Real-Time Detection and Reading of LED/LCD Displays for Visually Impaired Persons

Ender Tekin James M. Coughlan Huiying Shen
The Smith-Kettlewell Eye Research Institute
San Francisco, CA
{ender, coughlan, hshen}@ski.org

Abstract

Modern household appliances, such as microwave ovens and DVD players, increasingly require users to read an LED or LCD display to operate them, posing a severe obstacle for persons with blindness or visual impairment. While OCR-enabled devices are emerging to address the related problem of reading text in printed documents, they are not designed to tackle the challenge of finding and reading characters in appliance displays. Any system for reading these characters must address the challenge of first locating the characters among substantial amounts of background clutter; moreover, poor contrast and the abundance of specular highlights on the display surface – which degrade the image in an unpredictable way as the camera is moved – motivate the need for a system that processes images at a few frames per second, rather than forcing the user to take several photos, each of which can take seconds to acquire and process, until one is readable.

We describe a novel system that acquires video, detects and reads LED/LCD characters in real time, reading them aloud to the user with synthesized speech. The system has been implemented on both a desktop and a cell phone. Experimental results are reported on videos of display images, demonstrating the feasibility of the system.

1. Introduction

LED and LCD displays are becoming increasingly prevalent in modern household appliances, posing significant barriers to blind and visually impaired persons who want to use such appliances. While OCR-enabled devices are emerging to address the related problem of reading text in printed documents, they are not designed to tackle the challenge of finding and reading characters in appliance displays. Indeed, the most popular portable OCR-enabled device that is designed to help visually impaired users read printed documents such as books and restaurant menus, the knfbReader Mobile, [10], states in its user manual that "Other circumstances that may lower accuracy include: ...LED and LCD screens."

While it may seem that the low variability of LED and LCD display characters (e.g. the common 7-segment character set encompassing the digits 0 through 9, which we consider in this paper, forms a simple, fixed "font") should make them easy to read, they are in fact difficult to detect and read under typical imaging conditions. The first reason for this difficulty is that, while OCR techniques require the text of interest to fill most of an image, a typical image of an LED/LCD display contains mostly background clutter that must be discarded before the display can be read.

Moreover, several characteristics of LED/LCD displays pose additional visibility problems. LED displays are often so high-contrast that the digit segments saturate the image and "bleed" into fuzzy blobs; LCD displays are typically low-contrast, which makes them hard to detect, and to compound this problem, the edges of the digits are often so close to the borders of the display frame that they effectively disappear; and the visibility of both displays is greatly impaired by specularities from the display surface that often occlude parts of digits or entire sets of digits, see Fig. 1. Finally, note that these visibility problems vary greatly depending on the exact camera location relative to the

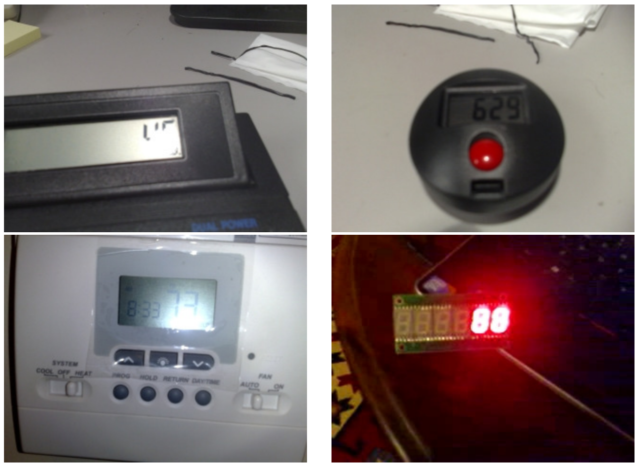
Figure 1. LCD's can suffer from glare and low contrast, whereas LED's can saturate images.

display. For instance, from a particular viewpoint, the reflection of a bright light may occlude one or more digits in a display – but the occlusion may disappear altogether if the viewpoint is changed slightly.

To make a practical display reader system, not only is it important that the computer vision algorithms be robust to the kinds of image noise and degradation obtained under typical viewing conditions, but the system must process video quickly, at least a few frames per second. Such rapid processing allows the user to slowly vary the camera position and angle in such a way as to maximize the chances of quickly obtaining at least one clear image of the display. By contrast with the operation of OCR-based systems such as the knfbReader, our system does not force the user to snap multiple photos, waiting up to a few seconds each time until a satisfactory image has been obtained.

We demonstrate a prototype display reader system which reads 7-segment LED/LCD displays on (i) a desktop computer using a webcam running Windows XP, and (ii) the Nokia N95 cell phone running Symbian OS. The computer vision algorithms underlying the system consists of a novel "blob" feature detection system, which quickly extracts candidate digit features in the image, followed by a Hough-type voting scheme to classify each blob as a digit (0 through 9) or non-digit, and then a grouping stage to determine the presence of coherent strings of LED/LCD display characters. Due to the fast nature of these steps, the algorithms run at real time, achieving over 15 frames per second on the desktop, and around 5 frames per second on the Nokia N95. We describe the algorithm performance on some videos of LED/LCD display images, demonstrating the feasibility of the system.

2. Related Work

Comparatively little work has addressed the specific problem of reading LED/LCD displays. One past approach is the Clearspeech system [7], which runs on a desktop PC and requires that special markers be affixed around the borders of each LED/LCD display to guide the system to the location of the display characters. Our approach builds on [9] on a cell phone-based LED/LCD display reader, which requires no such modification of the display. This past work extracts horizontal and vertical edge features in the image (corresponding to the horizontal and vertical segments of the the 7-segment characters) and groups them into "figure" and "ground" (i.e. digit region and background, respectively) using a simple graphical model (MRF).

In our experience with LED/LCD images, however, we found that many character segments are distorted in such a way that makes it difficult to reliably extract horizontal and vertical segments. Thus, we decided to extract "blob" features (see Sec. 3.2) instead, each of which typically corresponds to one character, which we found to be more reliable. Once suitable blob candidates are extracted it is straightforward to classify them into digit categories (0 through 9 or non-digit), as described in Sec. 3.5.

There exists a large body of literature on finding text in natural images ([12, 5, 2], see [6] for a survey) that we considered while devising possible approaches to detecting LED/LCD characters. However, due to the particular challenges of the led/lcd character domain and the need for real-time performance, we chose the blob feature extraction approach which we describe in more detail in section 3 and is fast enough even using low-powered processors such as those in a cellular phone. Specializing to the limited domain of 7-segment LED/LCD digits has simplified the problem so as to allow us to create a prototype cell phone system that runs in real time. The real-time performance also allows us to somewhat alleviate some of the issues that arise in this specific domain, such as the disappearance of digits due to specularities and low-light conditions, as the user can adjust the relative orientation of the display and camera to get a reading.

3. Finding and Reading Digits

The approach taken in this paper uses a connected component analysis on binarized input video to extract "blobs", which are then analyzed further to detect the candidates that are more likely to be digits. Such an approach allows for rapid detection of possible digits and is suitable for embedded implementation. The blobs are then grouped to get more reliable estimations of the digits, as otherwise it is possible to confuse background shapes with single digits very easily. Thus, we restrict our approach to finding groups of at least two digits. Finally, we estimate the class of the remaining blobs (digits 0...9 and 'non-digit') and display the results in the proper order. We present the details of the algorithm below.

3.1. Image binarization

We start out by binarizing our original image. Due to the low-contrast nature of LCD displays, we use a method similar to Niblack's method [8]), which uses a local mean and variance to build a threshold. We build on the algorithm proposed in Feng and Tan [3], but use some estimations to reduce the computational load. In our method, the threshold T for a pixel is calculated using two windows (the second being twice the size of the first), and is given by

$$T = \mu_s - \tau \tag{1}$$

$$\tau = \max\{\frac{(\mu_s - m_L) \times (1 - \sigma_s/\sigma_L)}{2}, \tau_{min}\} \tag{2}$$

where μ_s is the mean of the intensities in window around a pixel, m_L is the minimum intensity in a larger window, and σ_s and σ_L are the standard deviations of the intensities within the smaller and larger windows, and τ_{min} is the minimum threshold. If μ_s is less than τ_{min}, then we set the threshold at τ_{min}. In our experiments, we chose the minimum window to be 17 pixels and the larger window to be 35 pixels. τ_{min} was 4. As binarization is the most computationally intensive part of the application, to further speed up the calculations, we calculate the threshold at half the resolution of the original video, and the minimum, at a quarter the resolution. We note that we still achieve good results with a significant speed-up, see Fig. 2

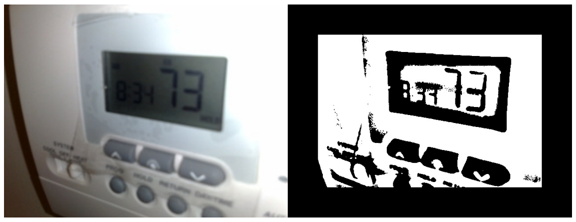
Figure 2. Result of binarization on LED screen

3.2. Extracting Blob Features

To extract the blobs, we do a simple connected component analysis. We first sweep the binarized image horizontally, and extract segments of pure black and pure white pixels. These are then further grouped together by vertical sweeps - for each segment, we move vertically down and add to a list any segment that is vertically overlapping with it for at least one pixel in the row below, and remove these segments from the global list of segments. We repeat this procedure for every segment in this growing list. Once no more segments can be added in a downward sweep, we switch directions and repeat this process again for each segment in the list. We keep alternating between directions until no more segments can be added to this group. We then save this blob, and repeat the procedure for all remaining segments in the global space until no more segments are left. We note that we keep separate lists of "white" and "black" blobs to be able to detect both polarities of LED/LCD displays.

3.3. Blob Filtering

Next, we filter out blobs that are considered too small or too large. During this filtering, we calculate bounding boxes on the blobs. All blobs smaller than 3x8 or larger than 40x80 are removed. We also remove any blobs that have aspect ratios (defined as height/width) of less than 0.8 or more than 5. The reason for the conservative ratios is due to the fact that at this point, some of the digits (especially ones that do not have the middle segment, 1 and 7) can be divided into two blobs (another challenge of LED/LCD digits is that their segments are not necessarily connected.)

To merge discrete blobs that may be part of the same digit, we look for vertically overlapping blobs that have about the same height, and are only a short distance away from each other. All such blobs are merged into a single blob. Once these are also merged,

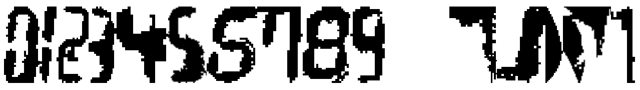
Figure 3. Some blobs. The first ten show blobs that belong to digits. The right section shows non-digit blobs found after filtering. Note that we find both polarities (white on black and black on white) of digits, we convert them to black on white for display purposes. Also, note that corners and edges can pop up similar to the digits 7 and 1.

we do a second level filtering to remove all blobs that are smaller than 6x16 and have aspect ratios smaller than 1.25. Fig. 3 shows some examples of digit-blobs and non-digit blobs.

3.4. Blob Grouping

Similar to text, it is very easy to mis-estimate background clutter as a likely digit without constraints (especially the digit '1' which is just a small segment.) Thus, we restrict our estimation to groups of at least two digits. We iterate over blobs and form a neighborhood list by seeing if there is a similarly sized blob to the left or right of a blob. This neighborhood list allows us to (i) eliminate single spurious estimates of digits from background blobs that may resemble a digit, and (ii) read the digits in correct order once decoded. If a digit in a group is not successfully decoded, it also allows us to signal this fact, and wait for a reliable class estimate for all before announcing the reading.

3.5. Blob Classification

We use an additive voting scheme similar to a Hough voting scheme, but diverges from the classical Hough in that each pixel in a blob votes for all digits in a one-dimensional space. We also experimented with Haar-like windows to detect segments similar to the classical Viola-Jones face detector [11], but the pixel-wise voting scheme outperformed this method and the extra computational load was negligible.

We first extracted a large amount of digit blobs from several training videos (3336 blobs) which were labeled with the correct digit. We then used a simple method to estimate the probabilities of each pixel being 'on' for a particular digit by binning the pixels in the training blobs to an image of size 10x20. For a given bin in the resized image, divided the number of 'on' pixels from the original image by the total number

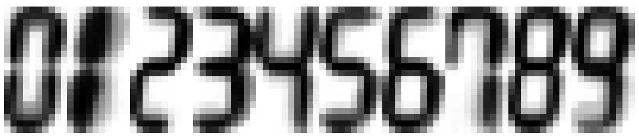
Figure 4. "Average" digits, reflecting the probability of a pixel being on/off for a given digit.

of pixels from the original image that fall in the same bin in the new image. We then averaged these probabilities over all images of the same category to get a representative 10x20 image that reflected the probability of a pixel in a 10x20 blob being on or off for a given digit, see Figure 4.

For inference, we did a similar thing for each blob. We resized the blobs by binning them to a 10x20 grid and calculated the probability that each bin is 'on' or 'off'. We then calculated the probability of a digit D based on the blob $B = \{B_i\}$ where i indicates a pixel index as:

$$p(D|B) = \prod_i P(D|B_i) \tag{3}$$

where

$$\begin{aligned} p(D|B_i) &= p(D|B_i = \text{'on'})p(B_i = \text{'on'}) \\ &\quad + p(D|B_i = \text{'off'})p(B_i = \text{'off'}) \end{aligned} \tag{4}$$

To avoid classifying all digits as blobs, we used a threshold for the probabilities of a digits. For the class of 'non'-digit blobs, we initially just assumed that each pixel had an equal likelihood of being 'on' or 'off' to calculate this threshold. However, we found this to be too conservative, and instead used a fixed threshold based on our experiments. We apply two thresholds: (i) the likelihood of the most likely digit must be greater than the second by a factor, and (ii) the likelihood of the best digit must not be below a certain threshold.

Before reporting the results, we ensure that all digits in a group have been decoded reliably, and signal this with an audio beep. Upon a button press, the system speaks out loud the decoded digits that were last decoded correctly as a group. We use this method to ensure that displays that are changing (such as clock) do not produce too much chatter.

4. Experimental Results

We have done a 5-fold cross-validation of our decoding algorithm on labeled blobs extracted from the images. We divided our database of 3336 labeled blobs randomly into 5 roughly equal sets. In turn, we trained our algorithm on 4 of the subsets, and used the 5th subset as a test set for inference. Among the 5 sets, We achieved an average error rate of 1.19%, and a maximum error rate of 1.52%.

We also tested our algorithm on several videos taken using the N95 cell phone camera, a webcam and directly on the N95. Even though the digits are missed in some frames, the real-time aspect means that in a short amount of time, the displayed digits are found. We note that unlike text, we cannot use a dictionary to reduce our error rate. However, by combining multiple frames for static displays, it is possible to increase the reliability of our estimate. Furthermore, this may allow the user to read parts of the display sequentially if glare may be a problem.

Figure 5 shows some still frames that are the result of running our algorithm on video taken by the N95. We note that the algorithm is rather robust to illumination conditions. However, we do sometimes miss digits, as seen in the bottom two figures. In one case, the 9 blends into the frame and is then filtered out, whereas in the bottom case, the '1' is lost due to the gap between strokes being larger than allowed. We provide some videos in our supplemental materials that also show some of the modes where we may miss digits.

We have also ported the algorithm and run it on the N95, achieving about 5 frames/sec in VGA mode. However, in the video mode and under low light conditions, the raw camera frames can have a lot of extra color noise or exposure problems. Thus, the reliability was not as good as we liked. We are currently investigating some methods to improve the picture quality from the N95 camera, and also other mobile platforms. We should stress that while binarization allows for a fast detection of blobs, we believe it would be more useful to consider the original images while decoding the digits as gradients can carry more refined edge estimations. We are hoping to find an improved trade-off between algorithm speed (which we believe is crucial), and better performance.

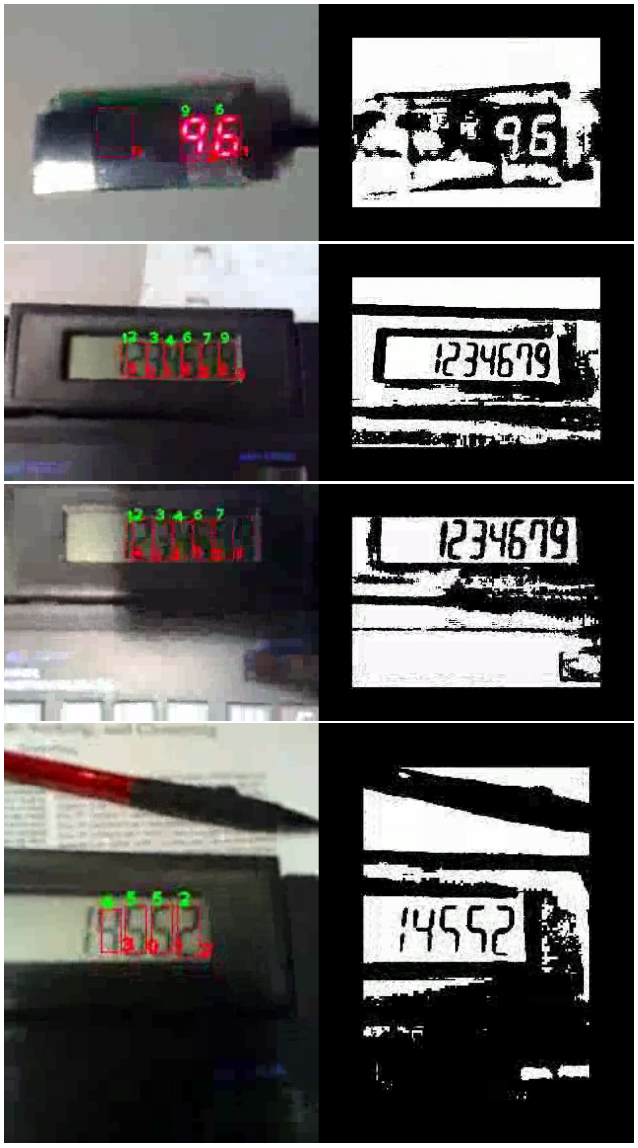
Figure 5. Experimental results on videos. The original images are on the left, and the corresponding binarized images are on the right. The red bounding boxes show discovered digits, the red indices at the bottom of each box is a blob index, and the green displayed digits are the digit estimates.

5. Conclusions

We proposed a novel, fast connected-component based algorithm to detect LED/LCD digits. We have developed our algorithm to run in real time (achieving over 15 frames/second on an Intel Pentium Dual-Core desktop with 2GB of RAM, and 5 frames/second on the Nokia N95 mobile phone running Symbian OS on a dual-core 332MHz TI processor), allowing a user to sweep around and possibly avoid issues such as glare, and furthermore making it possible to capture real-time displays such as microwave timers and clocks.

We will be testing our phone implementation with blind/visually impaired subjects, and incorporate their feedback into the final product. We are also exploring the use of other platforms such as Android and iOS, and plan to use newer cell phones that have better quality video cameras and faster processors. Also, we are considering using built-in accelerometers to give the users feedback and help the user hold the camera horizontal for displays such as microwave timers, ovens etc.

Another direction that we are currently exploring is using random trees, [1, 4], and some simple cues to improve the character decoding process. While training a random forest may be slow, we expect the inference to be fast enough for real-time performance.

In the future, we will integrate our functionality with general OCR to provide a complete suite of sign/display reader functionalities on a mobile device. While OCR does provide a more general functionality, for cases when OCR may be too unreliable, such as LED/LCD displays, or fast performance is required, we believe that it may still be useful to have a dedicated "display reader" mode, which will specialize either to 7-segment characters or other fixed "font" characters.

6. Acknowledgments

This work was supported by NIH grant 1 R01 EY018890-01.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588, 1996. 6

[2] X. Chen and A. Yuille. Adaboost learning for detecting and reading text in city scenes. In *CVPR*, 2004. 2

[3] M. Feng and Y.-P. Tan. Contrast adaptive binarization of low quality document images. *IEICE Electronic Express*, 1(16):501–506, 2004. 3

[4] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *CVPR*, 2009. 6

[5] A. K. Jain and B. Yu. Automatic text location in images and video frames. *Pattern Recognition, International Conference on*, 2:1497, 1998. 2

[6] J. Liang, D. Doermann, and H. Li. Camera-based analysis of text and documents: a survey. *International Journal on Document Analysis and Recognition*, 7:83–200, 2005. 2

[7] T. Morris, P. Blenkhorn, L. Crossey, Q. Ngo, M. Ross, D. Werner, and C. Wong. Clearspeech: A display reader for the visually handicapped. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 14(4):492–500, 2006. 2

[8] W. Niblack. *An Introduction to Digital Image Processing*. Prentice Hall, 1986. 3

[9] H. Shen and J. Coughlan. Reading lcd/led displays with a camera cell phone. In *CVPRW '06: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop*, page 119, Washington, DC, USA, 2006. IEEE Computer Society. 2

[10] K. Technologies. knfbreader. [Online] http://www.knfbreader.com/. 1

[11] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57:137–154, 2004. 4

[12] V. Wu, R. Manmatha, and E. M. Riseman. Finding text in images. In *DL '97: Proceedings of the second ACM international conference on Digital libraries*, pages 3–12, New York, NY, USA, 1997. ACM. 2