a template image T. A search is performed on I in order to try to identify possible locations where T might appear in I by comparing all T-sized patches in I to T with some comparison metric. OpenCount currently uses the *Normalized Correlation Coefficient* (NCC) [11] as the metric.

Due to differences in image conditions across ballot scans (e.g., scanner noise, skew), template matching may not detect all voting targets. If this happens, the operator can draw a second bounding box around another voting target. This allows OpenCount to template match on that voting target as well. If template matching produces false matches, as a last resort the operator can move, resize, or delete targets. In our experience, typically it suffices to identify one or two examples of voting targets. The operator must then review the detected target locations on all blank ballots, but this can usually be done rapidly.

4.3 Clustering targets into contests

Next, OpenCount attempts to automatically identify which voting targets are part of the same contest, using heuristics based upon the spacing and layout of the targets. The operator can inspect this clustering and adjust it manually if needed.

The intuition behind our algorithm is simple: on any given ballot, within a contest C, there typically is a fixed distance D between all voting targets within C. We use the smallest distance D' between any two targets as an estimate for D. Then, if two voting targets T_1 and T_2 are within $(1+\varepsilon)\cdot D'$ pixels of each other (where ε is a small error factor), we merge T_1 and T_2 into the same contest. All distances are measured using the Euclidean distance.



Contest Title: 4th Congressional Disti
Candidates
Vote for up to: 1
Completed 9 of 239.

Figure 4: The user interface for entering contest titles and candidate names.

4.4 Labeling contests

Next, OpenCount asks the operator to label each contest with its title (e.g., "President"), as well as the names of all candidates running in each contest (e.g., "Barack Obama", "John McCain", etc.). The operator manually enters this information for each contest on each different kind of blank ballot. If same contest appears on many ballots, the operator only needs to enter the text once; thereafter, the operator can select the contest from a drop-down of previously entered contests.

While this may seem like a scenario in which Optical Character Recognition (OCR) is easily applicable, we have found that open-source OCR libraries make too many errors and are too unreliable for this purpose. In addition, OCR is a "black box", so if it does not work well for some inputs, one has limited recourse, which makes systems that rely upon OCR more fragile. This makes OCR an unappealing choice for our purposes.

The user interface is shown in Figure 4. This screen includes a picture of the entire ballot (with color highlights used to indicate which contest the operator is currently working on and which contests have been completed) and an enlarged display of the current contest, which can be panned or zoomed by the operator if needed. OpenCount uses several heuristics to identify which part of the ballot to display enlarged; however, because the operator can pan this image, the heuristics do not need to be perfect.

4.5 Grouping

OpenCount automatically finds, for each voted ballots, its corresponding blank ballot by checking which one has a matching grouping patch.

The grouping computation assumes that ballots contain an image region, called the *grouping patch*, that allows for a unique mapping from a voted ballot to a blank

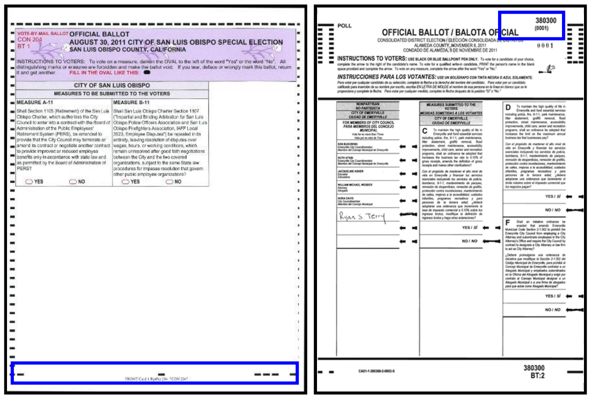

(a) San Luis Obispo County (b) Alameda County

Figure 5: Examples of grouping patches shown in blue for two types of ballots. For ballot (a), we use the timing marks as the grouping patch. For ballot (b), we used a precinct number and some text (which uniquely identifies the language) as the grouping patch.

ballot. This might be a portion of the timing marks (see Figure 5(a)), or it might be a region that captures the precinct number and ballot language (see Figure 5(b); the language was crucial in this example, because the voting targets are at different locations on different-language ballots). We utilize the NCC-based template matching technique, along with a direct pixel-based registration algorithm, to find the blank ballot whose grouping patch best matches that of the voted ballot. For each voted ballot, the algorithm simply scores how well the patch from the voted ballot matches the patch from each possible blank ballot; the closest match is assumed to be correct.

Direct pixel-based registration. A key challenge is that the voted ballot might not be perfectly aligned to the blank ballot, so we must correct for this. We model transformations between a voted ballot and a blank ballot using a *rigid model*. Motion between a point x and x' is governed by the equation $x' = R_\theta x + t$. R_θ is a rotation matrix, parameterized by θ, the degree of rotation, and t is a translation vector. This model allows for translation and rotation between the two images, but does not allow for other changes, such as scale or skew. Though other models have the power to capture broader families of transformation, the rigid model has fewer degrees of freedom, making it easier to estimate. Furthermore, the rigid model is sufficient to capture the variation typically introduced during the scanning process. We adopt the Lucas-Kanade [14] algorithm, which finds the parameters θ, t that minimize the least squared error between two images.

$$\text{squared error} = \sum_x [I(W(x;\theta,t)) - T(x)]^2$$

Algorithm 1 Pyramid ballot grouping

```
b: Voted ballot image
G: Set of grouping patches from blank ballots
mx: Maximum scale
 1: procedure BallotGrouping($b, G, mx$)
 2:     $p \leftarrow$ SuperRegion($b, G$)
 3:     $s \leftarrow$ InitMinScale($G$)
 4:     $step = (mx - s)/\log_2|G|$
 5:     while $|G| > 1$ do
 6:         for $i = 1, 2, \ldots, |G|$ do
 7:             $r_i \leftarrow$ NCC+LK($p, G_i, s$)
 8:         end for
 9:         Sort $r$, to get $\pi$ such that $r_{\pi(1)} \leq \cdots \leq r_{\pi(|G|)}$.
10:         $G \leftarrow \{G_{\pi(1)}, G_{\pi(2)}, \ldots, G_{\pi(|G|/2)}\}$
11:         $s \leftarrow s + step$
12:     end while
13: end procedure
```

The function $W(x;\theta,t)$ represents the location of pixel x after being warped by the transformation with parameters θ and t, which capture rotation and translation. We refer the reader to Lucas, Baker, et al. [14, 2] for a detailed review of this class of algorithms.

NCC+LK. To compute the similarity between a ballot and a particular grouping patch, we pair the NCC template-matching algorithm with Lucas-Kanade. This is done for two reasons: (1) variation in the scanning may affect the exact location of the grouping patch and (2) Lucas-Kanade is a gradient descent-type algorithm, sensitive to good initialization.

To score how well a grouping patch matches a blank ballot, we first perform NCC using the patch, then run Lucas-Kanade on the best matching location. The score output by NCC+LK is the least-squares registration error from Lucas-Kanade.

The pyramid optimization. We have developed an optimization that significantly speeds up this computation, by first performing the comparison on smaller, downsampled (lower-resolution) versions of the images to quickly prune away poor matches.

In computing the NCC+LK, we only consider the region in the ballot around the union of the grouping patch regions. We define SuperRegion(b, G) to compute the union of the bounding boxes of G, expanded, and cropped from ballot b. We adopt a pyramid scheme to efficiently find the closest match in G. The pyramid scheme initially applies NCC+LK to smaller-scale versions of the patches, then throws away the worst matches and repeats the process on the surviving pairs at a higher resolution.

Let the initial set of grouping patches be G. Let $p \leftarrow$ SuperRegion(b, G). We downsample p and the

Algorithm 2 Starting scale computation

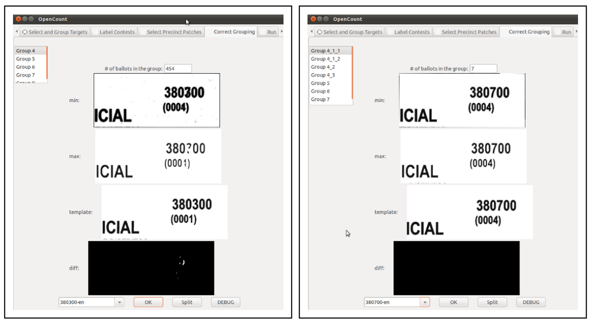
T: Set of blank ballot images
G: Set of grouping patches from T
mn: Minimum scale
mx: Maximum scale
d: Scale step

```
1:  procedure InitMinScale($T, G, mn, mx, d$)
2:      for $i = 1, 2, \ldots, |G|$ do
3:          $G' \leftarrow G \setminus \{G_i\}$
4:          $g \leftarrow$ SuperRegion($T_i, G'$)
5:          $s \leftarrow mx$
6:          for $j = 1, 2, \ldots, |G'|$ do
7:              $r_j \leftarrow$ Ncc+LK($g, G'_j, s$)
8:          end for
9:          Let $x$ be the index that minimizes $r_x$.
10:         repeat
11:             $s \leftarrow s - d$
12:             for $j = 1, 2, \ldots, |G'|$ do
13:                 $r_j \leftarrow$ Ncc+LK($g, G'_j, s$)
14:             end for
15:             Sort $r$, so $r_{\pi(1)} \leq \cdots \leq r_{\pi(|G|)}$.
16:         until $\pi^{-1}(x) \geq \frac{1}{2}|G'|$ or $s \leq mn$
17:         $y_i \leftarrow s$
18:     end for
19:     return Max($y$)
20: end procedure
```

patches in G to an initial minimum scale s. Next we measure the NCC+LK response of each patch in G to p and prune away the worst scoring half, retaining the remaining set. Then we increment the scale s and repeat the matching and pruning process on the pruned set G. This continues until only a single element remains in G. The algorithm is summarized in Algorithm 1. G_i represents the ith element of G.

This approach depends on knowing the smallest scale to begin the pyramid searching. Given the blank ballots T and patches G, we compute the smallest scale that allows for discrimination between grouping patches. The intuition behind our approach is that for any given blank ballot, we find the most similar grouping patch at the full scale, then continue reducing the scale until that patch is no longer among the top half closest matches to that blank ballot.

In more detail, consider a blank ballot T_i. We crop out the patch from T_i, call it g. We compare g to the patches G' from all other blank ballots, computing the NCC+LK response for each at full scale. Suppose $g^* \in G'$ is the closest match to g, at full scale. Intuitively, this means that g^* is the patch that is hardest to distinguish from g. We repeatedly reduce the scale and recompute the NCC+LK response at each scale until the patch g^* falls outside the top half of responses. This algorithm is summarized in Algorithm 2. We track the order of similarity of elements G to be used later in grouping verification.



(a) Cluster needing split (b) New cluster after split

Figure 6: A screenshot of the grouping verification UI. The figure on the left (a) shows a cluster overlay that contains at least one erroneously grouped ballot; the operator should click split to separate these ballots into two or more sub-clusters. The figure on the right (b) shows the resulting sub-cluster. One can readily verify that all of the ballots in the new sub-cluster are correctly grouped.

In our pyramid approach, the most NCC+LK comparisons are performed at the smallest scale, with the number of comparisons halved at every scale increase. The performance of both NCC and Lucas-Kanade is highly dependent on the scale of the image patches: they are much more efficient at small scales. With this approach, we can prune away the most dissimilar patches at lower resolutions while only comparing the most similar elements at the highest resolutions. In the situation where we have a large number of blank ballots, this results in significant computational gains. In practice this procedure is run twice — one for the original ballot image and a version rotated by 180° — to handle flipped ballots.

4.6 Grouping verification

Following the automatic grouping, the operator is presented with a simple interface to correct and verify the resulting mapping from voted ballots to blank ballots. We refer to the set of ballots matched to the same blank ballot as a *ballot cluster*. The operator views a summary image of all ballots in a ballot cluster, one cluster at a time (see Figure 6). For each cluster, the operator can (a) accept the cluster as accurately grouped, (b) indicate that the cluster contains voted ballots from multiple different underlying blank ballots, erroneously merged, or (b) correct errors by changing the blank ballot that all of the ballots in the cluster are associated with.

Verification. We visually summarize all of the ballots in the cluster using an *image overlay* [5]. An overlay is a

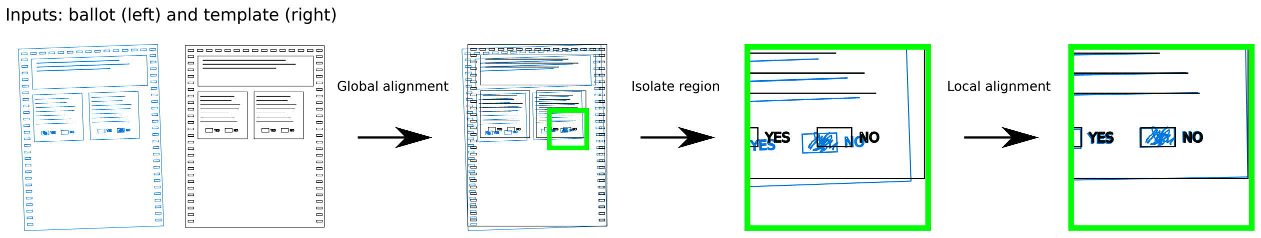


Figure 7: The target extraction procedure. The voted ballot is shown in blue and the blank ballot is in black. (1) First, we align the ballot to the blank ballot, globally. Since the transformation introduced by scanning is not perfectly uniform across a ballot, we (2) isolate a region around each voting target and (3) perform alignment once more, locally. Finally, we extract the bounding box corresponding to the location of the target on the blank ballot.

concise way of summarizing a large collection of images that are allegedly nearly identical. We show both a *min overlay*, which intuitively contains the union of the black pixels in the collection (if any image in the collection has a black pixel at location (x, y), then so does the overlay image), and a *max overlay*, which contains the intersection of the black pixels. To verify a cluster, the operator views an overlay of all the grouping patches from the cluster, juxtaposed with the grouping patch from the blank ballot that these were associated with, and the operator judges whether or not the association is correct.

Oor experience is that overlays make it easy for a human operator to quickly check whether all of the ballots were correctly associated to their matching grouping patch. See, for instance, Figure 6 for an example where the cluster contains at least one incorrectly grouped ballot (Fig. 6(a)) and an example where all ballots were correctly grouped (Fig. 6(b)).

Re-label cluster. Another action the operator can perform is to *re-label the cluster*. If the operator observes that the overlays have been incorrectly labelled (i.e., associated with the wrong blank ballot), she may use a drop-down menu to correct the error and select the matching blank ballot. The choices are ordered using scores from the grouping stage.

The ordering of blank ballots for a cluster is created in the following way. From the grouping step we have a ranked list of blank ballots, called L_b for a ballot b. Our goal is to combine the lists of all ballots in a cluster to form a single ranked list, to be used for the drop-down menu. We implement the following weighted voting scheme: for each ballot, go through its ordered list of L_b and cast a vote worth 2^{-j} for each blank ballot, where j is the position of the blank ballot in L_b. Thus, a blank ballot will receive 1 vote from ballots for which it is the best match, $\frac{1}{2}$ votes from ballots for which it is the second-best match, etc. The final ranked list of the cluster is obtained by sorting these votes in descending order.

Split cluster. Finally, if the operator observes that the cluster of ballots appears to represent multiple different blank ballots (i.e., some ballots were wrongly merged into this cluster), the operator can correct the error by *splitting the cluster*. This action will divide the cluster into smaller clusters based on scores from the grouping stage. The system then guides the operator through examining each of the smaller clusters, using the same interface.

To split a cluster of ballots, we again utilize each ballot's ranked list of candidate blank ballots, and group the ballots according to the n^{th} blank ballot on the list, where n is the smallest entry in such that we observe a difference in blank ballots among ballots in a cluster. For example, if all the ballots in a cluster have the same rank list of blank ballots up until the 5th element down their lists, then the ballots are re-grouped based on that ranked list index.

4.7 Voting target extraction

Once the voted-ballot-to-blank-ballot association is established, we extract the voting targets from each voted ballot. The blank ballots tell us where the voting targets are located; we then look at the same location in the voted ballot, and crop out regions around the voting targets so that later stages can classify them into filled vs. empty.

The main challenge is to align the voted ballot and its corresponding blank ballot, so that the locations of the voting targets in each are aligned. We solve this in two steps: (a) coarse global registration and (b) fine local registration around each individual target. We use the content of the ballot itself for alignment, not the ballot-specific registration marks which are sometimes obscured or missing from the scanning process. In this phase, we again use a rigid transformation model and direct registration algorithms (as described in Section 4.5).

The first step is to estimate the transformation between the voted ballot and its corresponding blank ballot at a coarse level, using the entire ballot image. We use the Lucas-Kanade algorithm, with a rigid transformation model. The ballot and blank ballot are both downsampled prior to coarse registration for performance reasons.

This produces an approximate alignment, but it is not perfect. Though the rigid transformation can model much of the variation between scanned images, often the variation introduced by scanning is not perfectly uniform across the image. Thus, it is possible for one region of the ballot to be well-aligned, while another region is not. Our second step, local alignment, addresses this problem. During local alignment, we apply the Lucas-Kanade algorithm to small regions around each voting target, separately and individually. Figure 7 shows an example of this two-step process.

4.8 Quarantining

For robustness, our processing pipeline is designed to automatically sanity-check its results and detect anomalies. Anomalous ballots are *quarantined* for manual review. Figure 12(b) shows examples where operator assistance was necessary. The top image shows an example where unusual voter marks can result in unpredictable situations that are best interpreted by a human; the bottom image illustrates a case where only part of the ballot was scanned, a situation that we cannot resolve automatically.

We detect anomalous ballots using a outlier detection procedure. We use distance-based outliers [10]: a value is a $DB(p,D)$-outlier when at least a fraction p of all other values in the dataset E are of distance greater than D from it. One appealing aspect of this approach is that the observed data need not follow a standard distribution.

During target alignment, we track the final least squares registration error from each target's local alignment step and use those values for outlier detection. This is a single value for each target that estimates the goodness of fit of the registration around that target. For each voting target, we form the set E of least squares registration errors for that target (across all ballots in the election) and check for $DB(p,D)$-outliers among the set E. In other words, we declare a registration error error $e \in E$ an outlier if we have $|e - e'| > D$ for at least a p fraction of all $e' \in E$. This can be computed efficiently by sorting the set E and iterating over its entries. In our work, we set $p = 0.999$ and $D = (\max(E) - \min(E))/2$ and perform the outlier test separately for each different voting target. Ballots containing any flagged targets are quarantined and set aside for manual review.

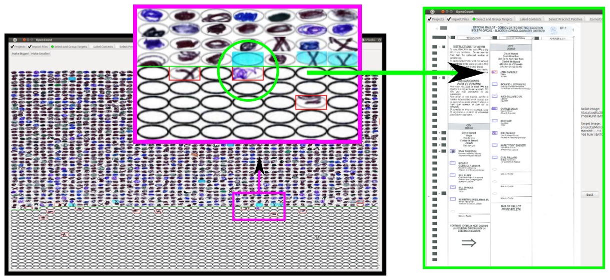


Figure 8: The mark classification step. The main screen shows all the targets in a grid. In purple, we show a zoomed-in version of a portion of this screen image. The operator can both set a global threshold and mark targets that are mis-classified by the threshold. Also, the operator may select a particular target and view the ballot containing that target, to help adjuticate voter intent.

4.9 Mark classification

The final step in OpenCount is to classify each extracted target from the voted ballots as either "marked" or "unmarked". We display all of the extracted targets in a grid, sorted by their average pixel intensity (similar to BallotTool [13]). The operator can then select a threshold that separates the marked voting targets from the unmarked.

In nearly all cases, voting targets are clearly marked or unmarked, as a majority of voters completely fill in the voting targets, so this approach works well for the overwhelming majority of voting targets. However, a small fraction of targets are borderline or less clearcut. We handle these marginal marks through manual review. We treat the operator as a domain-specific expert who has the final say, sidestepping the difficult problem of automatically inferring voter intent for marginal marks. In particular, the operator can adjuticate the status of any individual target and override the threshold-based classification for that target. If voter intent is unclear from the voting target alone, the operator can view the corresponding ballot (Figure 8) to help determine voter intent.

We have found that the grid-based interface is very helpful, as it allows the operator to efficiently scan a large collection of voting targets. Unusual or borderline cases tend to stand out visually.

As an optimization, OpenCount initially estimates a suggested threshold using clustering techniques. We make the simplifying assumption that the average pixel intensities of marked and unmarked targets each follow normal distributions, and we set the initial threshold as the value that minimizes the difference between the two actual distributions and the best-fit normal distributions.

We do not attempt to resolve write-ins [9]. Instead, we treat all write-in votes as votes for a single synthetic candidate, "write-in."

Election Name	Voted Ballots	Blank Ballots	Resolution
Alameda	1374	8	1460×2100
Merced	7120	1	1272×2100
San Luis Obispo (SLO)	10689	26	1700×2200
Stanislaus	3151	1	1700×2800
Ventura	17301	1	1408×3000

Table 1: We use data from five California counties.

Dataset	Prepr. (C)	Template creation (H)	Ballot grouping (C)	Grouping check (H)	Target extract. (C)	Target check (H)	Avg. per ballot (C)	Operator total (H)	Total in min.
Stanislaus	594	31s	-	-	478s	53s	0.34s	84s	19 mins
Merced	1011s	108s	-	-	3045s	712s	0.57s	820s	81 mins
Ventura	2851s	107s	-	-	6197s	490s	0.52s	597s	161 mins
Alameda	848s	510s	403s	87s	896s	162s	1.56s	759s	48 mins
SLO	1924s	438s	6219s	998s	5722s	29s	1.30s	1465s	256 mins

Table 2: Timing information for each dataset. Steps that use operator interaction time are labeled with (H) and steps that use computing time are labeled with (C). In template creation, we sum the total time required by the operator to detect targets, perform data entry, and select grouping patches.

5 Evaluation

We evaluated our system on data from the California 2011–2012 post-election risk-limiting audit pilot program [1]. The data covers elections in five counties and includes voted ballots, blank ballots, and CVRs. See Table 1 for a summary.

The CVRs obtained from the dataset were produced from a much earlier prototype which we used to conduct the audits in 2011. During those audits, we compared the CVRs against the official voting tallies. In the few cases where discrepancies were found, these CVRs were consistently found to be more accurate than the official tallies. For sake of evaluating our current system, we regard the existing CVRs as ground truth and measure the accuracy of OpenCount by comparing its output to the CVRs provided in the datasets.

In our evaluation, we measure both operator interaction time and computation time; see Table 2. A member of the team that developed OpenCount served as operator in all the tests. All timings were measured on a four-core machine with an Intel i7-950 CPU and 12GB of RAM.

We emphasize that we do not claim to develop the most sophisticated fully-automated vote tabulation system. Rather, our system helps a trained human operator find the difficult cases to make an informed judgement. We highlight all such difficult cases in each evaluation. We also highlight interesting examples of ballots detected as outliers for each dataset.

Three of the elections—Merced, Stanislaus, and Ventura—had only a single ballot style, and thus did not exercise the grouping step. Two others had multiple ballot styles and required grouping. We use OpenCount to process all datasets with no additional tuning. See Figures 9, 12, and 13, for example ballots.

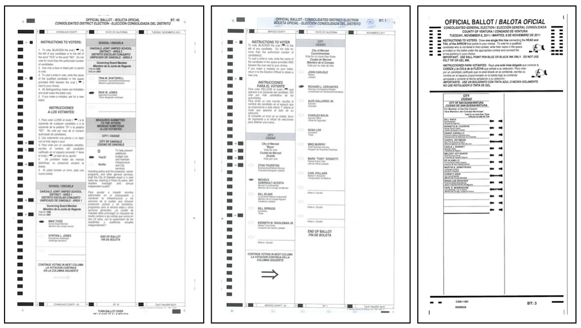

(a) Stanislaus (b) Merced (c) Ventura

Figure 9: Example ballots from three counties.

In every one of the five elections, with the possible exception of a few judgement calls, we achieve perfect accuracy. These judgement calls represent ambiguous cases where the human operator made a different decision in classifying the target, and where it is not clear what the right outcome should be. These are unrelated to the performance of our system.

Stanislaus. The Stanislaus County ballots were processed in 19 minutes. (0.34 seconds of computation per ballot). The CVRs generated by OpenCount matched the ground truth completely. In this election only a single contest was audited, resulting in minimal operator effort during contest labeling.

Merced. The Merced County ballots took 81 minutes to proces. We observed some discrepancies between the CVRs produced using OpenCount compared to the ground truth. Upon closer inspection, we believe that all

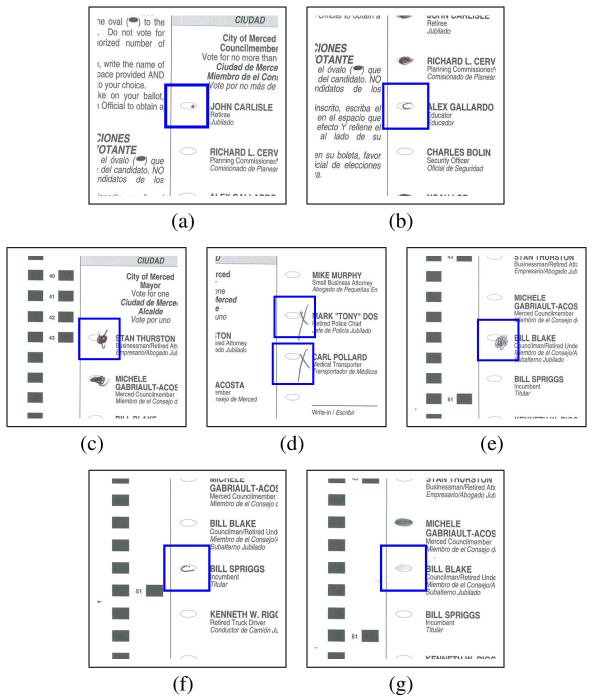


Figure 10: In these examples, the ground truth registered a vote but OpenCount's operator treated them as non-votes. In (g), the target appears to have been darkened through erasure, damaging the ballot.

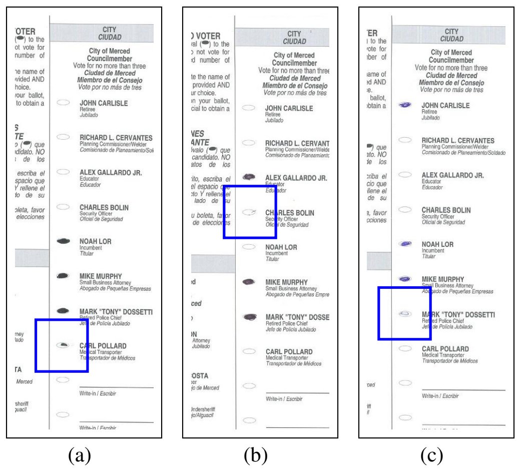


Figure 11: In these Merced "vote for three" contests, the ground-truth CVRs counted these as valid marks and treated the contest as overvoted. OpenCount's operator considered them hesitation marks (i.e., not a valid mark), so the contest is interpreted as containing three valid votes.

disagreements consist of borderline cases of interpreting voter intent. Figures 10 and 11 (in the appendix) display every target disagreed upon, highlighted in blue. In all cases, the OpenCount operator regarded the particular target as un-marked while the ground truth CVRs treat them as marked. As we rely upon a human operator to adjudicate such cases and it is not the purpose of these experiments to evaluate the operator's judgements, we make no claims about which result is "correct." It is enough that OpenCount helped the operator identify these cases, so that the operator could form a judgement.

The goal of our system is to steer the operator's attention to the difficult cases, so the operator can make an informed decision. OpenCount was successful in doing so. In the borderline cases mentioned above, the only difference between the results is that the operator judged the target differently.

Ventura. The Ventura County ballots took 161 minutes to process We identified only one discrepancy between our CVRs and the ground truth, shown in Figure 12(a). We consider this another difficult case, which is left up to election officials.

We discovered interesting ballots that OpenCount automatically flagged as suspicious (Figure 12(b)). In one example, the scanned image is completely occluded, halfway down the ballot. In the other, a voter scribbled out a marked vote, perhaps to make a correction. Both cases were automatically flagged as outliers and set aside for manual review.

Alameda. The Alameda County ballots took 48 minutes to process OpenCount produced the same CVRs as the ground truth.

The Alameda County case study presented an interesting challenge. Ballots were cast in four different precincts, and each precinct had an English/Spanish and English/Chinese version of each ballot. As Figure 13 shows, the choice of language influences the locations of the voting targets. As such, the choice of grouping patch was especially important in order for OpenCount to be able to correctly tabulate the voted ballots. Note that the timing marks, which are labeled in purple and encode the precinct number, did not appear to encode the language used. Therefore, we defined a grouping patch that included both the precinct number and a portion of text from the ballot (to capture the language).

San Luis Obispo (SLO). The SLO dataset was distinctive due to the relatively large number of blank ballots. OpenCount processed the ballots in 256 minutes Comparing the CVRs from our system yielded only one discrepancy, shown in Figure 14. This is another case of a judgement call in which the human operator made a different decision. All other ballots matched the ground truth CVR completely.

Discussion. Overall, our experience with OpenCount has been positive. However, we have identified several

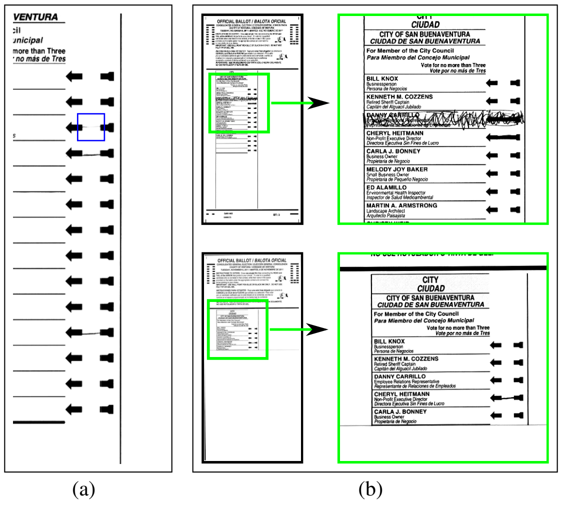


Figure 12: Examples from Ventura. (a) contains a case where the ground truth claimed three votes, while OpenCount claimed two votes. (b) contains examples of ballots automatically flagged for manual inspection.

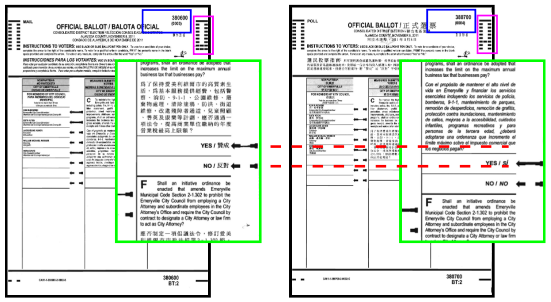


Figure 13: Example ballots from Alameda, which has four precincts (purple box) as well as two different language styles. We show two ballots from the same precinct, but with different languages; in green we zoom in on a single contest. As the red dashed line indicates, the language affects the location of the voting target. The blue grouping patch captures both the language and the precinct information in order to identify the ballot style.

areas for future improvement. While our current approach scales satisfactorily to elections with dozens of ballot styles, the operator effort to label all contests and verify the grouping results becomes prohibitive if there are thousands of different ballot styles. In addition, we have learned that the requirement to find and scan one of each kind of blank ballot is labor-intensive for election officials. A way to eliminate this requirement would be valuable.

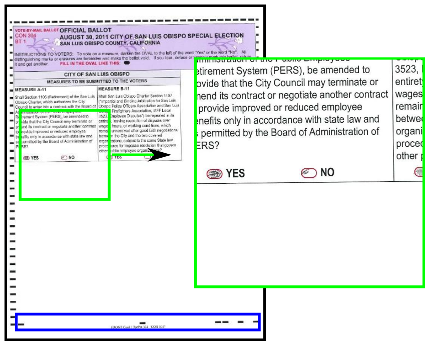


Figure 14: The one discrepancy for the SLO election. The operator of OpenCount adjuticated this as a vote for *YES*; the ground truth classifies it as an overvote.

6 Related Work

We were inspired by the pioneering work of TEVS [21] and Votoscope [8], which attempt to solve the same problem. As mentioned before, one difference between TEVS and OpenCount is that TEVS is specialized to a vendor's particular ballot layout, whereas we try to avoid making assumptions about ballot layout and, for robustness, avoid relying upon OCR. The current implementation of TEVS supports only Hart ballots. In addition, TEVS does not provide the operator an opportunity to verify or visualize intermediate results of the computation, whereas it is a key goal of OpenCount to enable the operator to verify the results of its computations.

BallotTool is another system to assist a human operator in processing a set of ballot images [13]. BallotTool requires the operator to define the ballot layout and identify voting targets, whereas we attempt to largely automate these tasks. Our work distinguishes itself in our tight integration of computer vision techniques with focused operator interaction. Our careful interleaving of processing and verification enables us to reduce the operator workload while being robust to scanning errors exhibited by the scanning process and general to a wide class of ballot styles.

Nagy et al. describe a method for identifying candidate locations for voting targets, based upon analysis of the timing marks on the edge of the ballot [15]. However, we have found that not all vendors' ballot styles contain these timing marks. For instance, Alameda and Ventura Counties use Sequoia ballots, which do not have the necessary timing marks. In addition, their method failed on 1% of ballots in their experiments, which is too high for our purposes. Therefore, we developed other methods.

Many recent works have looked at various aspects of the ballot analysis problem [5, 22, 9, 13, 23]. Works that are complementary to ours include tools for guiding an operator to discover sources of errors in scanned ballots [5], analysis of write-in regions [9], and capturing a stream of paper ballots at a distance using video [22]. Xiu et al. describe more more sophisticated approaches to fully automated target classification [23], though we have not found a need for these techniques in our experiments to date.

7 Conclusion

In this paper, we present OpenCount: a scaleable, robust, and accurate tool that helps support election auditing. OpenCount's unique combination of computer vision techniques and operator assistance allows us to operate on real-life elections, without relying upon or integrating with vendor systems. OpenCount has been successfully used to support risk-limiting audits in five California counties.

The OpenCount software is available to the public at `https://code.google.com/p/opencount/`.

8 Acknowledgments

We thank the California Secretary of State's office and election officials at Stanislaus, Merced, Ventura, Alameda, and San Luis Obispo counties for sharing scanned ballots. This research would not have been possible without their generous assistance. We also thank Philip Stark for helpful guidance, the anonymous reviewers and our shepherd Alex Halderman for many helpful comments, and Ed Smith and Dominion Voting Systems for helpful information about Dominion's future voting systems. We thank Andy Chang for experiments analyzing TEVS. Finally, we thank Piotr Dollár for his valuable feedback and for making his Matlab toolbox[1] available.

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] Post Election Risk-Limiting Audit Pilot Program 2011-2012. `http://www.sos.ca.gov/voting-systems/oversight/risk-limiting-pilot.htm`.

[2] BAKER, S., AND MATTHEWS, I. Lucas-Kanade 20 Years On: A Unifying Framework. *International Journal of Computer Vision 56*, 3 (March 2004), 221–255.

[3] BENALOH, J., JONES, D., LAZARUS, E. L., LINDEMAN, M., AND STARK, P. B. SOBA: Secrecy-preserving Observable Ballot-level Audits. In *Proceedings of EVT/WOTE 2011*.

[4] CALANDRINO, J. A., HALDERMAN, J. A., AND FELTEN, E. W. Machine-assisted election auditing. In *Proceedings of EVT 2007*.

[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] HALL, J. L., STARK, P. B., MIRATRIX, L. W., BRIONES, M., GINNOLD, E., OAKLEY, F., PEADEN, M., PELLERIN, G., STANIONIS, T., AND WEBBER, T. Implementing Risk-Limiting Post-Election Audits in California. In *Proceedings of EVT/WOTE 2009*.

[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] JI, T., KIM, E., SRIKANTAN, R., TSAI, A., CORDERO, A., AND WAGNER, D. An analysis of write-in marks on optical scan ballots. In *Proceedings of EVT/WOTE 2011*.

[10] KNORR, E. M., AND NG, R. T. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB '98)*, pp. 392–403.

[11] LEWIS, J. Fast Normalized Cross-Correlation, February 1998. `http://www.idiom.com/~zilla/Papers/nvisionInterface/nip.html#tex2html1`.

[12] LINDEMAN, M., AND STARK, P. B. A gentle introduction to risk-limiting audits. *IEEE Security and Privacy* (2012). Special Issue on Electronic Voting, to appear.

[13] LOPRESTI, D., NAGY, G., AND SMITH, E. B. A document analysis system for supporting electronic voting research. In *Proceedings of the 8th IAPR International Workshop on Document Analysis Systems (DAS '08)*, pp. 167–174.

[14] LUCAS, B. D., AND KANADE, T. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI'81) - Volume 2*, pp. 674–679.

[15] NAGY, G., LOPRESTI, D., SMITH, E. H. B., AND WU, Z. Characterizing Challenged Minnesota Ballots. In *Proceedings of SPIE* (2011), vol. 7874.

[16] RESCORLA, E. Understanding the Security Properties of Ballot-Based Verification Techniques. In *Proceedings of EVT 2009*.

[17] STARK, P. Report on second risk-limiting audit under AB 2023 in Monterey County California. Verified Voting Blog, May 2011. `http://blog.verifiedvoting.org/2011/05/07/1370`.

[18] STARK, P. Personal communication, May 2012.

[19] STARK, P. B. Efficient post-election audits of multiple contests: 2009 California tests. In *4th Annual Conference on Empirical Legal Studies (CELS 2009)*.

[20] STARK, P. B., AND WAGNER, D. A. Evidence-based elections. *IEEE Security and Privacy* (2012). Special Issue on Electronic Voting, to appear.

[21] TRACHTENBERG, M. Trachtenberg Election Verification System (TEVS). `https://code.google.com/p/tevs/`.

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

[23] XIU, P., LOPRESTI, D., BAIRD, H., NAGY, G., AND SMITH, E. B. Style-Based Ballot Mark Recognition. In *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR 2009)*.

[1] `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`