Adaptive Thresholding in IMAQ Vision

Publish Date: Aug 24, 2016

Overview

For Machine Vision users who are challenged by non-uniform or changing lighting conditions, the Adaptive Thresholding algorithm is an image segmentation tool that is both powerful and flexible. Unlike global value threshold algorithms, adaptive thresholding algorithm allows the user to extract relevant information from images under broad variations in lighting conditions

Table of Contents

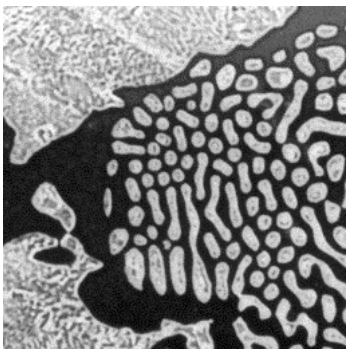
1. Global Value Thresholding

Thresholding is a technique that segments an image for the purpose of extracting information, such as the dimensions of an object. Thresholding is the most commonly used step in machine vision.

Using conventional thresholding methods, the accuracy of the results depend on good image contrast, which in turn depends on uniform lighting. But establishing uniform illumination conditions and maintaining them over time can be a challenge in some applications. To correct for variations in illumination, this application note describes a local value thresholding algorithm for non-uniform lighting conditions. The algorithm can also be used in a mode that continuously adapts to lighting conditions that change over time.

2. Selecting a Threshold Level

In order to maximize the performance of any thresholding algorithm, it is important to create the highest possible contrast between the object of interest and the background. A high-contrast image and its typical characteristics are illustrated in figure 1.



(a) Sample image

(c) Thresholded image

Figure 1. Sample image, histogram, and thresholding

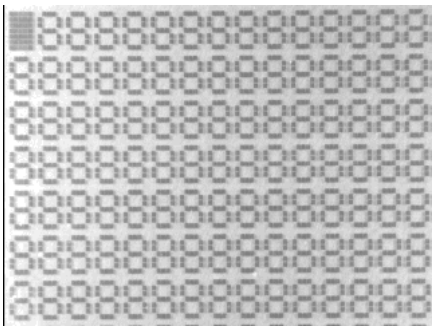
The two distinct intensity peaks correspond to the pixels within the object and pixels that represent the background. With this nearly-ideal distribution, you can separate the object from the background by simply choosing a threshold value somewhere between the two peaks. There are a number of auto-thresholding algorithms (such as Entropy, Binary Clustering, Metric,

Moments, Inter-class variance, Clustering, etc.) that can help select a threshold value. These methods analyze the entire image and use a combination of the statistical distribution (histogram) and statistical analysis of the intensities to compute the optimum threshold value.

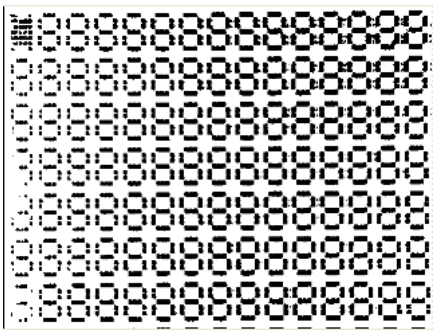
3. Effects of Non-Uniform Lighting

For some applications it is not possible to illuminate the object of interest in a way that consistently creates an ideal intensity distribution. The illumination may vary within the image (due to a non-uniform light source) or from image to image (due to inconsistent positioning of each object of interest) . It may also vary over time because a lighting element burns out or is accidentally moved.

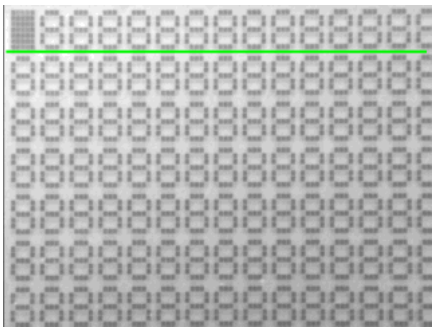
Figure 2(a) is an image of an LCD display that illustrates the effect of non-uniform illumination on the thresholding task.



(a) Non-uniformly illuminated display



(c) Thresholded image



(e) Green Line for line profile

Figure 2. Non-uniformly illuminated display

Consider an application where the task is to inspect this display and determine whether all the dots (display pixels) are functioning correctly. A visual inspection of the image in Figure 2a shows that all the dots are working. In order to inspect this display automatically, a machine vision system must distinguish dots that are "lit" from the background. But the histogram of the image (Figure 2b) shows that it will be difficult to select an effective threshold value as we were able to do in the previous example.

To understand why this is the case, let's examine what is going on in more detail. The histogram reveals that the image contrast is poor, because there are virtually no pixels with intensity values less than 125. In addition, the histogram peaks are not clearly separated by an obvious valley that would suggest an appropriate threshold value. Taking a background line profile (Figures 2e and 2f) also reveals that there is an illumination drift across the image. The drift is also visually apparent in other areas of the image. A consequence of the illumination drift is that any single threshold value will result in the loss of pixel state information in certain parts of the image (Figure 2c) .

4. Local Value Thresholding

In situations where lighting varies in space or time, it makes sense to use a local thresholding approach which adapts to changing illumination conditions by comparing the relative intensity of each pixel to the intensity of its neighbors. For every pixel, a programmable number of surrounding pixels are selected, and any of the auto-thresholding algorithms mentioned

earlier can be used to select an appropriate threshold level. One option to select a threshold levele is to select the mean value of the pixel group as the threshold level for that particular pixel.

The threshold values computed at each pixel location can also be saved to an array, which can then be used to
- analyze the variation in illumination
- balance the image
- threshold subsequent images (in applications where lighting conditions do not change over time)

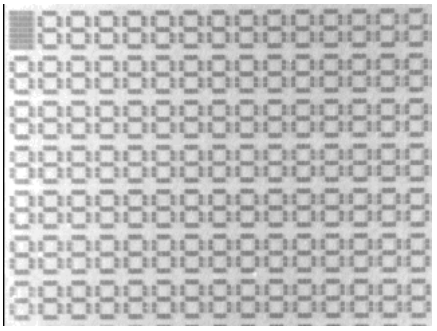
5. Pseudo Code

The following examples use an approach represented by this pseudocode:

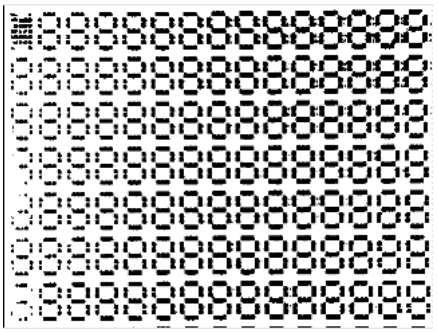
Steps
1. Select an image
2. Select a filter size
3. Set the boundary size of the image as the filter size
4. Select a region of interest (ROI) within the image
5. Extract a subimage that includes the ROI and a surrounding border of pixels corresponding to the filter size
6. Apply a convolution algorithm to the subimage to compute an array of mean value intensities for each pixel in the ROI
7. Subtract the mean value array from the original ROI to generate an image with balanced lighting.

An implementation of this algorithm is available as an example program at the link provided at the end of this document.

6. Sample Images



(a) Original Image



(c) Image thresholded using global value thresholding

Figure 3. Results from using the local thresholding algorithm

Figure 3 illustrates the application of the local thresholding algorithm on the image from the previous example. Figure 3b shows the balanced image (the original image - the local mean value array). Figure 3d shows the thresholded image that extracts the correct information.

7. Some Observations

1. If the illumination does not change over time, then the local thresholding algorithm does not need to be performed on each image. The image of thresholding values can be computed only once, eliminating much of the recurring processing time.

2. The implementation described in this application note is only a starting point. Other flavors of this approach can be customized for specific application requirements.

8. Implementation Pointers

The suggestions provided here are in reference to the example program available at the link provided below.

1. Provide a default mode where the whole image can be selected as the ROI.

2. When extracting the subimage confirm whether there is pixel information available in the border surrounding the ROI. For

ROIs that are closer to the edge of the image than the filter size, the border information is not completely available, so the algorithm must compensate for the missing information.

3. Before subtracting images, convert 8 bit images to 16 bit images to improve performance. The resulting (destination) image should also be 16 bits, because the pixel values can vary between -32,768 and 32,767.

4. When a mean value array is subtracted from an original image, the dynamic range of the resulting image is reduced. Apply the minimum and maximum intensity values to the subtracted image to increase its contrast.
http://sine.ni.com/apps/we/niepd_web_display.display_epd4?p_guid=C840B11531051CBCE034080020E74861&p_node=&p_source=external (http://zone.ni.com/devzone/cda/epd/p/id/3128)

Related Links:

Example Program for Adaptive Thresholding (http://zone.ni.com/devzone/cda/epd/p/id/3128)