Whiteboard Image Extraction and Archival Tool
Implementation in iPhone Application

Sukolsak Sakshuwong, Ben Tsai
Department of Electrical Engineering
Stanford University
Stanford, USA

An algorithm designed to take advantage of the properties specific to the problem of capturing information from whiteboard images is presented. It is implemented as an iPhone application with real-time user feedback. The application categorizes captured information based on color. The saved image is greatly compressed and has improved visual contrast.

Keywords—image binarization; mobile image processing; k-means clustering; whiteboard

I. MOTIVATION AND GOALS

In face-to-face engagements, the whiteboard is still one of the most effective tools of collaboration allowing multiple individuals to express a wide variety of ideas. Typical images taken with mobile phone can capture all relevant information but produces unnecessarily large files making it more inconvenient to distribute than text based meeting minutes.

The goal of this project is to implement an iPhone application that is as simple to use as a camera and produce lightweight binary images suitable for mass distribution. The application will allow users to capture an image using the phone’s camera and produce a binarized image.

Furthermore, whiteboard images are often composed of markings in different colors which can be used to categorize the information. Therefore, the application should be able to segment the binarized image based on the colors of the markers used.

II. ALGORITHM

The algorithm is composed of two stages. The first stage globally thresholds the image and displays the results in real time. The second stage of segmenting the binarized image occurs only on demand when the user makes a request to save.

A. Real Time Features

During this stage, first the image is converted to grayscale. This conversion is a simple average of color components

Subsequently, a difference image is generated where each pixel is compared against a weighted average of its neighbors. To achieve efficient computation, an integral image is first generated where each pixel is the sum of itself and all pixels above and to its left. Simple averages over any arbitrary boxes can then be generated by arithmetic operations on the corners in the integral image. For each pixel, a weighted average where closer pixels are weight more than further ones is achieved by averaging the averages of overlapping rectangles of differing shapes.

Subsequently, a global threshold is computed using Otsu’s method [1] and a binarized image is produced by comparing each pixel to the global threshold. Pixels within a certain distance to the edge of the image are ignored for the determination of this threshold to improve resiliency against non-whiteboard features often found at the edges.

B. On Demand Features

After the binarized image is available, the algorithm then looks at the color information of the identified pixels to perform clustering and separating them into the color groups.

In order for the clustering algorithm to be effective, for every pixel, each individual color component is taken relative to sum of all components. To deal with non-white lighting, the difference operating used to precondition the grayscale image before global thresholding is subsequently performed. In order for this operation to be effective, each component ratio value is compared against gray value to zero out the distinction between grays and the lack of said particular component color.

Then k-means clustering is used to partition each identified pixel into k groups using distance between them in the color space. The algorithm works by first initializing a set of k means to arbitrary pixels. Then, for each identified pixel, it finds the closest mean and assigns the pixel to the cluster of that mean. We use Euclidean distance in the color coordinates as previously described. For each cluster, it finds the centroid of the points and moves the mean to the new centroid. The algorithm repeats this process until individual pixels are no longer reassigned or the maximum number of iterations allowed has been reached.

After k-means clustering is performed, each pixel is compared to its neighbors. If the pixel does not belong in the majority color and the number of neighboring pixels of the same color is a threshold below the number of pixels in the majority color, then the said pixel is assigned to the majority color. Since likewise colors are likely to occur next to each other, this operation can improve resiliency against the weakness of k-means clustering. Subsequently, a variation of the dilation and erosion is performed on each individual color
image. For every pixel, a count of the neighboring pixels is performed. If the count exceeds a certain threshold, the pixel is considered true. Likewise, if the count is below another predetermined threshold, the pixel is considered false. If neither condition is met, the pixel remains its original value. Both these operations rely on integral images to generate kernel.

Lastly, the component images are summed back together to produce the composite image.

---

**C. Computational Analysis**

In order to run on a mobile device, it is important that the algorithm is computationally effective. The difference images operations scales linearly with image size and linearly with number of boxes used to generate the weighted average. Global thresholding uses Otsu’s method which is known to scale linearly with image size. The color preconditioning uses difference images. The color separation scales linearly with number of identified pixels which is bounded by image size and number of k-means iterations which is bounded by a predetermined value. Both color comparison with neighboring pixels and the final erosion/dilation operation relies on integral images and scales linearly identified pixels count.

In conclusion, all operations scale linearly with only size of image and predetermined iterations of each operation and importantly are independent of the kernel size respectively used.

---

**III. IMPLEMENTATION**

We implement the algorithm as an iPhone application. The application runs on iOS 7. It is written in Objective-C for the user interface part and C++ for the calculation part. It uses Eigen, an open-source linear algebra library. We test the application on an iPhone 5s.

**A. Computation**

The image from the camera on the iPhone is provided as a one-dimensional array of pixels. Each pixel consists of blue, green, red, and alpha values. Each value is an unsigned 8-bit integer. We convert it into a matrix in Eigen for ease of manipulations.

Several optimization techniques are employed to improve computational effectiveness for real time operation. We avoid dynamic memory allocation by allocating memory on the thread’s stack when possible. We also avoid matrix copy by using pointers and reuse matrix variables whenever possible. We use low-resolution photos for the preview images. We iterate over columns first and then rows to utilize spatial locality in cache because Eigen arranges data in the column-major manner. The k-means algorithm uses squared Euclidean distances to avoid computing square roots.

**B. User Interface**

The user interface consists of the main screen, the camera screen, and the image screen.

When the application starts, the user sees the main screen. The main screen shows a grid of photos of whiteboards that the user has taken. The user can tap a button on the upper-right corner to take a photo. This button brings the user to the camera screen.

The camera screen shows a preview image and a thumbnail image on the upper-left corner. The preview image is a real-time result after doing binarization. The thumbnail image is the unmodified input from the camera. The user can tap and hold on the preview image to see the unmodified input. The button for taking a photo is at the bottom. The user specifies the number of colors of the photos. After the user takes a photo, the photo is processed and then added to the grid on the main screen.

Tapping an image on the main screen brings the user to the image screen. The image screen shows the composite image, the original image, and the image from each component. There are tabs on the top of the screen where the user can use to switch the images. The file size of each image is shown at the bottom.

---

**IV. ACHIEVED PERFORMANCE**

Our app is able to perform the desired behavior of converting typical images of whiteboard into binary images. It can handle uneven gradients in lighting and a reasonable amount of reflections. However, it fails whenever the white background detail is similar in magnitude to the marker information. Select results shown below.

---

**Fig. 2. Three screens of application.**

**Fig. 3. Sample Use Case. Original image size: 613kB. Composite image file size: 18kB**
V. CONCLUSION

We create an iPhone application that takes photos and turns them into binarized images by using thresholding which can be done in real time. The application also separates the binary image based on the marker colors on demand using k-means clustering. The resulting binary images are up to 50 times smaller than the original and have better visual contrast. The application performs at a speed comparable to taking a photo with the native phone application and is able to correctly convert the results of the typical whiteboard.

REFERENCES


APPENDIX A – DIVISION OF WORK

General algorithm design and project goals are performed jointly. Sukolsak focused on iOS specific code and architecture of the software project. Ben focused on algorithm testing and fine tuning.

APPENDIX B – SOURCE CODE README

File description:
- Blackboard/GridViewController.* - the main screen
- Blackboard/CameraViewController.* - the camera view
- Blackboard/ImageViewController.* - the image view
- Blackboard/process.* - the algorithm
- Blackboard/Eigen - the linear algebra library

To test the algorithm without using the user interface, type "make", create a folder named "output", and then run "./main" (or "main.exe" on Windows.) Specify the input file in main.cpp.