Experiment (a) - texts from newspaper

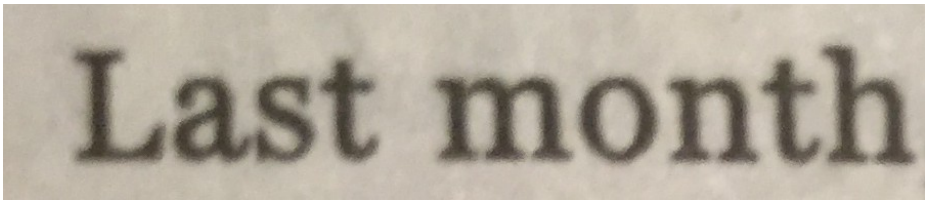


Image 1: Original image taken from mobile camera (cropped to a few words)

Image 2: Binarized image (using GIMP image editor, color->threshold, auto threshold)

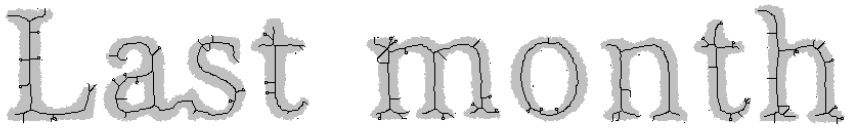


Image 3: Skeleton directly from image (2)

Image 4: Skeleton from a smoothed/cleaned image (with 3x3 filter)

The skeleton algorithm is implemented in Java, using the so called **Zhang-Suen Skeletonizing** algorithm. The **smoothing/cleaning** algorithm is reusing the output from exercise 2 (3 X 3 filter). The binarization is done by image editing tool with auto threshold.

From the result, it is concluded that the skeletonizing effect of image (4) is very good. All the major characteristics of the skeleton are kept and the features can be easy to extract.

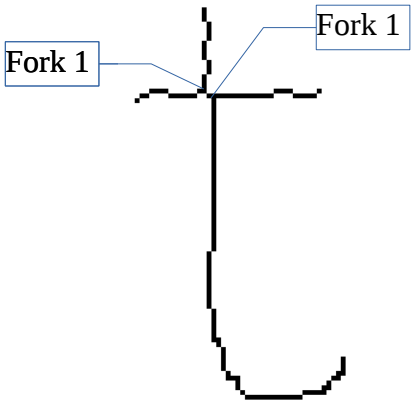
However if the image is not preprocessed with smoothing and cleaning, the skeletonizing effect is poor especially with those extended end-points from the glitches. Generally saying, in the Zhang-Suen algorithm, if there is a glitch of 2 pixels or more, then the glitch will be kept and extended till the center of the skeleton. By applying 3 x 3 smoothing filter, the 1 or 2 pixels' glitches are removed ahead so the final skeletons are smooth.

COMP 6731 Assignment #3 Guo Peng Liang 40012502

Following features can be extracted from the skeleton of the letters, so as for further classification:

- Straight lines – the continuous 8-connect neighbored points which can form approximately a straight line.
- Endpoints (location and direction) – points only have one 8-connect neighbors.
- Forks (location and direction) – points which have 3 8-connect neighbors.
  The direction of the forks can be eight directions (east, west, north, south, east-south, west-sourth, east-north, west-north).
  Some examples:

- Crosses (location and direction) – points which have 4 8-connect neighbors.
- Two very closed forks, which can be considered as a cross instead.
  An example is as below for the letter 't' in image (4):



Experiment (b) - images of fruits

Image 5: Original image taken from camera

A sample image is taken as below as the source of the experiment.

From experiment, it is found that there are couple of issues when doing the preprocessing and skeletonizing.

Issue 1: simple thresholding does not work.

Solution: create a gray-scale image from the difference of R-G-B channels.

The image of fruits are not able to be binarized using a threshold for the gray level. In the example of image 5, the shape of the fruits can not be got at all even manually selecting a threshold. This is because the gray level of the fruits is not dark enough to be differentiated from the background. This issue is due to the fact that the fruits are normally colorful but the grayscale image loses all color information.

An algorithm is taken as below, which has a good effect on the example. The basic idea is to calculate how colorful each pixel is by inspecting the difference of the three color channels.

For each pixel at (x, y), which color is represent as (r, g, b), where r, g, b is the intensity of red, green and blue channels (0 – 255), use either of the below two formulars to calculate a difference of the three colors:

$$gray(x,y) = average(r,g,b)$$

$$If\ taking\ the\ standard\ error: \quad diff(x,y) = \sqrt{\frac{(r-gray)^2 + (g-gray)^2 + (b-gray)^2}{3}}, \text{ or}$$

$$If\ taking\ the\ delta\ of\ max/min: \quad diff(x,y) = max(r,g,b) - min(r,g,b)$$

Then the output gray level of the pixel at (x, y) is the invert of the color-deviation.

$$gray'(x,y) = 255 - diff(x,y)$$

The output of this color-difference grayscale image is as below.

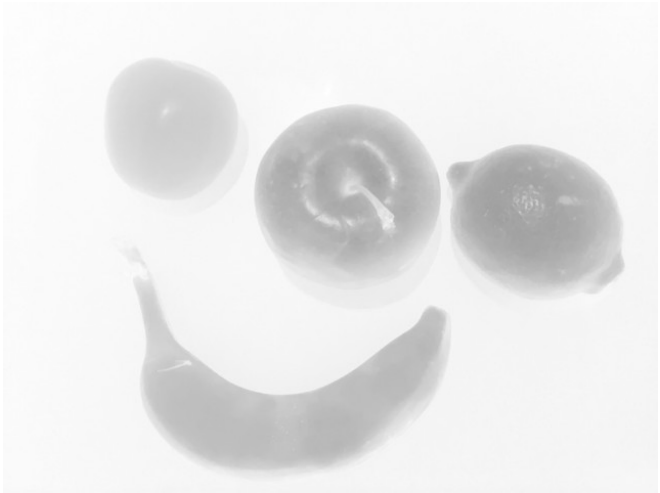


Image 7: Color-diff with standard error

Image 6: Color-diff with max-min

By taking a threshold to binary the images (6), following black-white images are got.

Image 8: Binarized using auto-threshold

Image 9: Binarized using manual threshold

The auto-threshold algorithm is not as good as a manual threshold. Some adaptive algorithm could be considered to improve the binarization.

Issue 2: the skeleton of the fruits does not reflect the shape very well.

Below is the skeleton after smoothing and cleaning image 9:

It is found that the skeleton is not regular, and does not reflect the shape very well.

This is mainly because:

- The shapes of the fruits are normally round, which does not really have a “skeleton”. As a result, a little difference in the local shape of the border would result to a big difference in the skeleton.
- There are high-light areas with great light reflection. The location of the high-light areas impacts the skeleton very much.

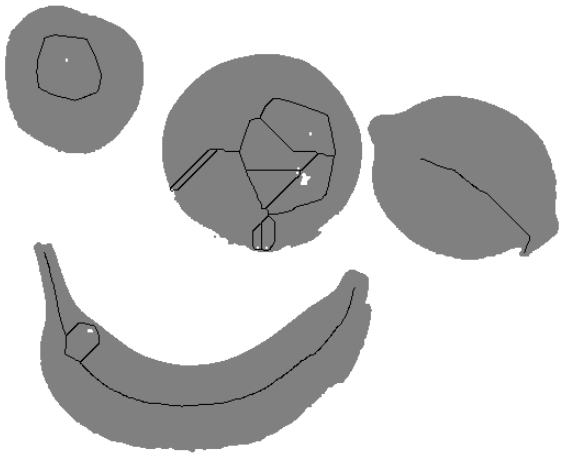


Image 10: Skeleton with the smoothed image from image 9

The conclusion is that the skeleton is not a good way for feature extraction of fruits. From my opinion, the better way is to get the features from the contours:

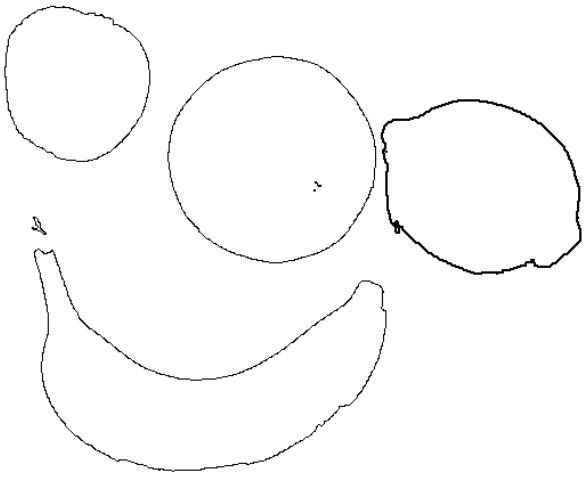


Image 11: Contours extracted from image (9)

The features can be extracted from the contours with below ideas:

- By measuring the width from different secants, and the distribution of the widths represent different shapes. An illustration is below:

- The color balance of the pixels within the contour is very important for recognizing the fruits. For example, balana shall have more red and green, tomator shall have more red, etc.

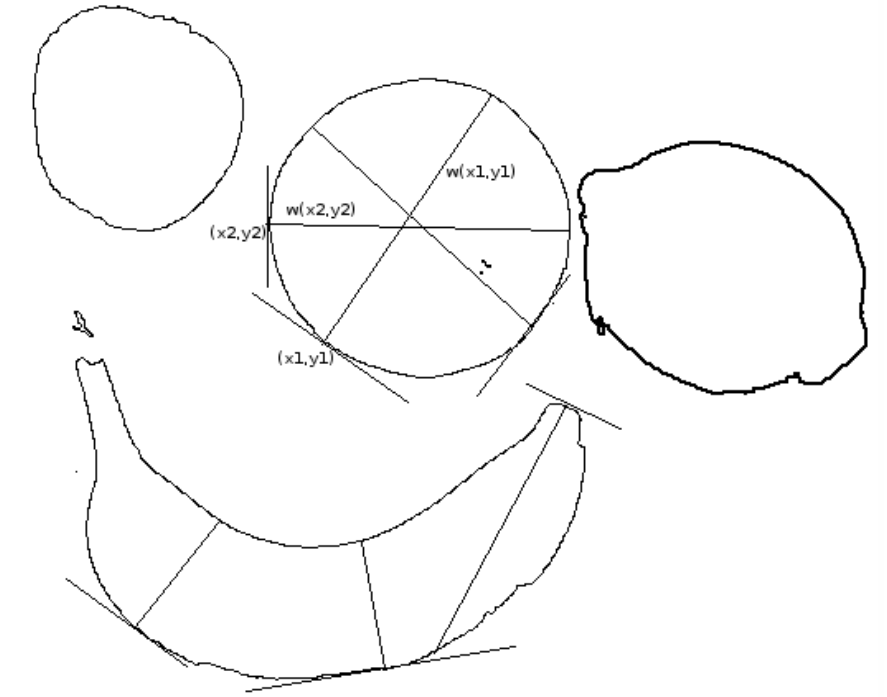


Image 12: Illustration of taking width as features

Description of the Program

The skeletonizing code is within the class ***feature.Skeleton***

The main function is in ***main.Assignment_3_main***.

The main function now does below tasks:

- Do and mark skeleton for the binary image of newspaper
- Generate a color-diff image from the image of fruit
- Do and mark skeleton for the binary image of fruits got from the colordiff image
- Do a contour tracing for the binary image of fruits.