**Image Processing**

**Analysis and Application**

**Table of Contents**

**Chapter 1 Introduction**…………………………………………………………….1

**Chapter 2 Review of Related Literature**…………………………......................7

2.1 Otsu’s Method………………………………………………………………22

2.1.1 Overview………………………………………………….………...7

2.1.2 Algorithm.........................................................................................10

2.2 Natural Color System……….………………………………………………22

2.3 Parallel Image Processing System…………………………………………22

2.3.1 Overview………………………………………………….………...7

2.3.2 System Architecture.......................................................................10

2.3.3 System Performance.......................................................................10

2.4 Canny Edge Detector…………………..……………………………………22

2.4.1 Overview………………………………………………….………...7

2.4.2 Algorithm.......................................................................10

2.4.3 Experiment and Analysis..................................................................10

2.4.3 Summary.......................................................................10

**Chapter 3 Preliminary Explorations**………………………………......................7

3.1 Otsu’s Method………………………………………………………………22

3.1.1 Implementation……………………………………………………12

3.1.2 Experiment and Analysis….…………………………………….....12

3.1.3 Algorithm Modifications and Improvements…………...…………12

3.1.4 Summary……….……………………………………………….....14

**Chapter 4 Novel Algorithms on Orange Grading System**……...………………22

4.1 Orange Grading System Architecture…..……………………………………22

4.2 RGB Color Class…………………...….……………………………………22

4.2.1 Experiment and Analysis………………………………….………...7

4.2.2 Derived Formula…….…………………………………….………...7

4.2.3 Algorithm…………………………………………………………...7

4.2.4 Implementation…….…………………………………….………...7

4.2.5 Advanced Data Analysis………………………………….………...7

4.2.5.1 Missing Class……………….…………………….……………7

4.2.5.2 Color Contrast…………….…………………….……………7

4.2.6 Summary…….…………………………………….………...7

4.3 Class Mean………..……………………………………………………….37

4.3.1 Overview………………………………………………….………...7

4.3.2 Algorithm…………………………………………………………...7

4.3.3 Implementation……………………………………………………...7

3.3.4 Algorithm Modifications and Improvements….………….………...7

4.3.5 Advanced Data Analysis…………………….………….………...7

4.3.5.1 Effects of Illumination Intensity Variations on Ripe Orange

Skin……………….…………………………………………….7

4.3.5.2 Effects of Illumination Intensity Variations on Unripe Orange Skin……………………………………………………………7

4.3.6 Summary......................................................................................…...7

4.4 Class Standard Deviation………..………………………………………….37

4.4.1 Overview………………………………………………….………...7

4.4.2 Algorithm……………………………………………….………...7

4.4.3 Implementation………………………………………….………...7

4.4.4 Summary………………………………………………….………...7

4.5 Class Distance……………….…..………………………………………….37

4.5.1 Overview………………………………………………….………...7

4.5.2 Algorithm……………………………………………….………...7

4.5.3 Implementation………………………………………….………...7

4.5.4 Summary………………………………………………….………...7

4.6 Closest Neighbor Class.…….…..………………………………………….37

4.6.1 Overview………………………………………………….………...7

4.6.2 Algorithm……………………………………………….………...7

4.6.3 Implementation………………………………………….………...7

4.6.4 Experiment and Analysis………………………………….………...7

4.6.5 Summary………………………………………………….………...7

4.7 Class Reclassification…………………………………………………….....37

4.7.1 Overview………………………………………………….………...7

4.7.2 Algorithm……………………………………………….………...7

4.7.3 Implementation………………………………………….………...7

4.7.4 Experiment and Analysis………………………………….………...7

4.6.5 Summary………………………………………………….………...7

4.8 Pixel Reclassification…………………………………………………….....37

4.8.1 Overview………………………………………………….………...7

4.8.2 Algorithm……………………………………………….………...7

4.8.3 Implementation………………………………………….………...7

4.8.4 Summary………………………………………………….………...7

**Chapter 5 Parallel Image Processing**......………………………………………...78

5.1 Overview…………………......………………………………………………8

5.2 Algorithm………………......………………………………………………8

5.3 Experiment and Analysis…....………………………………………………8

5.4 Summary…………………………………………………………………......8

**Chapter 2**

**Review of Related Literature**

**2.1 Otsu’s Method**

**2.1.1 Overview**

Thresholding is the simplest method of image segmentation. The core process in the thresholding is the choice of segmentation point. From a grayscale image, thresholding can be used to create binary images. Individual pixels in an image are marked as foreground pixels for instance if their value is greater than the segmentation point, as background pixels otherwise. Logical matrix contains only “0” and “1” can be used to represent an image. Typically the foreground pixel is given a value of “1”, and the background pixel is given a value of “0”. The segmentation point may be selected manually by user or computed automatically using a thresholding method. The mean or median value will work well to obtain a sufficient threshold value based on the most dominant pixel values in a noiseless image with uniform foreground and background pixel value, however this will generally not be the case.



***Fig.10. Simple Foreground Segmentation***

In machine vision and image processing, Otsu’s thresholding method is used to automatically perform histogram shape-based image thresholding, or converting a gray level image to a binary image. The histogram method assumes that there is some average value for the foreground and background pixels, but that the actual pixel values have some variation around these average values. Otsu’s thresholding method minimizes the weighted within-class variance and this turns out to be the same as maximizing the between-class variance. Otsu’s thresholding method operates directly on the gray level histogram and assumes that the image to be thresholded contains two classes of pixels(e.g. foreground and background). The difficult part during the process of clustering pixels is that two classes of pixels usually overlap, so minimizing the error of classifying a background pixel as a foreground becomes significant. Otsu’s thresholding method calculates the optimum threshold by separating those two classes, so that their combined spread(intra-class variance) is minimal. Otsu’s thresholding method makes each cluster as tight as possible and minimizing their overlap. Otsu’s thresholding method does not require much specific knowledge of the image, and is robust against image noise.

The cost of Otsu’s thresholding method is computationally cheap once the histogram is generated. However, the cost of Otsu’s thresholding method would be very expensive when extended to a multi-level threshold due to the fact that a large number of iterations are required for computing the cumulative probability and the mean of a class. In real time application, most of the methods suffer from time-consuming computation for multilevel thresholding. TSMO thresholding method is a two–state multithreshold Otsu method which can significantly improve the efficiency with an accuracy equivalent to Otsu’s method by greatly reducing the iterations required for computing the between-class variance in a gray image.

The way to adjust the threshold is to increase the spread of one class and decrease the spread of the other. The gold then is to select the threshold that minimizes the combined spread. The within-class variance as the weighted sum of the variances of each cluster is defined as:

\sigma^2_w(t)=\omega_1(t)\sigma^2_1(t)+\omega_2(t)\sigma^2_2(t)

|  |  |
| --- | --- |
| **T** | **Threshold** |
| **ω*i*** | **Class Probabilities** |
| **\sigma^2_ i** | **Variances of two classes** |

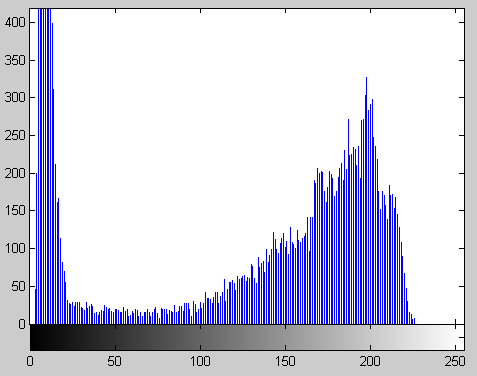
Otsu’s thresholding method minimizes the weighted within-class variance and this turns out to be the same as maximizing the between-class variance which is defined as:

\sigma^2_b(t)=\sigma^2-\sigma^2_w(t)=\omega_1(t)\omega_2(t)\left[\mu_1(t)-\mu_2(t)\right]^2

|  |  |
| --- | --- |
| **ω*i*** | **Class Probabilities** |
| **μ*i*** | **Class Means** |

**2.1.2 Algorithm**

1. Initialize grey scale ranging from black(0) to white(L-1). L is possible values(or levels) for a pixel and often chosen to be 256.
2. Initialize class probability(**ω*1***) of class one to be 0.
3. Initialize the maximum or minimum between-class variance to be 0.
4. Find the maximum and minimum grey level separately in a given grey scale image.
5. Compute image histogram for a given grey scale image. The distribution of values for the pixels in an image is called the image histogram. For example, the following histogram may be for a grey scale image with L=235 and N=25050. Relatively speaking, this image has a large number of white pixels with few black pixels.

****

***Fig.10. Histogram Analysis***

1. Compute grey levels probabilities *P****i*** for each grey level based on the image histogram computed in Step 5.

*P****i*** *=**ni / N, P****i*** *≥ 0,* ∑*P****i*** = 1, 0*≤ i ≤ L*

niis the number of pixels at grey level i. N is the total number of pixels in a given grey scale image.

1. Compute grey levels mean *M* for a given grey scale image.

*M* = ∑(*P****i*** *\* i),* 0*≤ i ≤ L*

1. Step through all possible thresholds *t* = 1…maximum grey level. The assumption is that the pixels in a given grey scale image are divided into two classes, class one and class two(foreground and background) by a threshold *t*. Class one denotes pixels with grey levels from 1 to *t,* and class two denotes pixels with grey levels from *t*+1 to maximum grey level*.*
2. Compute the class probabilities of class one and class two separated by a threshold t.
   1. Compute class probability for class one.

*ω1 =* ∑*P****i****,* 0*≤ i ≤ t*

Sum up all the grey levels probabilities *P****i*** from grey level 0 to *t*.

1. Compute class probability for class two.

*ω2 =* ∑*P****i****, t+1≤ i ≤ maximum grey level*

*or*

*ω2 = 1 - ω1*

Sum up all the grey levels probabilities from grey level *t+1* to maximum grey level, alternatively this is equivalent to subtract the class probability of class one from 1.

1. Compute the class means of class one and class two separated by a threshold t.
2. Compute class mean for class one.

*μ1 =* ∑(*i \* P****i****) / ω1,* 0*≤ i ≤ t*

Multiply each grey level *i* by its corresponding grey levels probability from grey level 0 to *t.*

Sum up all the multiplications and then divide by the class probability of class one *ω1.*

1. Compute class mean for class two.

*μ2 =* ∑(*i \* P****i****) / ω2,*

*t+1≤ i ≤ maximum grey level*

*or*

*μ2 = (M -* ∑(*i \* P****i****)) / ω2,  0 ≤ i ≤ t*

Multiply each grey level *i* by its corresponding grey levels probability from grey level *t+1* to maximum grey level*.* Sum up all the multiplications and then divide by the class probability of class two *ω2.*

1. Compute weighted sum of variances of class one and class two separated by a threshold *t*.

1. Compute class variances for class one.

*σ12* = ∑*(i -μ1)2 \* P****i /*** *ω1*, 0*≤ i ≤ t*

2*.* Compute class variances for class two.

*σ22* = ∑*(i –μ2)2 \* P****i /*** *ω2*, *t+1≤ i ≤ maximum grey level*

3. Compute weighted sum of variances of two classes.

*σw2 = ω1 \* σ12 + ω2 \* σ22,*

minimizing the between-class variance

*or*

*σb2 = ω1 \* ω2 \* (μ1 -μ2)2*,

maximizing the between-class variance

1. Desired threshold corresponds to the minimum *σw2* or maximum *σb2*.

This algorithm is implemented in the next chapter.

**2.2 Natural Color System**

Color is the visual perceptual property corresponding in humans to the categories called red, yellow, blue and others. Spectrum of light is the distribution of light energy versus wavelength. Color derives from the spectrum of light interacting in the eye with the spectral sensitivities of the light receptors. Color can be represented as tuples of numbers, typically as three or four values or color components (e.g. RGB and CMYK are color models). The way of describing colors in such an abstract mathematical model is called color space.

|  |  |
| --- | --- |
| **RGB** | **CMYK** |
|  |  |

*Fig.10. RGB and CMYK Color Space*

Human eyes are sensitive to three additive primary colors, red, green, and blue. In machine vision, RGB(**R**ed **G**reen **B**lue) color space is one of the most popular additive color systems which are derived from human perception of color. It is called an additive color system, since you could add light from the primary colors to make new colors. The human visual system is able to differentiate between 100-200 grey levels and 30,000-50,000 colors. For this reason most vision systems use 256 grey levels(8 bits per pixel) or 256 levels of Red, Green and Blue(24 bits per pixel). The 256 levels of primary usually do not represent equally spaced intensities, due to gamma correction. RGB color space is the most common way to encode color in computing for sensing, representation, and display of images. Red, green, and blue light are added together in various ways to reproduce a broad array of colors, more than 16 million. Every single color that we can see is made up of some combination of these three colors, e.g.

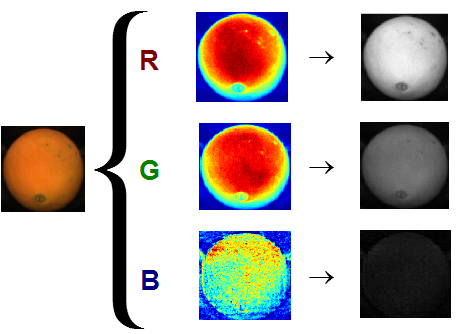
* Pure black is derived by removing all three primary colors completely.
* Pure white is derived by mixing fully saturated versions of all three primary colors together.
* A shade of gray is derived by mixing equal amounts of all three primary colors at some percentage between zero to one hundred percent.

The following chart shows using combinations of these three primary colors to reproduce the colors around the center of the spectrum.

|  |  |
| --- | --- |
| **RGB Color Space** | |
|  |  |

*Fig.10. RGB Color Space*

A color image is made of pixels, and pixels are made of different combination of primary colors. In machine vision, these primary colors are expressed as channels which are used to refer to a certain component of an image in a conventional term. For instance, RGB color space has three channels, red, green, and blue channel. A channel is made of the one of these primary colors. It is a grayscale image with the same size as the original color image. In a grayscale image, each pixel only carries intensity information. The intensity of a pixel is expressed within a given range between a minimum and maximum, inclusive. For instance, the weakest intensity is black(0), the strongest intensity is white(255) and many shades of gray in between. The following is an example of color channel splitting of a full RGB color image. The column at left shows the isolated color channels in natural colors, while at right there are their grayscale equivalences:



*Fig.10. Isolated Color Channels*

**2.3 Parallel Image Processing System**

**2.3.1 Overview**

Parallel Image Processing(PIP) has been a topic for many years. The basic idea is to use multi-processors to perform a single task or multiple tasks at the same time. The maximum speedup is n with n processors. But in practice, it can’t be achieved according to Amdal’s Law. The serial section is constant, such as reading data from a CD.

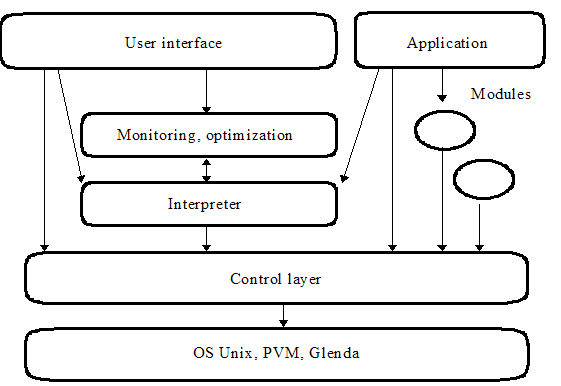
Nowadays large amount data are required for image processing, the traditional single processor is not able to complete complex tasks within a reasonable time. So the idea of parallel image processing comes out. To be able to use multi-processors at the same time, the parallel environment has to be set up first. In Massey university, students can use maximum 8 processors at the same time, and use Message Passing Interface(MPI) sending and receiving messages between processors.

Parallel System is good, but not perfect. The main issue of the Parallel System is the communication time between processors. In the worse case, communication time can dominate the whole execution time. In some cases, there is no point to use Parallel System such as the input data size is small.

Finally, Parallel Image Processing is a very interesting topic, various problems will be identified this thesis.

**2.3.2 System Architecture**

A well developed Parallel Image Processing System will be introduced in this section.

******

*Fig.10. PIP System*

The whole system consists of six logical layers, System layer, Control Layer, Functional Layer, Shell Layer, Optimization Layer, and Application Layer. Each layer performs a specific task.

1. System Layer: *parallel environment (like stcluster)*
2. Control Layer: *communication between processors (like MPI)*
3. Functional Layer: *image processing algorithm, such as zero crossings of laplacian*
4. Shell Layer: *user interface*
5. Optimization Layer: *monitoring the system and optimize the performance*
6. Application Layer: *various image processing task, such as blur or sharpen an input image*

The Parallel Image Processing System can be divided into two parts, applications and modules. Functional modules can be implemented in parallel.

The Parallel Image Processing system uses master-worker parallel programming to control modules by applications:

* An application (master) sends commands to the system.
* The commands are stored in shared memory.
* The functional modules (workers) take these commands, perform requested task and return the results back to

the application.

* One application can send another command before getting results of the previous one.

Each functional module may internally consist of many parallel processes.

* It will be very efficient if the function provided by the module can be paralleled.
* E.g. A segmentation module can have several processes (sub modules). Each of them does the segmentation in a part of the image. Then the module joins the results of all the sub modules.

Two ways to reduce the unnecessary communication:

1. A shared database of data objects, not application.

2. Sending data directly from one module to the other (pipelining).

The shared memory is implemented as a shared associative memory,

* E.g. The data is accessed by a key value, not address.
* Provided by the coordination language Linda.
* The Parallel Image Process System is designed to use Linda implementation called Glenda as a basic parallel environment.

**2.3.3 System Performance**

The following is an experiment to find out what speedup can be reached by parallelizing a simple convolution operation.

* The size of convolution mask: 5 \* 5
* The size of two input images: 256 \* 256 and 512 \*512.
* Measure the execution time in seconds.
* Each input image was tested 3 times on the same number of processors.

It is very important to test each input image on the same number of processors more than once. If the execution time varies in a very small range, then accept the result and calculate the average execution time. A few factors may affect the total execution time, such as memory accessing speed. Loading the same image is normally faster in the second time.

Input image size 256 \* 256 (small)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Processors | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Test 1 | 3.2000 | 1.9500 | 1.7500 | 1.5800 | 1.5600 | 1.6300 | 1.8200 | 1.9500 |
| Test 2 | 3.2000 | 1.9500 | 1.6700 | 1.5500 | 1.6700 | 1.7000 | 1.8100 | 2.0600 |
| Test 3 | 3.2500 | 1.9700 | 1.6700 | 1.5300 | 1.6200 | 1.6400 | 1.8000 | 1.9000 |

*Fig.10. Experiment One, Image Size 256\*256*

Execution time decreases when the number of processors increases. Running on 4 processors achieved the best result. Running on 8 processors is even slower than running on 4 processors, because the communication time takes longer.

Input image size 512 \* 512 (large)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Processors | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Test 1 | 11.3000 | 6.6500 | 5.5800 | 4.8500 | 4.2500 | 3.8800 | 3.7600 | 3.5400 |
| Test 2 | 11.7000 | 6.6800 | 5.5900 | 4.9300 | 4.3400 | 3.9000 | 3.7100 | 3.4900 |
| Test 3 | 11.3600 | 6.7200 | 5.6000 | 4.8600 | 4.3200 | 3.9000 | 3.7300 | 3.5300 |

*Fig.10. Experiment Two, Image Size 512\*512*

Execution time decreases when the number of processors increases. Running on 8 processors achieved the best result.

But there is no big difference on using 6, 7 and 8 processors.

*Fig.10. Comparison between Experiment 1 and 2*

Parallel algorithm is more efficient for larger input image size. There is an upper bound on the number of processors used. The communication between processors is very time consuming. The more processors you use, the more communication time is going to be taken. Notice the maximum speedup is not n with n processors, it is not linear speed up.

**2.4 Canny Edge Detection**

**2.4.1 Overview**

Edge detection is the process of finding sharp contrasts in intensities in an image. This process significantly reduces the amount of data in the image, while preserving the most important structural features of that image.Canny Edge Detection is considered to be the ideal edge detection algorithm for images that are corrupted with white noise.Canny Edge Detector is well known for its ability to generate single-pixel thick continuous edges. The Canny Edge Detector arises from the earlier work of Marr and Hildreth, who were concerned with modeling the early stages of human visual perception.Canny concentrated an ideal step edge, represented as a Sign function in one dimension, corrupted by an assumed Gaussian noise process.

Canny’s ideas and methods can be found in his paper, *"A Computational Approach to Edge Detection"*. He followed a list of criteria to improve current methods of edge detection.

1. The first and most obvious is low error rate. It is important that edges occurring in images should not be missed and that there be no responses to non-edges.
2. The second criterion is that the edge points be well localized. In other words, the distance between the edge pixels as found by the detector and the actual edge is to be at a minimum.
3. A third criterion is to have only one response to a single edge. This was implemented because the first two steps were not substantial enough to completely eliminate the possibility of multiple responses to an edge.

The three goals of Canny Edge Detection are:

1. Good Detection: The ability to locate and mark all real edges.
2. Good Localization: Minimize distance between the detected edge and real edge.
3. Clear Response: Only one response per edge.

**2.4.2 Algorithm**

Canny’s algorithm is based strongly on convolution of the image function with Gaussian operators and their derivatives.

1. Laplacian of a Gaussian

* The Laplacian locates edges, but it does not differentiate between edges of different widths.
  + If the image is blurred by a Gaussian then the Laplacian will only detect wide edges.
  + These two operations are usually combined by finding the Laplacian of a Gaussian operator.

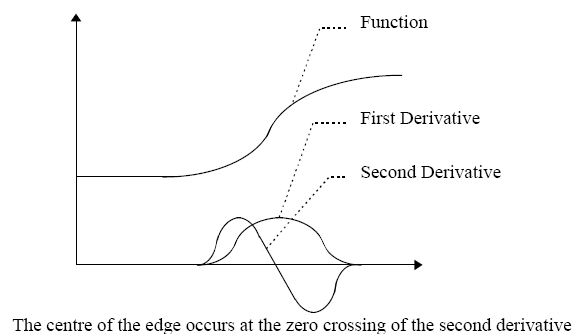
******

1. First and Second Derivative

First Derivative: ******

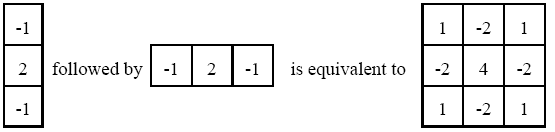
Second Derivative: ******

* The first and second derivatives are often used in edge detection.
* Any place where the first derivative reaches the peak will be on an edge.
* If the position of the edge is required then the second derivative can be used. Any place where the second derivative crosses zero will be on an edge.

******

*Fig.10. First and Second Derivatives*

1. Convolution Masks
   * Two small convolution masks can be used to replace one large mask.

******

*Fig.10. Convolution Mask*

* If a mask can be split into two one dimensional masks it is said to be separable.
* In order to be separable each row in the mask must be a multiple of all other rows and each column must be a multiple of all other columns.
* Using separable masks it is possible to use large neighborhood sizes because the number of computations is proportional to the width of the neighborhood and not its area.

Canny edge detection includes the following four stages:

1. Stage One: Image Smoothing
2. Stage Two: Image Differentiation
3. Stage Three: Non-maximum Suppression
4. Stage Four: Edge Thresholding
5. Image Smoothing

* The first step is to filter out any noise in the original image before trying to locate and detect any edges.
* Calculate a suitable convolution mask.
* Perform Gaussian smoothing using standard convolution methods.
* In practice, two dimensional convolution with large Gaussians takes longer time. It is common to approximate this by two one dimensional Gaussians, one aligned with the x-axis, the other with the y axis.

1. Image Differentiation

* The values in the x-smoothed image array are convolved with a first derivative of a one dimensional Gaussian of identical http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/edges/img27.gif aligned with y.
* The values in the y-smoothed image array are convolved with a first derivative of a one dimensional Gaussian of identical **http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/edges/img28.gif**aligned with x.
* From the computed x and y gradient values, the magnitude and angle of the slope can be calculated.

1. Non-maximum Suppression
   * Having found the rate of intensity change at each point in the image, edges must be placed at the points of maxima or non-maxima must be suppressed.
   * A local maximum occurs at a peak in the gradient function.
   * Suppress non-maxima perpendicular to the edge direction.
2. Edge Thresholding

* The thresholder used in the Canny operator uses a method called "hysteresis". Most thresholders used a single threshold limit.
* If the edge values fluctuate above & below this value, the line will appear broken (streaking).
* Hysteresis counters streaking by setting an upper & lower edge value limit.
* Accepted if a value lies above the upper threshold limit.
* Rejected if a value lies below the lower threshold limit.
* Points which lie in between are accepted if they connected to pixels which exhibit strong response.

**2.4.3 Experiment and Analysis**

Major Steps:

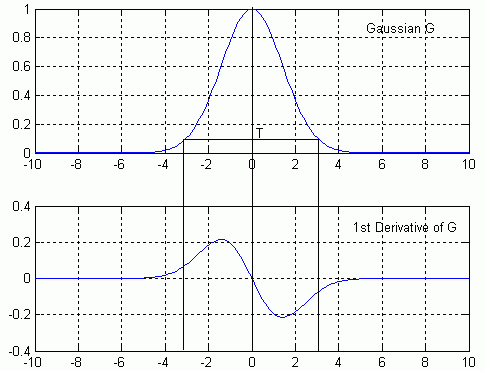
* Generation of Masks
* Applying Mask to Images
* Non-Maxima Suppression
* Hysteresis Thresholding

Input Data:

* Image *I*
* Value of Smoothing Parameter Sigma
* High Threshold *Th*
* Low Threshold *TI*

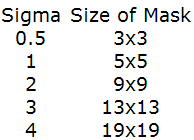
1. Generation of Masks

* This module requires the value of sigma as an input and generates x- and y-derivative masks as output.
* In Canny's method, the masks used are 1st derivative of a Gaussian in x- and y-directions.

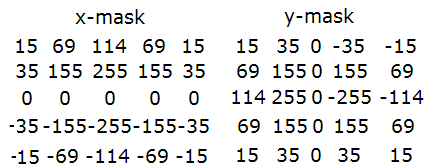
******

*Fig.10. Gaussian and 1st Derivative*

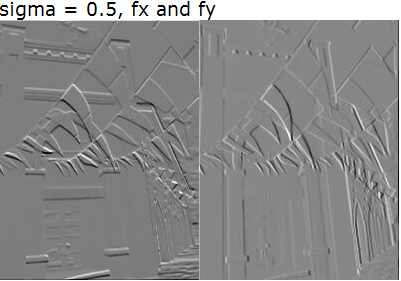
* + Choose a reasonable mask size based on analyzing the Gaussian and applying a threshold *T*. *T* is a real-number between zero and one.
  + If the mask size is too large, then unnecessary computations will be involved during convolution.
  + If the mask size is too small, then features will not be detected.
* The mask size is *2\*sHalf + 1* to incorporate both positive and negative sides of the mask.
  + *sHalf* is computed by finding the point on the curve where the Gaussian value drops below T.
    - *exp(-x^2/(2\*sigma^2) = T*
    - *sHalf = round(sqrt(-log(T) \* 2 \* sigma^2)).*
* The values of mask size obtained by this method for various sigma values are shown below:



* Scaled masks for *sigma = 1* are shown below:

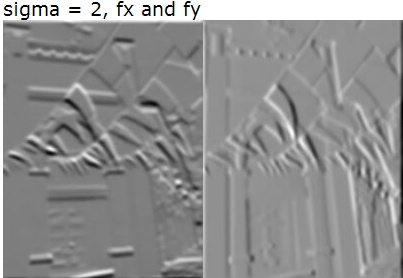
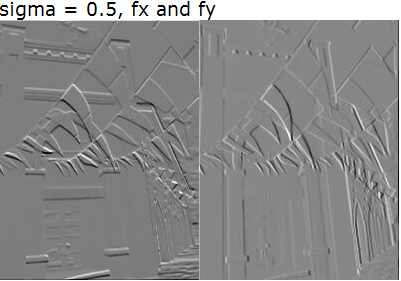
****

1. Applying Masks to Images
   * The masks are applied to the images using convolution. The result is scaled down by the same factor which was used to scale up the masks.
   * To write output to image files, the min and max values are scaled to *0* and *255* respectively.



*Fig.10. Apply Masks to Image*

* The effect of increasing sigma is obvious from the convolved images below:
  + The gradients are much smoother
  + horizontally running edges are identified in *fx*
  + vertical running edges are identified in *fy*



*Fig.10. Increasing Sigma*

* Gradient magnitude, *M*, is computed from *fx* and *fy* images, using the magnitude formula.
  + The effect of increasing sigma is dramatically highlighted in these two examples

******

*Fig.10. Effect of Increasing Sigma*

* Gradient Direction, Phi, is computed using atan2 function.
  + *atan2* returns in the range of *-pi* to *pi*.
  + Convert the angle returned by atan2 function to degrees and add 180 to get an output range of 0-360 degrees.
* Visualizing gradient direction in a meaningful fashion is difficult. Raw gradient direction, if mapped to the range of 0-255 looks as follows:



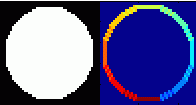
*Fig.10. Raw Gradient Direction*

* A better technique is to view only those values of direction for which gradient magnitude is high enough
  + The output is visible only at significant edges.
  + Edges with similar direction are colored similarly.
  + Each direction has two colors associated with it.

******

*Fig.10. High Gradient Magnitude*

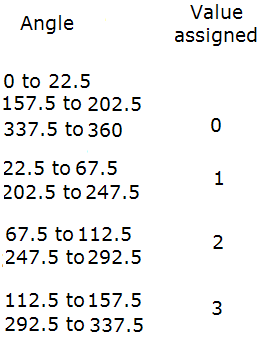
* Visualizing gradient direction can be used on a circle image to show edges in different directions have different values of *Phi.*
  + For a white circle on a black background.
* This clearly indicates that Phi is different for edges running in different directions.

******

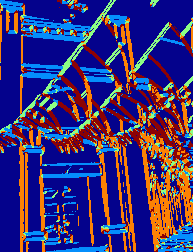
*Fig.10. Edge on Circle Image*

1. Non-Maxima Suppression

* Non maxima suppression step makes all edges in *M* one pixel thick. The first step is to quantize gradient direction into just four directions.

******

* Out of the eight discredited directions, four opposing pairs are unique in terms of picking neighborhood pairs in a straight line around each edge point.
* Dark blue shows no edge, and four other colors show four quantized directions.

******

*Fig.10. Quantized Direction*

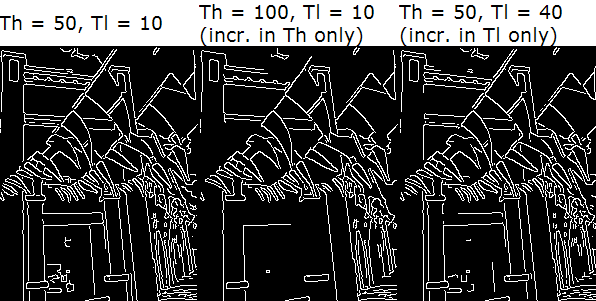
* Pick two neighbors of each edge point along the gradient direction. Gradient direction is perpendicular to the edge, so this is the direction searching for edge points.
  + For example, for quantized direction zero the gradient direction could be less than zero degree, meaning the edge is a horizontal edge. Therefore the two neighbors that need to be picked for comparison are the north and south neighbors, denoted in code by *(r-1, c)* and *(r+1,c)* respectively
  + If the edge point both these neighbors, then it is maintained in M otherwise it is made zero.
* The output before and after Non-Maxima Suppression for *sigma = 2* is shown below:
  + Each edge is one pixel thick, but has different values of gradient magnitude based on its strength.

******

*Fig.10. Before and After Non-Maxima Suppression*

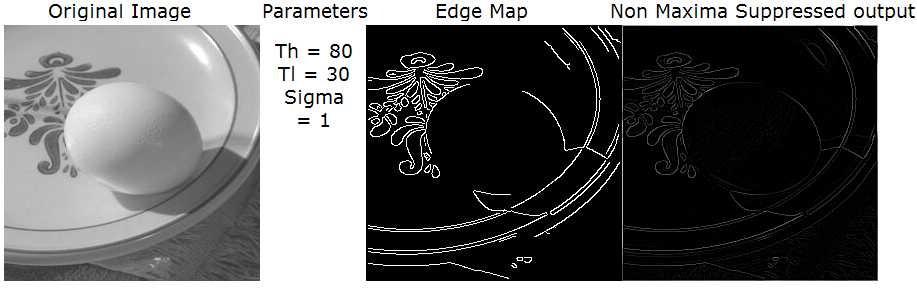
1. Hysteresis Thresholding
   * The final step in Canny's edge detection algorithm is to apply two thresholds to follow edges.
   * Set the border pixels to zero, so finding neighbors does not go out of bounds of the image. Scan the image from left to right, top to bottom.
   * The first pixel in non-maxima suppressed magnitude image which is above a certain threshold, *Th*, is declared an edge.
   * All its neighbors above threshold *Tl* are marked as an edge.
   * A visited map is maintained so that recursion does not loop infinitely
   * Two stopping conditions:
     + A neighbor is below *Tl.*
     + A neighbor has already been visited.

* If *Th* is increased but *Tl* remains the same, lesser edge components will be detected, but their lengths will be the same.
* If *Tl* is increased but *Th* is the same, then the same number of edge components will be detected but their lengths will be shorter.

******

*Fig.10.* Hysteresis Thresholding

*The following examples are results on other images. It is very difficult to obtain the complete closed edge of the egg, because the edge in the middle portion is very weak.*

******

*Fig.10. Example of Edge Detection*

**2.4.4 Summary**

There are many ways to perform edge detection. The majority of different methods may be grouped into two categories, gradient and Laplacian.

* The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image
* The Laplacian method searches for zero crossings in the second derivative of the image to find edges.

Sobel is one of the well known edge detection techniques.

The Canny Edge Detector defines edges as zero-crossings of second derivatives in the direction of greatest first derivative. It is based strongly on convolution of the image function with Gaussian operators and their derivatives. It is the ideal edge detection algorithm.

**Chapter 3**

**Preliminary Explorations**

**3.1 Otsu’s Method**

**3.1.1 Implementation**

The algorithm of otsu’s thresholding method is fully implemented in the following example section. Subset of a grey scale image is selected to be the sample data for demonstration purpose. The sample data is a two dimensional array of picture elements which is used to make up an image. It has a width(W=6) and height(H=6). The total number of pixels in this sample data(N=36) is WxH(6X6).

|  |
| --- |
| **Sample Data** |
|  |

***Fig.10. Sample Data***

1. Initialize grey scale ranging from 0 to 255(L-1). L is chosen to be 256.

*L = 256*

2. Initialize class probability(**ω*1***) of class one to be 0.

***ω1*** *= 0*

**3.** Initialize the maximum or minimum between-class variance to be 0;

*Minσw2 = 0,*  minimizing the between-class variance

*or*

*Maxσb2 = 0,* maximizing the between –class variance

4. Find the maximum and minimum grey level separately.

*MinGreyLevel = 0*

*MaxGreyLevel = 220*

5. Compute image histogram which is the distribution of values for the pixels in the sample data.

|  |
| --- |
| **Histogram** |
|  |
|  |

***Fig.10. Histogram Analysis***

Index starting value of an excel spreadsheet is 1. So the first column indexed 1 is corresponding to grey level 0. It can be interpreted as there are 8 pixels at grey level 0 in the sample data.

6. Compute grey levels probabilities *P****i*** for each grey level.

*P****0*** *=**n0 / N, 0.2222 = 8 / 36, at grey level 0*

*P****1*** *=**n1 / N, 0.1111 = 4 / 36, at grey level 1*

*P****2*** *=**n2 / N, 0 = 0 / 36, at grey level 2*

**........ , .............., ..............,**

**........ , .............., ..............,**

**........ , .............., ..............,**

*P****255*** *=**n255 / N, 0 = 0 / 36, at grey level 255*

|  |
| --- |
| **Probability** |
|  |
|  |

***Fig.10. Probability Analysis***

7. Compute grey levels mean *M.*

*M* = ∑(*P****i*** *\* i) = (0.2222 \* 0) +*

*(0.1111 \* 1) +*

*(0 \* 2) +*

*...... +*

*+ (0 \* 255)*

*= 35.6667, 0≤ i ≤ L*

8. Step through all possible thresholds. The algorithm used is to maximize the between-class variance.

*t = [MinGreyLevel...MaxGreyLevel] = [0...220]*

* **At *t = 0, 0 ≤ i ≤ t***

*ω1 =* ∑*P****i*** *= P****0*** *= 0.2222*

*ω2 = 1 - ω1 = 1 – 0.2222 = 0.7778*

*temp =* ∑(*i \* P****i****) = 0 \* 0.2222 = 0*

*μ1 =* ∑(*i \* P****i****) / ω1 = temp /ω1*  *= 0 / 0.2222 = 0*

*μ2 = (M -* ∑(*i \* P****i****)) / ω2 = (M - temp) / ω2*

*=(35.6667 - 0) / 0.7778 = 45.8559*

*σb2 = ω1 \* ω2 \* (μ1 -μ2)2*

*= 0.2222 \* 0.7778 \* (0 – 45.8559)2 = 363.4147*

* **At *t = 1, 0 ≤ i ≤ t***

*ω1 =* ∑*P****i*** *= P****0*** *+ P****1*** *= 0.2222 + 0.1111 = 0.3333*

*ω2 = 1 - ω1 = 1 – 0.3333 = 0.6667*

*temp =* ∑(*i \* P****i****) = 0 \* 0.2222 + 1 \* 0.1111 = 0.1111*

*μ1 =* ∑(*i \* P****i****) / ω1 = temp /ω1*  *= 0.1111 / 0.3333*

*= 0.3333*

*μ2 = (M -* ∑(*i \* P****i****)) / ω2 = (M - temp) / ω2*

*=(35.6667 – 0.1111) / 0.6668 = 45.8559 = 53.3307*

*σb2 = ω1 \* ω2 \* (μ1 -μ2)2*

*= 0.3333 \* 0.6667 \* (0.3333 – 53.3307)2 = 624.1298*

* **At *t = 2, 0 ≤ i ≤ t***

*ω1 =* ∑*P****i*** *= P****0*** *+ P****1*** *+**P****2*** *= 0.2222 + 0.1111 + 0 = 0.3333*

*ω2 = 1 - ω1 = 1 – 0.3333 = 0.6667*

*temp =* ∑(*i \* P****i****)*

*= 0 \* 0.2222 + 1 \* 0.1111 + 2 \* 0 = 0.1111*

*μ1 =* ∑(*i \* P****i****) / ω1 = temp /ω1*  *= 0.1111 / 0.3333*

*= 0.3333*

*μ2 = (M -* ∑(*i \* P****i****)) / ω2 = (M - temp) / ω2*

*= (35.6667 – 0.1111) / 0.6668 = 45.8559 = 53.3307*

*σb2 = ω1 \* ω2 \* (μ1 -μ2)2*

*= 0.3333 \* 0.6667 \* (0.3333 – 53.3307)2 = 624.1298*

* **..................................................**
* **..................................................**

9. This step is equivalent to step 8. The algorithm used is to minimize the between-class variance. Step through all possible thresholds.

*t = [MinGreyLevel...MaxGreyLevel] = [0...220]*

* **At *t = 0, 0 ≤ i ≤ t***

*ω1 =* ∑*P****i*** *= P****0*** *= 0.2222*

*ω2 = 1 - ω1 = 1 – 0.2222 = 0.7778*

*temp =* ∑(*i \* P****i****) = 0 \* 0.2222 = 0*

*μ1 =* ∑(*i \* P****i****) / ω1 = temp /ω1*  *= 0 / 0.2222 = 0*

*μ2 = (M -* ∑(*i \* P****i****)) / ω2 = (M - temp) / ω2*

*= (35.6667 - 0) / 0.7778 = 45.8559*

*temp1 =* ∑*(i -μ1)2 \* P****i*** *= (0 - 0)2 \*0.2222 = 0*

*temp2 =* ∑*(i –μ2)2 \* P****i,******t+1 ≤ i ≤ MaxGreyLevel***

*= (1 – 45.8559)2 \* 0.1111 +*

*(2 – 45.8559)2 \* 0 +*

*...... ...... ...... +*

*(220 – 45.8559)2 \* 0.0278*

*= 2181.9*

*σ12* = ∑*(i -μ1)2 \* P****i /*** *ω1 = temp1* ***/*** *ω1*

*= 0 / 0.2222 = 0*

*σ22* = ∑*(i –μ2)2 \* P****i /*** *ω2 = temp2* ***/*** *ω2*

*= 2181.9 / 0.7778 = 2805.2*

*σw2 = ω1 \* σ12 + ω2 \* σ22*

*= 0.2222 \* 0 + 0.7778 \* 2805.2*

*= 2181.9*

* **At *t = 1, 0 ≤ i ≤ t***

*ω1 =* ∑*P****i*** *= P****0*** *+ P****1*** *= 0.2222 + 0.1111 = 0.3333*

*ω2 = 1 - ω1 = 1 – 0.3333 = 0.6667*

*temp =* ∑(*i \* P****i****) = 0 \* 0.2222 + 1 \* 0.1111 = 0.1111*

*μ1 =* ∑(*i \* P****i****) / ω1 = temp /ω1*  *= 0.1111 / 0.3333*

*= 0.3333*

*μ2 = (M -* ∑(*i \* P****i****)) / ω2 = (M - temp) / ω2*

*=(35.6667 – 0.1111) / 0.6668 = 45.8559 = 53.3307*

*temp1 =* ∑*(i -μ1)2 \* P****i*** *= (0 – 0.3333)2 \*0.2222 +*

*(1 – 0.3333)2 \* 0.1111*

*= 0.0741*

*temp2 =* ∑*(i –μ2)2 \* P****i,******t+1 ≤ i ≤ MaxGreyLevel***

*= (2 – 53.3307)2 \* 0 +*

*(3 – 53.3307)2 \* 0 +*

*...... ...... .... +*

*(220 – 53.3307)2 \* 0.0278*

*= 1921.1*

*σ12* = ∑*(i -μ1)2 \* P****i /*** *ω1 = temp1* ***/*** *ω1*

*= 0.0741 / 0.3333 = 0.2223*

*σ22* = ∑*(i –μ2)2 \* P****i /*** *ω2 = temp2* ***/*** *ω2*

*= 1921.1 / 0.6667 = 2881.5059*

*σw2 = ω1 \* σ12 + ω2 \* σ22*

*= 0.3333 \* 0.2223 + 0.6667 \* 2881.5059*

*= 1921.2*

* **At *t = 2, 0 ≤ i ≤ t***

*ω1 =* ∑*P****i*** *= P****0*** *+ P****1*** *+**P****2*** *= 0.2222 + 0.1111 + 0 = 0.3333*

*ω2 = 1 - ω1 = 1 – 0.3333 = 0.6667*

*temp =* ∑(*i \* P****i****)*

*= 0 \* 0.2222 + 1 \* 0.1111 + 2 \* 0 = 0.1111*

*μ1 =* ∑(*i \* P****i****) / ω1 = temp /ω1*  *= 0.1111 / 0.3333*

*= 0.3333*

*μ2 = (M -* ∑(*i \* P****i****)) / ω2 = (M - temp) / ω2*

*= (35.6667 – 0.1111) / 0.6668 = 45.8559 = 53.3307*

*temp1 =* ∑*(i -μ1)2 \* P****i*** *= (0 – 0.3333)2 \*0.2222 +*

*(1 – 0.3333)2 \* 0.1111 +*

*(2 – 0.3333)2 \* 0*

*= 0.0741*

*temp2 =* ∑*(i –μ2)2 \* P****i,******t+1 ≤ i ≤ MaxGreyLevel***

*= (3 – 53.3307)2 \* 0 +*

*(4 – 53.3307)2 \* 0.0278 +*

*...... ...... .... +*

*(220 – 53.3307)2 \* 0.0278*

*= 1921.1*

*σ12* = ∑*(i -μ1)2 \* P****i /*** *ω1 = temp1* ***/*** *ω1*

*= 0.0741 / 0.3333 = 0.2223*

*σ22* = ∑*(i –μ2)2 \* P****i /*** *ω2 = temp2* ***/*** *ω2*

*= 1921.1 / 0.6667 = 2881.5059*

*σw2 = ω1 \* σ12 + ω2 \* σ22*

*= 0.3333 \* 0.2223 + 0.6667 \* 2881.5059*

*= 1921.2*

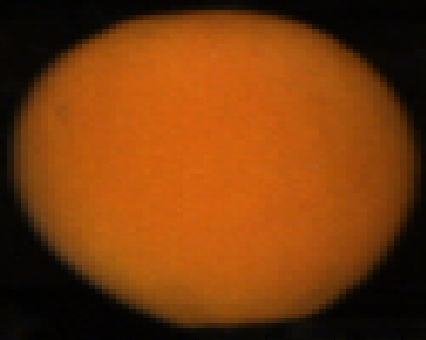
* **.................................................,**
* **.................................................**

10. Desired threshold corresponds to the maximum *Maxσb2* or minimum *Minσw2* which is at grey level 60.

**3.1.2 Experiment and Analysis**

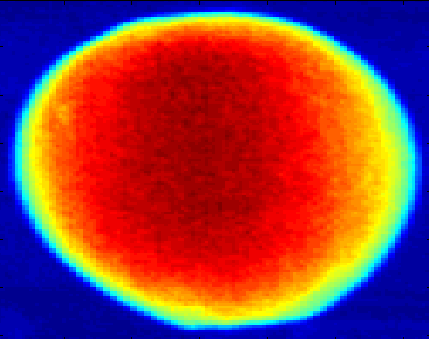
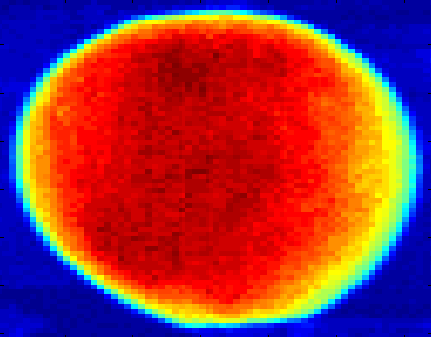
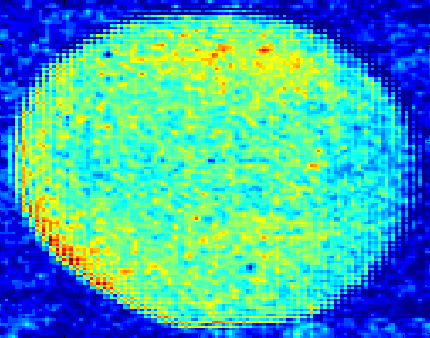
In machine vision and image processing, Otsu’s thresholding method is used to automatically perform histogram shape-based image thresholding. The histogram method assumes that there is some average value for the foreground and background pixels. In this case, the foreground is the orange itself, and the background is the conveyor belt. The following examples demonstrate Otsu’s method in the real application.

**1. Experiment One:**

****

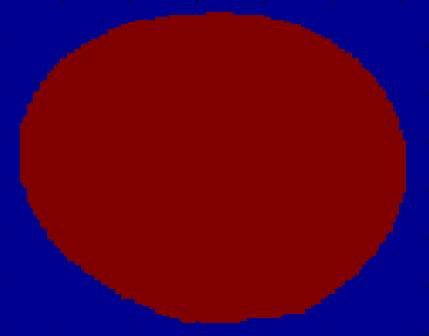
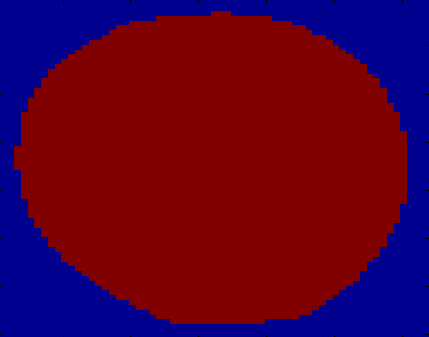
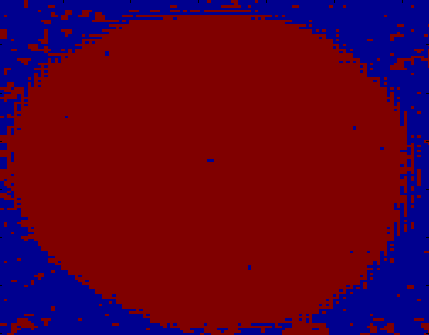
*Fig.10. Orange Sample One*

Orange sample one is selected from a database which contains all the good oranges. Extract red, green, and blue channels respectively from orange sample one. The following images show isolated color channels in natural colors.

*Fig.10. Isolated Color Channels*

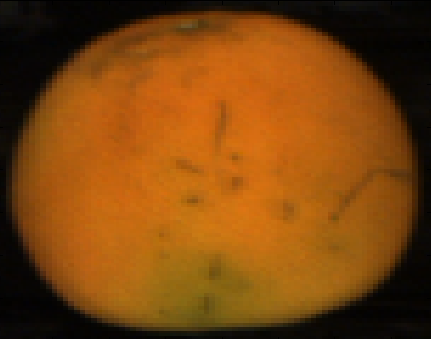
The first one is isolated red channel in natural colors, the second one is isolated green channel in natural colors, and the third one is isolated blue channel in natural colors. Compute the best threshold for three channels respectively using Otsu’s method. The best threshold for red channel is 96, the best threshold for green channel is 46 and the best threshold for blue channel is 9. The following images show after applying the best thresholds on three channels respectively.

*Fig.10. Orange Segmentation*

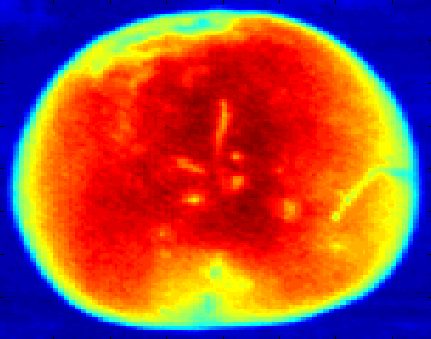
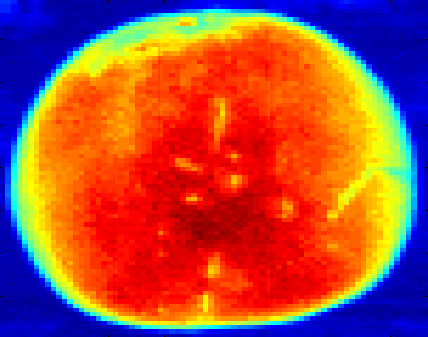
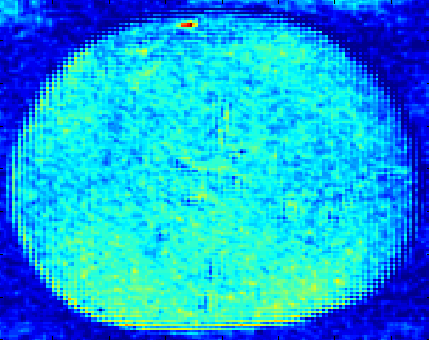
According to the above images, it is clearly to see that Otsu’s method works perfectly on the isolated red and green channels. The reason why background pixels appear on the isolated blue channel is because the pixel intensity values on the blue channel are very low. It is mixed up a litter bit with the pixels on the background. The pixel intensity value on the blue channel is not important for the whole process due to the nature of the orange. So the conclusion is Otsu’s method works well on good orange foreground segmentation.

**2. Experiment Two:**

****

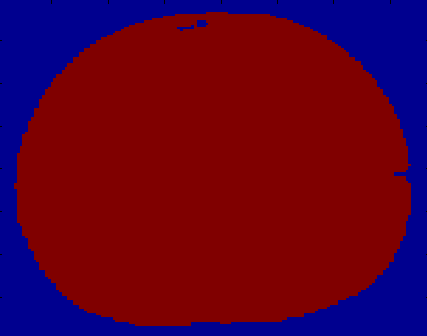
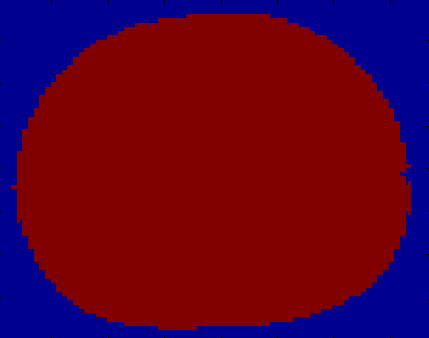
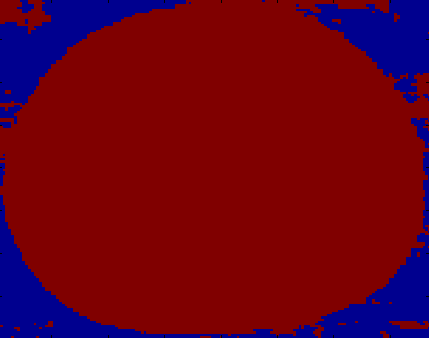
*Fig.10. Orange Sample Two*

Orange sample two is selected from a database which contains all the blemished oranges. Extract red, green, and blue channels respectively from orange sample two. The following images show isolated color channels in natural colors.

*Fig.10. Isolated Color Channels*

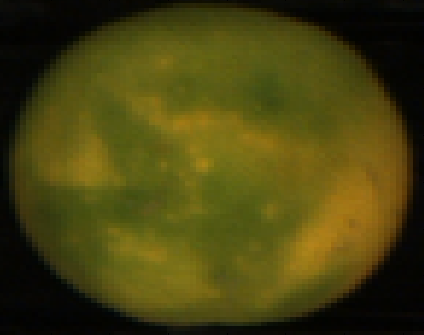
Compute the best threshold for three channels respectively using Otsu’s method. The best threshold for red channel is 94, the best threshold for green channel is 53 and the best threshold for blue channel is 11. The following images show after applying the best thresholds on three channels respectively.

*Fig.10. Orange Segmentation*

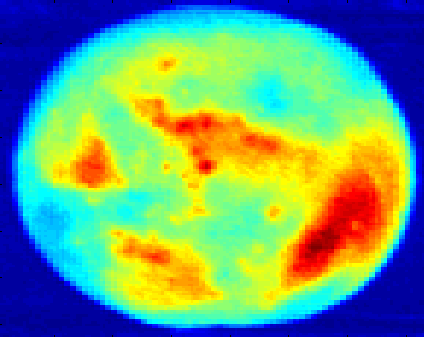
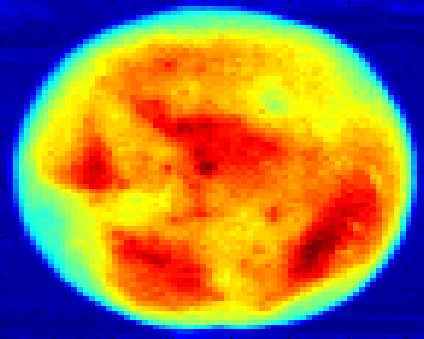
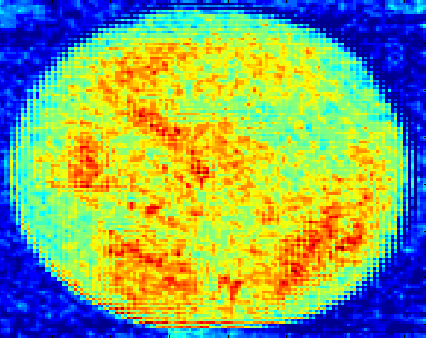
According to the above images, it is clearly to see that Otsu’s method works fine on the isolated red and green channels. There are two very small holes on the first image. They are caused by low intensity parts, such as blemish or stem. It wound not affect the overall performance. In conclusion Otsu’s method works well on blemished orange foreground segmentation.

**3. Experiment Three:**

****

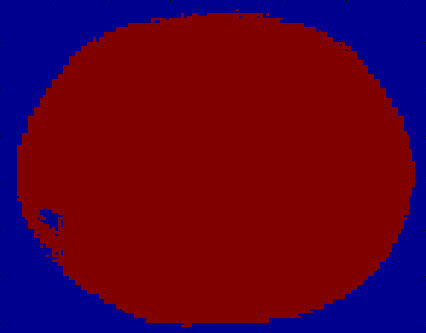
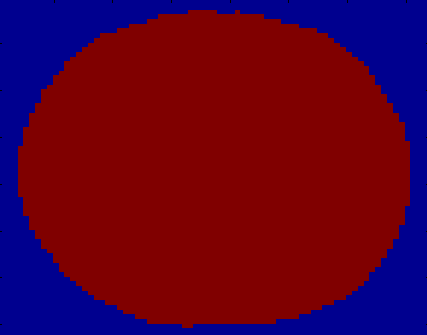
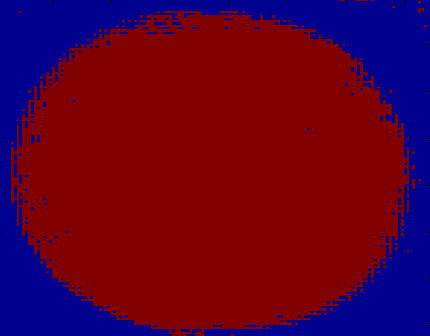
*Fig.10. Orange Sample Three*

Orange sample three is selected from a database which contains all the green oranges. Extract red, green, and blue channels respectively from orange sample three. The following images show isolated color channels in natural colors.

*Fig.10. Isolated Color Channels*

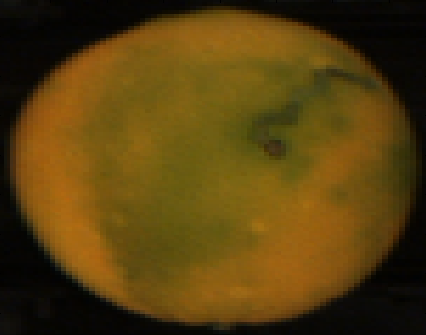
Compute the best threshold for three channels respectively using Otsu’s method. The best threshold for red channel is 55, the best threshold for green channel is 50 and the best threshold for blue channel is 11. The threshold for the red channel is similar to the threshold for the green channel. That means the pixel intensity values on the red channel is more close to the pixel intensity values on the green channel compared with Experiment One and Two. The following images show after applying the best thresholds on three channels respectively.

*Fig.10. Orange Segmentation*

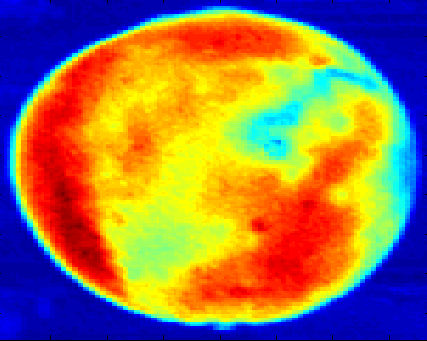
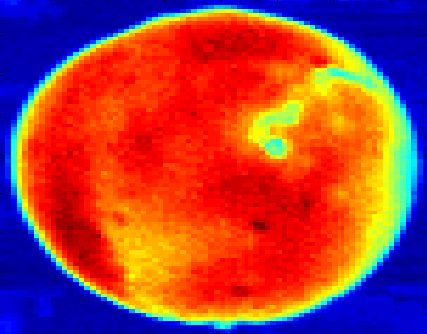
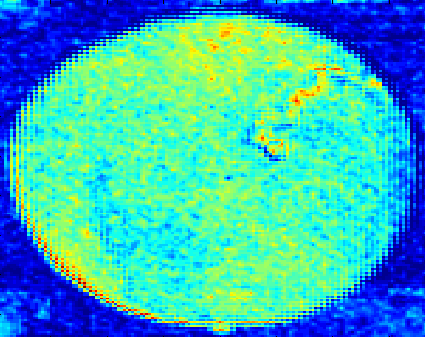
According to the above images, it is clearly to see that Otsu’s method works fine on the isolated red, green and blue channels. There is a hole on the first image. It is caused by a low intensity portion compared with its neighbors. From Experiment Two and Three, it is clearly to see that the pixel intensity value on the red channel is very sensitive to stems and blemishes. In another word, the intensity value for both stems and blemishes on the red channel dropped a lot compared to normal skin. In conclusion Otsu’s method works well on green orange foreground segmentation.

**4. Experiment Four:**

****

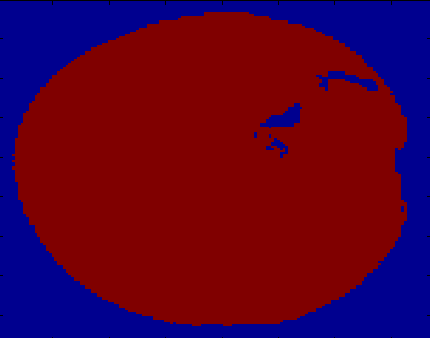
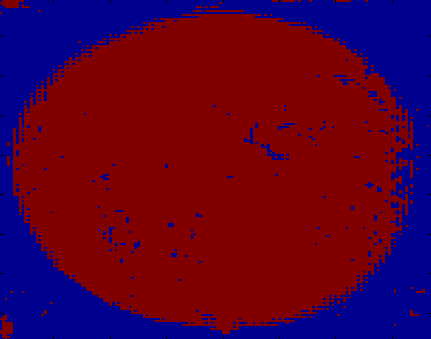
*Fig.10. Orange Sample Four*

Orange sample four is selected from a database which contains all the blemished green oranges. Extract red, green, and blue channels respectively from orange sample three. The following images show isolated color channels in natural colors.

*Fig.10. Isolated Color Channels*

Compute the best threshold for three channels respectively using Otsu’s method. The best threshold for red channel is 67, the best threshold for green channel is 48 and the best threshold for blue channel is 11. The following images show after applying the best thresholds on three channels respectively.

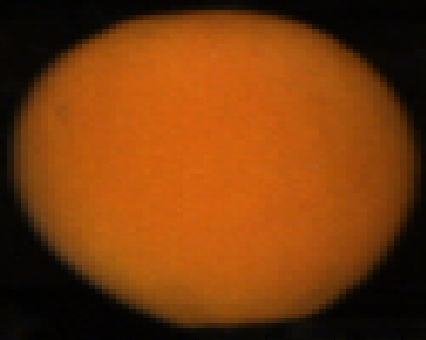
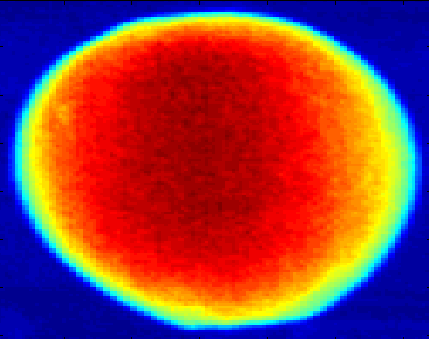
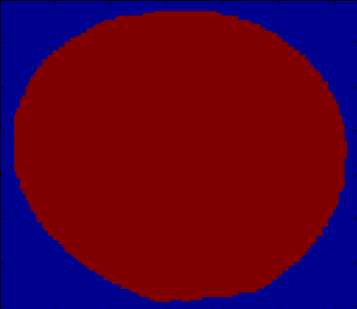
  

*Fig.10. Orange Segmentation*

According to the above images, it is clearly to see that Otsu’s method works fine on the isolated red, green and blue channels. There are a few holes on the first and second images. The holes on the first image are more related to the color changes, such as the color suddenly changed from red to green. The holes in the second image are more related to the blemishes, such as dark spots appeared on the green skin. In another word, for a green orange, the pixel intensity on the green channel should be focused on. In conclusion Otsu’s method works well on blemished orange foreground segmentation.

**3.1.3 Algorithm Modifications and Improvements**

Otsu’s method works well for the segmentation of orange and conveyor, however, the cost of Otsu’s method is computationally expensive. Due to the nature of the orange and conveyor, a constant threshold can be set for all the images for segmentation purposes. The pixel intensity value on the conveyor is very low and the pixel intensity value on the orange is high. The best contrast appears on the red channel. Setting a constant threshold on the red channel can achieve the same result as Otsu’s method. The following example shows the result of orange segmentation on the red channel using constant threshold 50.

 **** 

*Fig.10. Constant Threshold*

One hundred sample images are selected from different orange databases. The comparison is made between Otsu’s method and a constant threshold. Otsu’s method is computational expensive, but more accurate on image segmentation. Using a constant threshold is computational cheap, but less accurate than Otsu’s method. The assumption is that Otsu’s method is one hundred percent accurate on foreground segmentation for selected data samples. Foreground segmented using a constant threshold is compared with foreground segmented using the Otsu’s method. False positive rate and true positive rate are selected as the statistical judgment.

1. False Positive Rate

False positive rate is the proportion of negative instances that were erroneously reported as being positive. If a pixel belongs to the background, but erroneously classified as a foreground pixel, then this pixel is counted as a false positive pixel. All the pixels should belong to the background are classified as negative instances.

*fpr = number of false positive pixels / total number of negative instances*

2. True Positive Rate

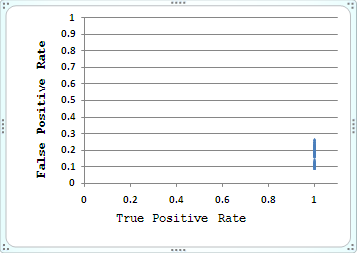
True positive rate is the proportion of positive instances that reported as being positive. If a pixel should belong

to the foreground pixel and it is reported as a foreground pixel, then this pixel will be counted as a true positive

pixel. All the pixels should belong to the foreground pixel are classified as positive instances.

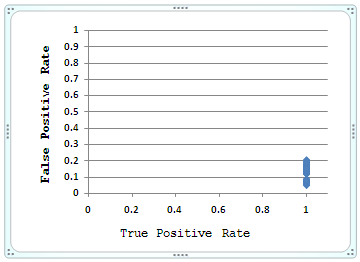
*tpr = number of true positive pixels / total number of positive instances*

Low false positive rate and high true positive rate means that the testing result is close to the ideal assumption. The following chart shows the result of using a constant threshold of 50.



*Fig.10. Constant Threshold Fifty*

All the pixels should be classified as foreground pixels are reported, but many pixels on the background are classified as foreground pixels as well. This means the threshold value 50 is a bit low. The following chart shows the result of using a constant threshold of 70.



*Fig.10. Constant Threshold Seventy*

It is clearly to see that the false positive rate decreased a lot after increasing the constant threshold to 70. The true positive rate states unchanged. That means many pixels should belong to the background are actually reclassified correctly without affecting the foreground pixels. So a constant threshold may be selected as the segmentation point between orange and conveyor according to the data source.

**3.1.4 Summary**

Otsu’s method works well on the object segmentation between orange and conveyor, however, it is computational expensive. Using a constant threshold for object segmentation is computational cheap. The statistical analysis on true and false positive rates shows that all the pixels belong to the orange are classified correctly, and a few pixels belong to the conveyor are misclassified. In conclusion, Otsu’s method can be replaced by using a constant threshold value on the red channel of a RGB image. The value of this threshold has to be changed accordingly when the lighting condition is changed.

1. Adjust the camera and light source.

2. Select one hundred orange samples randomly.

3. Select a constant threshold value, and apply to the red channel of the orange samples.

4. Compare the result with Otsu’s method using statistical

measurement of true and false positive rates.

5. The best threshold appears when the true positive rate

is high and the false positive rate is low.

Once the orange is extracted from the conveyor on the red channel, then a mask can be generated and apply to the green and blue channels separately for orange segmentation. It is more accurate than applying Otsu’s method on the blue channel. Because all the pixel intensity values on the blue channel are very low, the foreground and background pixels are sometimes mixed up together. A constant threshold can also be used for stem segmentation as well. This algorithm will be talked about in a later chapter.

**Chapter 4**

**Novel Algorithms on Orange Grading System**

* 1. **Orange Grading System Architecture**
  2. **RGB Color Class**

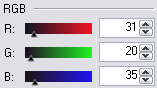
**4.2.1 Experiment and Analysis**

All possible colors can be made from three primary colors red, green, and blue. The following experiment demonstrates that a broad array of colors can be displayed by using an appropriate combination of red, green, and blue intensities. There are three colored light beams with dimmer switches, one red light, one green light, and one blue light. Three colored light beams are used to shine three primary colors onto a black wall and dimmer switches are used to adjust the intensity of each primary color.



*Fig.10. Color Combination*

A similar experiment could be simulated by using Paint.net which is a famous tool for image processing. Three colored light beams are simulated by using red, green, and blue colors separately. The dimmer switch on each light beam is simulated by adjusting the intensity value of each color.



*Fig.10. Color and Intensity*

1. If only the red light is on, the wall looks red.



Dim the red light some, the wall still looks red, but darker.

|  |  |  |  |
| --- | --- | --- | --- |
| **Red Light** | | | |
|  |  |  |  |
| **High Intensity Low Intensity** | | | |

Bring up the red light a bit, the wall still looks red, but brighter.

|  |  |  |  |
| --- | --- | --- | --- |
| **Red Light** | | | |
|  |  |  |  |
| **High Intensity Low Intensity** | | | |

1. If only the green light is on, the wall looks green.



Dim the green light some, the wall still looks green, but darker.

|  |  |  |  |
| --- | --- | --- | --- |
| **Green Light** | | | |
|  |  |  |  |
| **High Intensity Low Intensity** | | | |

Bring up the green light a bit, the wall still looks green, but brighter.

|  |  |  |  |
| --- | --- | --- | --- |
| **Green Light** | | | |
|  |  |  |  |
| **High Intensity Low Intensity** | | | |

1. If only the blue light is on, the wall looks blue.



Dim the blue light some, the wall still looks blue, but darker.

|  |  |  |  |
| --- | --- | --- | --- |
| **Blue Light** | | | |
|  |  |  |  |
| **High Intensity Low Intensity** | | | |

Bring up the blue light a bit, the wall still looks blue, but brighter.

|  |  |  |  |
| --- | --- | --- | --- |
| **Blue Light** | | | |
|  |  |  |  |
| **High Intensity Low Intensity** | | | |

Each of these primary colors produced different results in color and brightness. RGB Color Class is derived from the features of the composition of light which are detectable by the human eye. In human visual system, each of these primary colors with arbitrary intensity can be categorized by the spectral sensitivities of the light receptors in the human eye. For this reason, each of these primary colors is classified as a distinct RGB Color Class. For instance, red color is classified as a RGB Red Color Class, green color is classified as a RGB Green Color Class, and blue color is classified as a RGB Blue Color Class. Each of these color classes may contain many intensity levels from weakest to strongest. The average or mean intensity level for that class is a very useful feature in statistics.

1. If red and green lights are on together with equal intensity, the wall looks yellow.



Dim the red light some, and the wall becomes more chartreuse.

|  |  |  |  |
| --- | --- | --- | --- |
| **Red and Green Lights** | | | |
|  |  |  |  |
| **High Intensity Red Light Low Intensity** | | | |

Dim the green light a bit, the wall becomes more orange.

|  |  |  |  |
| --- | --- | --- | --- |
| **Red and Green Light** | | | |
|  |  |  |  |
| **High Intensity Green Light Low Intensity** | | | |

1. If red and blue lights are on together with equal intensity, the wall looks magenta.



Dim the red light some, and the wall becomes more violet.

|  |  |  |  |
| --- | --- | --- | --- |
| **Red and Blue Lights** | | | |
|  |  |  |  |
| **High Intensity Red Light Low Intensity** | | | |

Dim the blue light a bit, the wall becomes more rose.

|  |  |  |  |
| --- | --- | --- | --- |
| **Red and Blue Light** | | | |
|  |  |  |  |
| **High Intensity Blue Light Low Intensity** | | | |

1. If green and blue lights are on together with equal intensity, the wall looks cyan.



Dim the green light some, and the wall becomes more azure.

|  |  |  |  |
| --- | --- | --- | --- |
| **Green and Blue Lights** | | | |
|  |  |  |  |
| **High Intensity Green Light Low Intensity** | | | |

Dim the blue light a bit, the wall becomes more aquamarine.

|  |  |  |  |
| --- | --- | --- | --- |
| **Green and Blue Light** | | | |
|  |  |  |  |
| **High Intensity Blue Light Low Intensity** | | | |

Yellow is formed by the sum of red and green colors of equal intensity, magenta is formed by the sum of red and blue colors of equal intensity, and cyan is formed by the sum of green and blue colors of equal intensity. Yellow, magenta, and cyan are called secondary colors which are formed by the sum of two primary colors of equal intensity. Each of these secondary colors is classified as a distinct RGB Color Class, e.g. yellow color is classified as a RGB Yellow Class. If a pixel in a colored image is classified as a member of the RGB Yellow Class, then this pixel must have only two channels, red channel and green channel. However, the intensity levels are not necessarily to be the same for both channels. In another word, the different combination of colors is the only necessary property to distinguish among RGB Color Classes. The following example shows three member pixels in RGB Yellow Class.

|  |  |  |  |
| --- | --- | --- | --- |
| RGB Yellow Class | | | |
| Members | **Red Channel** | **Green Channel** | **Blue Channel** |
| Pixel A | 255 | 255 | None |
| Pixel B | 150 | 80 | None |
| Pixel C | 60 | 200 | None |

*Fig.10. RGB Yellow Class*

Pixel A, Pixel B, and Pixel C are members of RGB Yellow Class. They all have only two channels, red channel and green channel, which is the feature of the composition of light. The following example shows three member pixels in RGB Magenta Class.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RGB Magenta Class | | | | |
| Members | **Red Channel** | **Green Channel** | **Blue Channel** | **Color** |
| Pixel D | 255 | None | 255 |  |
| Pixel E | 150 | None | 80 |  |
| Pixel F | 60 | None | 200 |  |

*Fig.10. RGB Magenta Class*

Pixel D, Pixel E, and Pixel F are members of RGB Magenta Class. They all have only two channels, red channel and blue channel. The colors of these three pixels look different, but they all have two channels in common. That is the reason why they are categorized into the same class. The following example shows three member pixels in RGB Cyan Class. Pixel G, Pixel H, and Pixel I are members of RGB Cyan Class. They all have only two channels, green channel and blue channel.

|  |  |  |  |
| --- | --- | --- | --- |
| RGB Cyan Class | | | |
| Members | **Red Channel** | **Green Channel** | **Blue Channel** |
| Pixel G | None | 255 | 255 |
| Pixel H | None | 80 | 150 |
| Pixel I | None | 200 | 60 |

*Fig.10. RGB Cyan Class*

1. If red, green, and blue lights are on together with full intensities, then the wall looks white.

Dim the red light some, and the wall becomes a bit blue.

|  |  |  |  |
| --- | --- | --- | --- |
| **Red, Green and Blue Lights** | | | |
|  |  |  |  |
| **High Intensity Red Light Low Intensity** | | | |

Dim the green light a bit, the wall becomes a bit pink.

|  |  |  |  |
| --- | --- | --- | --- |
| **Red, Green and Blue Light** | | | |
|  |  |  |  |
| **High Intensity Green Light Low Intensity** | | | |

Dim the blue light a bit, the wall becomes a bit yellow.

|  |  |  |  |
| --- | --- | --- | --- |
| **Red, Green and Blue Light** | | | |
|  |  |  |  |
| **High Intensity Blue Light Low Intensity** | | | |

Yellow is the complement of blue, magenta is the complement of green, and cyan is the complement of red. Yellow, magenta, and cyan are secondary colors, and red, green and blue are primary colors. So every secondary color is the complement of one primary color. When two colors, one primary and its complementary secondary color, are added together, the wall looks white. For this reason, the combination of one primary color and its complementary secondary color is classified as a RGB White Class. In another word, whenever a pixel has three channels, red, green, and blue channel, then this pixel is categorized as a member of RGB White Class. The following example shows three pixels in RGB White Class.

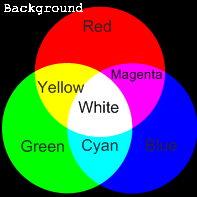
|  |  |  |  |
| --- | --- | --- | --- |
| RGB White Class | | | |
| Members | **Red Channel** | **Green Channel** | **Blue Channel** |
| Pixel J | 255 | 255 | 255 |
| Pixel K | 111 | 80 | 150 |
| Pixel L | 40 | 200 | 60 |

*Fig.10. RGB White Class*

Pixel J, Pixel K, and Pixel L are members of RGB white class. They all have three channels, red, green, and blue channel, although the intensity levels on each channel are different.

1. If all red, green, and blue lights are off, the wall looks black. This is a special case, no light. For all the pixels with zero intensity levels for all three color channels are classified as a RGB Background Class. All the pixels in this class are treated as background pixels. In another word, all the pixels which do not belong to RGB Background Class are treated as object pixels. The background is related to the conveyor, and the object is related to the orange in this case.

There are eight RGB Color Classes in total, e.g. RGB Red Class, RGB Green Class, RGB Blue Class, RGB Yellow Class, RGB Magenta Class, RGB Cyan Class, RGB White Class, and RGB Background Class. The following chart shows the distribution of each color class.

*Fig.10. Distribution of RGB Color Classes*

* + 1. **Derived Formula**

A RGB image has three channels, red, green, and blue channel. Each of these three channels can be manipulated separately from the others. A channel is the grayscale image of the same size as the color image. Otsu’s thresholding method is used to convert a grayscale image to a binary/logical image which contains only “0” and “1”. The following formula shows the method to classify each individual pixel into an appropriate RGB Color Class using binary image.

***ClassNoij = 4 \* Redij + 2 \* Greenij + Blueij + 1***

*Where 1≤ i ≤ Rows, 1≤ j ≤ Columns*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RGB Color Class | | | | |
| Class No. | **Class Name** | **Red** | **Green** | **Blue** |
| 1 | Background Class | **0** | **0** | **0** |
| 2 | Blue Class | **0** | **0** | 1 |
| 3 | Green Class | 0 | 1 | 0 |
| 4 | Cyan Class | 0 | 1 | 1 |
| 5 | Red Class | 1 | 0 | 0 |
| 6 | Magenta Class | 1 | 0 | 1 |
| 7 | Yellow Class | 1 | 1 | 0 |
| 8 | White Class | 1 | 1 | 1 |

*Fig.10. RGB Color Class*

There are total eight RGB Color Classes indexed from one to eight. In the above diagram, “0” means attendance, and “1” means absence. For instance, if a pixel is a member of RGB Red Class, then its binary value on the red channel will be “1”, and its binary values on both green and blue channels will be “0”.

* + 1. **Algorithm**

1. For a given RGB image, extract red, green, and blue channel separately.
2. Using Otsu’s method to find the optimum threshold value for three channels separately.
3. Convert three channels to three binary images separately according to their corresponding optimum threshold values.

***0, if Pij ≤ threshold***

***Bij =***

***1, if Pij > threshold***

*Where 1≤ i ≤ Rows, 1≤ j ≤ Columns*

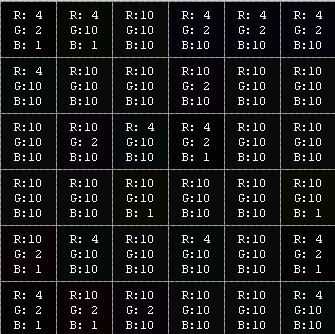
* 1. If the intensity of a pixel is less than or equal to the threshold value, then mark this pixel as a background pixel. A background pixel is given a value of “0”.
  2. If the intensity of a pixel is greater than the threshold value, then mark this pixel as foreground pixel. A foreground pixel is given a value of “1”.

1. Create a matrix of the same size as the RGB image. This matrix is defined as a RGB Class Distribution Matrix. RGB Class Distribution Matrix is a two dimensional array.
2. Loop through all the pixels in the RGB image.
   1. Apply the derived formula, and calculate the class number for each given pixel.
   2. Store the class number of each given pixel into the RGB Class Distribution Matrix.

RGB Class Distribution Matrix holds the class information for each individual pixel.

* + 1. **Implementation**

All the pixels in a RGB image should be categorized into eight one of eight RGB Color Classes and all the class information is stored in the RGB Class Distribution Matrix for further analysis. Subset of a RGB image is selected as the sample data for demonstration purpose. The sample data selected is a three dimensional array of picture elements. A three dimensional array is basically multi two dimensional arrays layered on top of each other. A two dimensional array, which is also referred to as a table, consists of both rows and columns of elements.



*Fig.10. Sample Data*

This sample data consists of three two dimensional arrays layered on top of each other. Each two dimensional array consists of six rows and six columns, thus is called a 6-by-6 array. The first dimension is referred to red channel, the second dimension is referred to green channel, and the third dimension is referred to blue channel.

1. Extract red, green, and blue channel separately from the sample data set. Each channel is a two dimensional array of picture elements with the same size as the original image.

|  |
| --- |
| Red Channel |
|  |

*Fig.10. Red Channel*

|  |
| --- |
| Green Channel |
|  |

*Fig.10. Green Channel*

|  |
| --- |
| Blue Channel |
|  |

*Fig.10. Blue Channel*

1. Compute the optimum threshold for each isolated color channel separately using Otsu’s method.

|  |  |
| --- | --- |
| Channel | Optimum Threshold Value |
| Red | 4 |
| Green | 2 |
| Blue | 1 |

*Fig.10. Optimum Threshold*

1. Convert grey scale image to binary image. A binary image is a digital image that has only two possible values for each pixel, such as zero and one.

*For i = 1:rows*

*For j = 1:columns*

*If Pij > threshold*

*Bij = 1*

*Else*

*Bij = 0*

*End*

*End*

*End*

**1. Case One**

The intensity value of *P(1,1)* in the red channel is four, and the optimum threshold value for the red channel is four. The intensity value of *P(1,1)* equals to the optimum threshold, so the *B(1,1)* in the binary image is given to a value “0”.

*P(1,1) = 4, threshold = 4,*

*P(1,1) = threshold,*

*B(1,1) = 0*

**2. Case Two**

The intensity value of *P(2,1)* in the blue channel is ten, and the optimum threshold value for the blue channel is one. The intensity value of *P(2,1)* isgreater than the optimum threshold, so the *B(2,1)* in the binary image is given to a value “1”.

*P(2,1) = 10, threshold = 1,*

*P(2,1) > threshold,*

*B(2,1) = 1*

The converted binary images for red, green, and blue channel separately are shown below.

|  |
| --- |
| Red Channel – Binary Image |
|  |

*Fig.10. Binary Image of Red Channel*

|  |
| --- |
| Green Channel – Binary Image |
|  |

*Fig.10. Binary Image of Green Channel*

|  |
| --- |
| Blue Channel – Binary Image |
|  |

*Fig.10. Binary Image of Blue Channel*

1. Initialize the RGB Class Distribution Matrix which is the same size as the sample data.
2. Loop through all the pixels in the sample data set. Compute the class number for each individual pixel.

*For i = 1:rows*

*For j = 1:columns*

*ClassNoij = 4\*Rij + 2\*Gij + Bij + 1*

*End*

*End*

1. **Case One**

Compute the class number for *P(1,2).* The value of *R(1,2)* in the binary image of red channel is zero. The value of *G(1,2)* in the binary image of green channel is one. The value of *B(1,2)* in the binary image of blue channel is zero.

*R(1,2) = 0, G(1,2) = 1, B(1,2) = 0,*

*ClassNo(1,2) = 4 \* R(1,2) + 2 \* G(1,2) + B(1,2) + 1*

*= 4 \* 0 + 2 \* 1 + 0 + 1 = 3*

The class number of *P(1,2)* is three. *P(1,2)* is classified as a member of RGB Green Class.

1. **Case Two**

Compute the class number for *P(1,6).* The value of *R(1,6)* in the binary image of red channel is zero. The value of *G(1,6)* in the binary image of green channel is one. The value of *B(1,6)* in the binary image of blue channel is one.

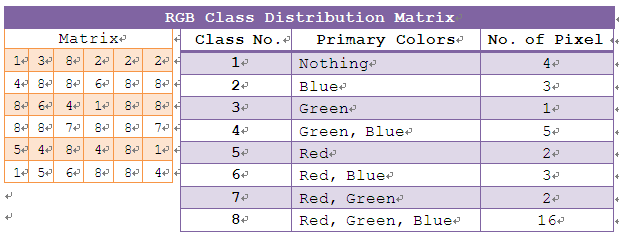
*R(1,6) = 0, G(1,6) = 0, B(1,6) = 1,*

*ClassNo(1,6) = 4 \* R(1,6) + 2 \* G(1,6) + B(1,6) + 1*

*= 4 \* 0 + 2 \* 0 + 1 + 1 = 2*

The class number of *P(1,6)* is two. *P(1,6)* is classified as a member of RGB Blue Class.

1. Store the class number of each pixel into RGB Class Distribution Matrix for further analysis.



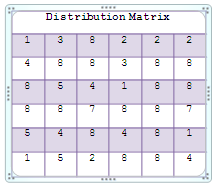
*Fig.10. RGB Class Distribution Matrix*

From the above diagram, it is clearly to see that *P(1,1)* is a member of RGB Background Class, *P(1,2)* is a member of RGB Green Class, and so on.

* + 1. **Advanced Data Analysis**

**4.2.5.1 Missing Class**

There are total eight RGB Color Classes. Some of these classes might be missing or not available. It is important to analyze which RGB Color Class is actually missing. The following diagram is an instance of RGB Class Distribution Matrix. RGB Class Distribution Matrix shows the distribution and availability of RGB Color Class in a given RGB image.

**

*Fig.10. RGB Distribution Matrix*

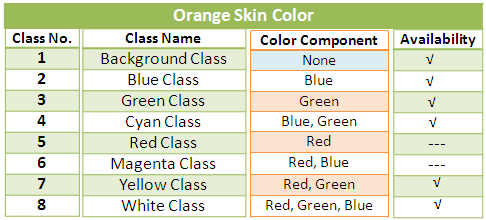
From the above diagram, it is clearly to see that RGB Magenta Class is missing. That means the combination of red and blue color is not available.

**1. Case One**

One hundred oranges are randomly selected from the database. Fifty of them are blemished oranges and fifty of them are good oranges. The orange skin color is orange. After a class availability test, all the selected oranges are missing RGB Red Class and RGB Magenta Class. That means the pure red color and the combination of red and blue colors are not available. All the missing classes will not be analyzed in the further.



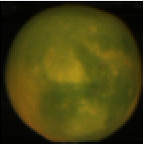
*Fig.10. Sample Orange*



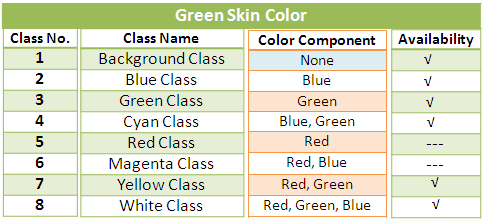
*Fig.10. Orange Skin Color*

**2. Case Two**

Fifty oranges are randomly selected from the database. Twenty of them are blemished oranges and twenty of them are good oranges. The orange skin color this time is green. After a class availability test, all the selected oranges are missing RGB Red Class and RGB Magenta Class. That means the pure red color and the combination of red and blue colors are not available.



*Fig.10. Sample Orange*



*Fig.10. Green Skin Color*

In Conclusion, all selected sample oranges are missing RGB Red Class and RGB Magenta Class. This may refer to the light source and the nature of the orange.

**4.2.5.2 Color Contrast**

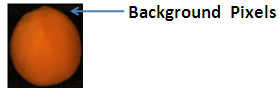
Pixels are classified into eight RGB Color Classes in a given RGB image. Each of these RGB Color Classes produces different results in color and brightness. Due to the nature of the orange texture, the brightness of RGB Color Classes increases from class one to class eight.



*Fig.10. Brightness*

1. **RGB Background Class**

All the pixels in this class belong to the background, such as conveyor. Pixels in this class have the lowest intensity compared with pixels in other classes.



*Fig.10. RGB Background Class*

1. **RGB Blue Class**

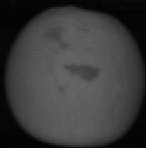
Pixels in this class have the pure blue color with very low intensity. This is the nature of the orange texture. The color of the orange skin can be orange or green, but not blue. The following image shows the isolated blue channel from a RGB image. It is hardly to see anything in this gray scale image. Increase



*Fig.10. RGB Blue Class*

1. **RGB Green Class**

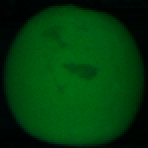
Pixels in this class have the pure green color with high intensity. This is true for green oranges especially. The following image shows the isolated green channel from a RGB image. From this grey scale image, it is reasonable easy to identify the texture feature on the orange skin.



*Fig.10. RGB Green Class*

1. **RGB Cyan Class**

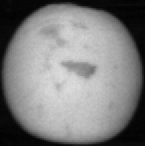
Cyan is called a secondary color which is formed by the sum of green and blue color. All the pixels in this class have only two channels, green channel and blue channel. Pixels in this class are a litter bit brighter than pixels in the RGB Green Class. The blue color is added on top of the green color.



*Fig.10. RGB Cyan Class*

1. **RGB Red Class**

Pixels in this class have the pure red color with very high intensity. This is true for green oranges as well. The following image shows the isolated red channel from a RGB image. From this grey scale image, it is very easy to identify the texture feature on the orange skin.



*Fig.10. RGB Red Class*

1. **RGB Magenta Class**

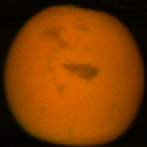
Magentais called a secondary color which is formed by the sum of red and blue color. All the pixels in this class have only two channels, red channel and blue channel. Pixels in this class are a litter bit brighter than pixels in the RGB Red Class. The blue color is added on top of the red color.



*Fig.10. RGB Magenta Class*

1. **RGB Yellow Class**

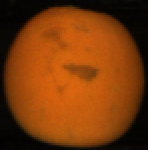
Yellowis called a secondary color which is formed by the sum of red and green color. All the pixels in this class have only two channels, red channel and green channel. Pixels in this class have very high intensity.



*Fig.10. RGB Yellow Class*

1. **RGB White Class**

White is formed by the combination of one primary color and its complementary secondary color. All the pixels in this class have three channels, red, green, and blue channel. Pixels in this class have the highest intensity. RGB White Class and RGB Yellow Class are very similar.



*Fig.10. RGB White Class*

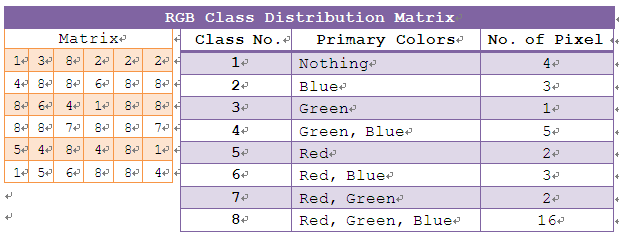
* + 1. **Summary**

All possible colors can be made from three primary colors red, green, and blue. All the pixels in a RGB image are classified into eight RGB Color Classes. The derived formula shows the method to classify each individual pixel into an appropriate RGB Color Class using binary image.

***ClassNoij = 4 \* Redij + 2 \* Greenij + Blueij + 1***

*Where 1≤ i ≤ Rows, 1≤ j ≤ Columns*

The class number for each pixel is stored in the RGB Class Distribution Matrix for further analysis.



*Fig.10. RGB Class Distribution Matrix*

Some of the RGB Color Classes might be not available, such as RGB Red Class and RGB Magenta Class. This is caused by the light source and the nature of the orange skin. The brightness of the RGB Color Classes increases from class one to eight. This order is specially designed for orange only. For other fruits, such as lemon, the formula used to compute the class number has to be changed accordingly.

* 1. **Class Mean**

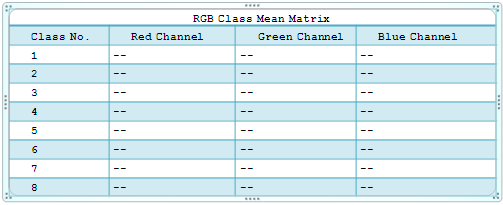
**4.3.1 Overview**

Mean has two related meanings in statistics, such as arithmetic mean and population mean. Arithmetic mean is the one selected in this algorithm and often simply called the “mean”. It is sometimes stated that the “mean” means average. For a given data set, the average is the sum of the measurements divided by the number of measurements and to compute a number as being the average. Changing the order of the measurements does not affect the final result. The formula of mean is defined as follow:

***M = 1/n* ∑*ai = 1/n (a1 + a2 +.... + an),***

***where* *1 ≤ i ≤ n***

In this research, the class mean refers to a measure of the average intensity of the data set. It is a measurement of the central tendency for each RGB Color Class. A RGB image is a combination of three channels, red, green, and blue channel. So the class mean is computed for each RGB Color Class on three channels separately. Class mean is stored in the RGB Class Mean matrix which consists of eight rows and three columns, e.g.



*Fig.10. RGB Class Mean Matrix*

* + 1. **Algorithm**

1. Compute the number of pixels that belong to class one using the RGB Class Distribution Matrix.

1. Initialize ***n1*** to be 0.

2. If the class number is equal to *1* then count this number, otherwise, do not count.

***n1 + 1, if ClassNoij == 1***

***n1 =***

***n1, if ClassNoij != 1***

*Where 1≤ i ≤ Rows, 1≤ j ≤ Columns*

2. Compute the sum of pixel intensities of class one in the red channel.

1. Create a temporary two dimensional matrix of the same size as the red channel. This temporary matrix is used as a mask to extract all the pixel intensities out from the red channel.

2. If the class number is equal to *1* in the RGB Class Distribution Matrix then write one in the temporary matrix, otherwise, write zero.

***1, if ClassNoij == 1***

***tempij =***

***0, if ClassNoij != 1***

*Where 1≤ i ≤ Rows, 1≤ j ≤ Columns*

1. Multiply Temp matrix with RedChannel.

*temp1,1 temp1,2 .....*

Temp = *temp2,1 temp2,2 .....*

*...................*

and

*rc1,1 rc1,2 .....*

RedChannel = *rc2,1 rc2,2 .....*

*...............*

then

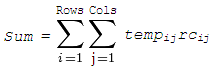
*temp1,1rc1,1  temp1,2rc1,2 ...*

*temp2,1rc2,1 temp2,2rc2,2 ...*

...........................

*Where 1≤ i ≤ Rows, 1≤ j ≤ Columns*

1. Compute the sum of the products.

**

*= temp1,1rc1,1 + temp1,2rc1,2 + ... + tempijrcij*

5. Compute the class mean of class one on red channel.

*ClassMean1 = Sum / n1*

6. Store *ClassMean1* in the RGB Class Mean Matrix.

7*.* Repeat step one to six, and compute class means for class two to eight on red channel.

8. Repeat step one to seven on green channel.

9. Repeat step one to seven on blue channel.

* + 1. **Implementation**

1. Compute the number of pixels for each class.

1. Initialize ***N*** to be 0.

*N* is an array that contains a group of 8 elements. The index of each element corresponds to the class number and the value of each element is the number of pixels in that class.

*N = {n1, n2, n3, n4, n5, n6, n7, n8}*

= *{0, 0, 0, 0, 0, 0, 0, 0}*

1. Step through RGB Class Distribution Matrix and count the number of pixels for each RGB Color Class.

*For i = 1 to rows*

*For j = 1 to columns*

*classNo = Pij*

*nclassNo = nclassNo + 1;*

*End*

*End*

The following diagram shows an instance of the RGB Class

Distribution Matrix. The number of pixels for each RGB Color Class is stored in the array *N.* It is clearly to see that the number of pixels for class one is four, the number of pixels for class two is three, and so on.

|  |
| --- |
| RGB Class Distribution Matrix |
|  |

*Fig.10. RGB Class Distribution Matrix*

|  |
| --- |
| N |
|  |

*Fig.10. Number of Pixels*

2. Compute the sum of pixel intensities of class one on the

red channel.

1. Create a temporary two dimensional matrix *Temp* which consists of six rows and six columns

2. Search for all the pixels in the RGB Class Distribution Matrix which belong to class one.

*For i = 1 to 6*

*For j = 1 to 6*

*If Pij*== 1

*tempij = 1*

*Else*

*tempij = 0*

*End*

*End*

*End*

This temporary matrix is used as a mask. All the pixels belong to class one are marked as “*1”*, otherwise, “*0”*.

|  |
| --- |
| Temp |
|  |

*Fig.10. Temporary Matrix*

1. Multiply Temp matrix with RedChannel. RedChannel is

a grey scale image which consists of six rows and six columns in this example.

|  |
| --- |
| Red Channel |
|  |

*Fig.10. Red Channel*

*1 0 .....*

Temp = *0 0.....*

*..........*

and

*4 4 .....*

RedChannel = *4 10.....*

*..........*

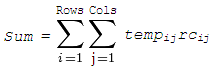
then

*4 0 .....*

TempRedChannel = *0 0.....*

..........

4. Compute the sum of the products.

**

*= temp1,1rc1,1 + temp1,2rc1,2 + ... + tempijrcij*

*= 4 + 0 + ... + 0 = 16*

3. Compute the class mean of class one on red channel.

*classMean1 = Sum / n1 = 16 / 4 = 4*

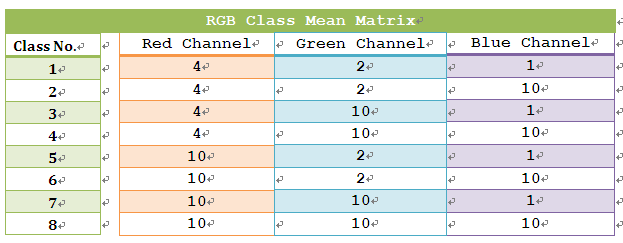
4. Store *classMean1* in the RGB Class Mean Matrix.

5*.* Repeat step two to four, and compute the class mean for class two to eight on red channel.

6. Repeat step two to five on green channel.

7. Repeat step two to five on blue channel.

After computation for each RGB Color Class, the RGB Class Mean Matrix looks like the following diagram.



*Fig.10. RGB Class Mean Matrix*

Each RGB Color Class has three class means, one is for the red channel, one is for the green channel, and one is for the blue channel. There are total twenty-four class means to be computed. So the RGB Class Mean Matrix is a two dimensional array with eight rows and three columns.

* + 1. **Algorithm Modifications and Improvements**

The mean selected in this algorithm is the arithmetic average of a set of values. The distribution of pixel intensities within each RGB Color Class is considered as a normal distribution. The normal distribution describes data that clusters around the mean. Sometimes a set of numbers might contain outliers. The outlier is the intensity of a pixel which is much lower or much higher than the others. The outliers are erroneous data caused by different reasons, such as leaves on the conveyor. The following example shows how the outliers affect the arithmetic mean.

**1. Case One**

The arithmetic mean of six values: *1, 2, 3, 4, 6, 20*

*(1 + 2 + 3 + 4 + 6 + 20) / 6 = 6*

The value *20* is an outlier which is much bigger than the

other values.

**2. Case Two**

The arithmetic mean of six values:

*1, 15, 17, 18, 19, 20*

*(1 + 15 + 17 + 18 + 19 + 20) / 6 = 15*

The value *1* is an outlier which is much smaller than the

other values.

**3. Case Three**

The arithmetic mean of six values:

*8, 30, 32, 34, 36, 100*

*(8 + 30 + 32 + 34 + 36 + 100) / 6 = 40*

The value *8* and *100* are outliers. The value *8* is much

smaller than the other values, and the value *100* is much bigger than the other values.

The outliers affect the accuracy of the arithmetic mean. If the outliers are removed from the data set, then the accuracy can be improved. There are three steps involved to recalculate the arithmetic mean.

1. Sort the selected data set.

2. Discard an equal amount of data at the high and the low end. For most statistical applications, five to twenty-five percent of the ends are discarded.

3. Compute the arithmetic mean of the remaining data.

The modified arithmetic mean is more accurate than the original arithmetic mean.

**4. Case Four**

The modified arithmetic mean of six values:

*30, 8, 32, 34, 100, 36*

1. Sort the selected data set.

*8, 30, 32, 34, 36, 100*

1. Discard one number at the high and the low end.

*30, 32, 34, 36*

3. Compute the arithmetic mean of the remaining data.

*(30 + 32 + 34 + 36) / 4 = 33*

The outliers *8* and *100* are discarded in this example. It is clearly to see that the modified arithmetic mean *33*

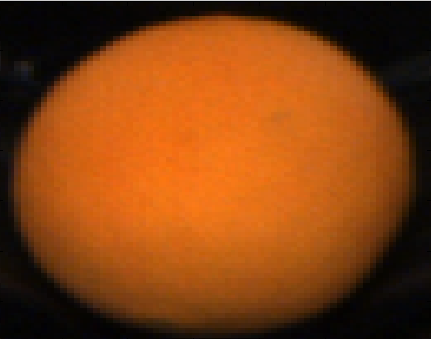
is more accurate than the original arithmetic mean *40* in

Case Three.

Ten percent of the ends are discarded in the real application. The modified arithmetic mean can improve the accuracy of this algorithm.

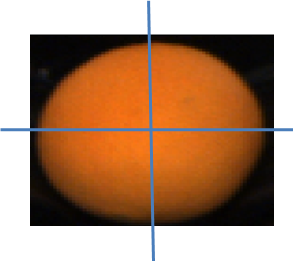
* + 1. **Advanced Data Analysis**
       1. **Effects of Illumination Intensity Variations on Ripe Orange Skin**

Most of oranges have orange skin color. The following is an example of orange skin.



*Fig.10. Orange Skin*

The brightness of an orange image from outside to inside is getting brighter and brighter. The size of selected sample image is 140 by 140. Two blue lines are drawing across the centre of this image.



*Fig.10. Intensity Change*

The following data shows the intensity changes along the blue lines.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Intensity Changes along the Horizontal Line | | | | | | | | | | | | | |
| cols | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 | 130 |
| R | 12 | 16 | 104 | 157 | 183 | 204 | 255 | 218 | 176 | 145 | 97 | 20 | 14 |
| G | 10 | 11 | 50 | 77 | 84 | 96 | 104 | 101 | 82 | 68 | 51 | 14 | 12 |
| B | 6 | 9 | 10 | 18 | 18 | 24 | 27 | 25 | 16 | 14 | 13 | 11 | 7 |

*Fig.10. Horizontal Intensity Change*

The pink line in the above diagram is the column selected along the horizontal blue line. The center of this horizontal blue line is row seventy and column seventy. All the other columns are selected using symmetry principle, such as column sixty and column eighty. It is clearly to see that the intensity increases from outside to inside for all three channels.

1. The intensity changes a lot on the red channel from outside to inside. The most significant intensity change happens on the red channel. The range is from *12* to *255*.

2. The intensity changes many on the green channel from outside to inside, but not like red channel. This is the second significant intensity change on the green channel. The range is from *10* to *104.*

3. The intensity changes a little bit on the blue channel. The intensity change on the blue channel is not important. The range is from *6* to *27*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Intensity Changes along the Vertical Line | | | | | | | | | | | | | |
| rows | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 | 130 |
| R | 11 | 19 | 109 | 177 | 199 | 227 | 255 | 222 | 182 | 157 | 85 | 28 | 17 |
| G | 8 | 10 | 56 | 81 | 97 | 102 | 104 | 99 | 92 | 77 | 43 | 18 | 10 |
| B | 4 | 8 | 10 | 11 | 21 | 23 | 27 | 25 | 25 | 21 | 11 | 10 | 2 |

*Fig.10. Vertical Intensity Change*

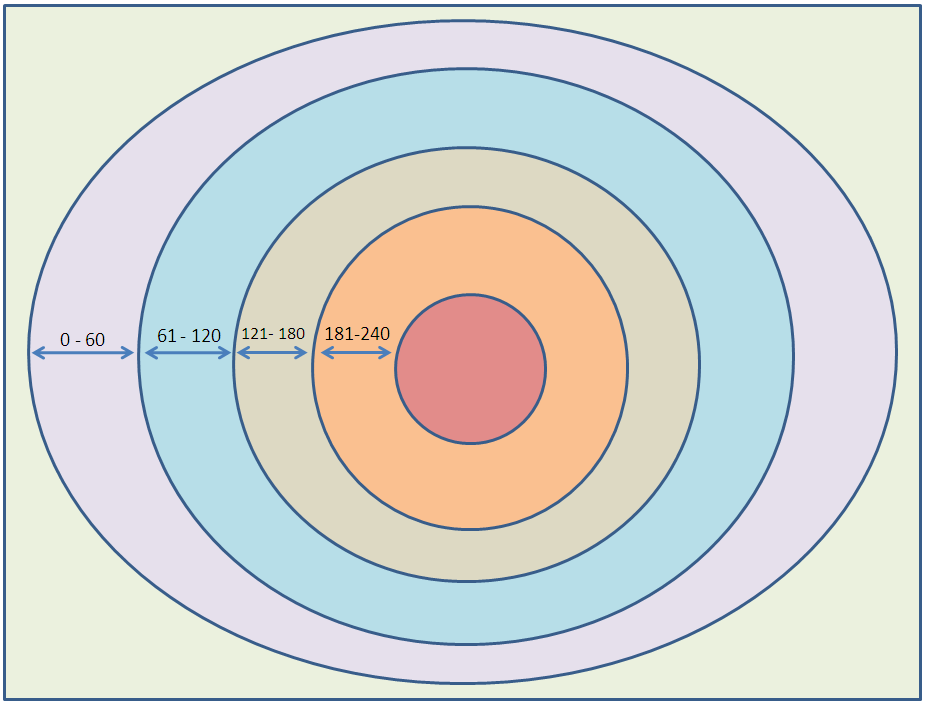
The pink line in the above diagram is the row selected along the vertical blue line. The center of this vertical blue line is row seventy and column seventy. All the other rows are selected using symmetry principle, such as row sixty and row eighty. It is clearly to see that the intensity increases from outside to inside for all three channels.

1. The intensity changes a lot on the red channel. The range is from *11* to *255*

2. The intensity changes many on the green channel. The range is from *8* to *104*.

3. The intensity changes a little bit on the blue channel. The range is from *4* to *27*

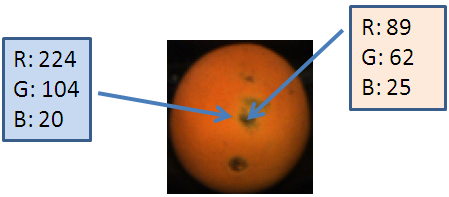
According to the data analysis along vertical and horizontal directions, the brightness increases from outside to inside. The important intensity changes occur on the red and green channel. For a uniform orange image, the intensity changes can be shown as follow.



*Fig.10. Intensity Layers*

If all the intensities within a specific range are painted using the same color, then the image can form certain layers. The intensity range selected in the above example is sixty. The intensity range can be changed to a smaller range if more intensity layers are required. The intensities between *61* and *120* are painted using blue color, the intensities between *121* and *180* are painted using grey color, and so on. The intensity increases from outside to inside. The intensity layers are actually similar to the RGB Color Classes. The brightness of RGB Color Classes increases from class one to eight. There are eight RGB Color Classes which can be matched to eight intensity layers. But the intensity range of each layer varies for each orange image, so it cannot be set as a constant value. Compute eight RGB Color Classes can help to set up each intensity layer in an accurate and dynamic way. The RGB Class Mean is the measure of average intensity for each RGB Color Class.

The following example shows the intensity changes between normal skin and blemished skin.



*Fig.10. Normal Skin and Blemished Skin*

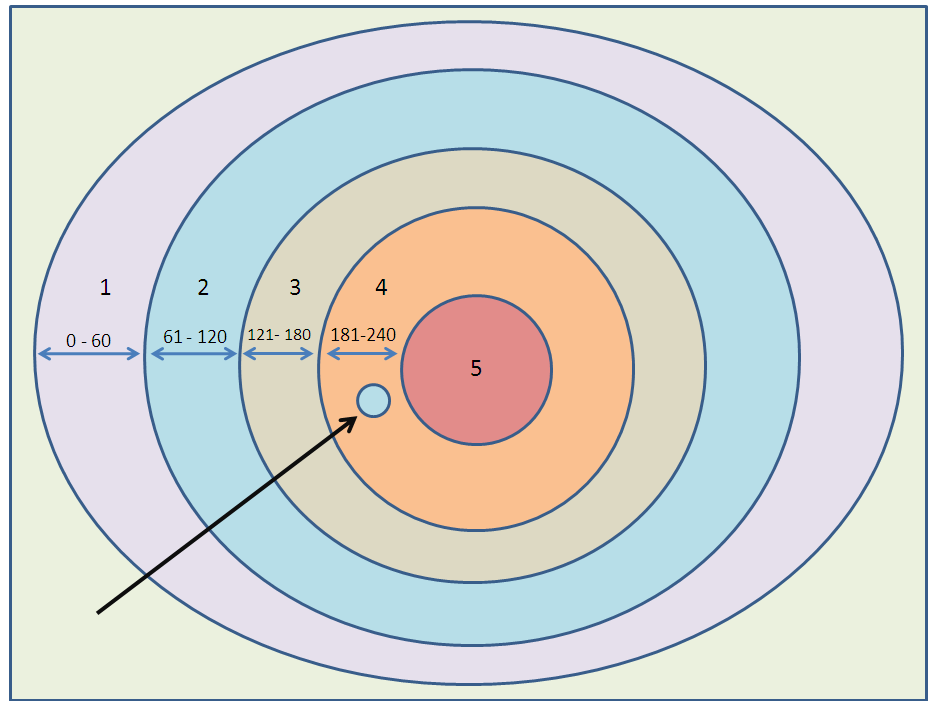
Two pixels are selected for analysis. The left one is selected from the normal orange skin, and the right one is selected from the blemished orange skin. It is clearly to see that the blemished orange skin is much darker than the normal orange skin.

1. The intensity decreases a lot on the red channel from the normal skin to blemished skin. In this case, the intensity dropped from *224* to *89*. The most significant intensity change occurs on the red channel.

2. The intensity decreases many on the green channel from the normal skin to blemished skin. In this case, the intensity dropped from *104* to *62.* This is the second important intensity change on the green channel.

3. The intensity increases a litter bit on the blue channel from the normal skin to blemished skin. In this case, the intensity increased from *20* to *25.* This may be caused by the texture of the blemished skin. The intensity change on the blue channel is very minor, so it is not important.

The intensity change between normal skin and blemished skin is focused on the red and green channel. If the intensity dropped a lot in an area compared with neighbors, then that area is of interest to analysis. The following example shows a blemished area on a uniform orange image.

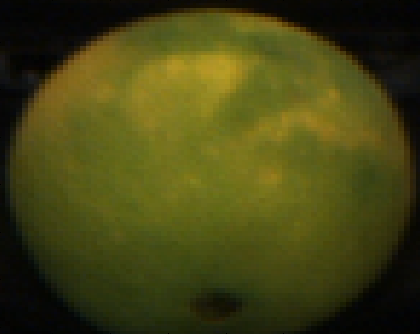


*Fig.10. Blemish Identification*

All the pixels on layer four should be painted using orange color. In another word, the intensity on layer four should be between *181* and *240*. However, the blue spot on layer four does not belong to it. In another word, the intensity of the blue spot is lower than the intensity of layer four. The intensity of the blue spot is within the range of layer two. That is why the blue spot on layer four is painted using the same color as layer two. The intensity on layer two is between *61* and *120.* That means the blue spot has low intensity compared with its neighbors. So the blue spot is identified as a blemished area.

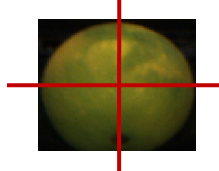
* + - 1. **Effects of Illumination Intensity Variations on Unripe Orange Skin**

Some oranges have green skin color. The following is an example of green skin.



*Fig.10. Green Skin*

The brightness of a green orange image increases from outside to inside. The selected sample image has a size of 140 by 140. Two red lines are drawing across the centre of this image for further analysis.



*Fig.10. Intensity Change*

The following data shows how the intensity changes along the red lines.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Intensity Changes along the Horizontal Line | | | | | | | | | | | | | |
| cols | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 | 130 |
| R | 70 | 87 | 96 | 113 | 121 | 137 | 144 | 114 | 110 | 105 | 89 | 77 | 68 |
| G | 71 | 84 | 101 | 107 | 111 | 115 | 118 | 108 | 102 | 100 | 81 | 73 | 70 |
| B | 6 | 19 | 24 | 27 | 28 | 27 | 29 | 25 | 25 | 19 | 18 | 16 | 10 |

*Fig.10. Horizontal Intensity Change*

The pink line in the above diagram is the column selected along the horizontal red line. The center of this horizontal red line is row seventy and column seventy. All the other columns are selected using symmetry principle, such as column sixty and column eighty. It is clearly to see that the intensity increases from outside to inside for all three channels.

1. The intensity varies between *68* and *144* on the red channel. The intensity on green skin orange does not change too much compared with the orange skin orange. The intensity on the red channel cannot dominate the intensity changes by itself.

2. The intensity varies between *70* and *118* on the green channel*.* The pixel intensities between the red and green channel are similar. This is because the orange skin color is green in this case. The intensity change on the green channel becomes more important.

3. The intensity varies between *6* and *29* on the blue channel*.* The intensity change on the blue channel is less important than the others.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Intensity Changes along the Vertical Line | | | | | | | | | | | | | |
| rows | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 | 130 |
| R | 86 | 96 | 117 | 120 | 124 | 137 | 147 | 134 | 115 | 109 | 104 | 70 | 49 |
| G | 71 | 88 | 94 | 106 | 106 | 112 | 114 | 106 | 90 | 75 | 56 | 27 | 18 |
| B | 14 | 18 | 19 | 20 | 24 | 24 | 30 | 28 | 23 | 19 | 18 | 4 | 2 |

*Fig.10. Vertical Intensity Change*

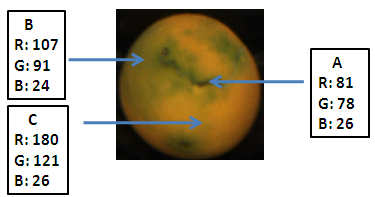
The pink line in the above diagram is the row selected along the vertical blue line. The center of this vertical blue line is row seventy and column seventy. All the other rows are selected using symmetry principle, such as row sixty and row eighty. It is clearly to see that the intensity increases from outside to inside for all three channels.

1. The intensity varies between *49* and *147* on the red channel. The intensity change on the red channel reduced to a smaller range compared with the orange skin.

2. The intensity varies between *18* and *114* on the green channel*.* The pixel intensities towards to the centre part of the orange are similar between the red and green channel. The light source around the bottom edge is not very good. It is getting very dark in there.

3. The intensity varies between *2* and *30* on the blue channel*.* The intensity change on the blue channel is less important than the others. However, the intensity change on the blue channel is more important compared with the orange skin.

According to the data analysis along both vertical and horizontal directions, the brightness increases from outside to inside. The intensity changes on both red and green channels are important. But the intensity changes on the red channel have different meanings with the intensity changes on the green channel. The following example shows the intensity changes on the orange and green skin.



*Fig.10. Skin Analysis*

1. “A” is a pixel randomly selected from a blemished area on green skin. “A” stands for the blemishes on a green skin.

2. “B” is a pixel randomly selected from the green skin. “B” stands for the green skin.

3. “C” is a pixel randomly selected from the orange skin. “C” stands for the orange skin.

There are three intensity changes which need to be focused on. The intensity changes from C to B, B to A, and C to A.

1. Intensity Change from C to B
2. The intensity on the red channel dropped from *180* to *107*. This is a significant change which means that the pixel intensities on the red channel are very sensitive to the skin color change. Whenever the skin color changed from orange to green, then the pixel intensities on the red channel will drop a lot.
3. The pixel intensity on the green channel dropped from *121* to *91.* The intensity change on the green channel is smaller than the one on the red channel. The intensity drops when the skin color changed from orange to green.
4. The intensity on the blue channel changed a little bit which is not important in this case.

In conclusion, the intensity dropped on both red and green channels when the skin color changed from orange to green. In this case, the intensity change on the red channel is more important.

1. Intensity Change from C to A
   1. The intensity on the red channel dropped from *180* to *81*. The blemished skin is always much darker than normal skin. The intensity change on the red channel is very obvious.
   2. The pixel intensity on the green channel dropped from *121* to *78.* The intensity change on the green channel is smaller than the one on the red channel. The intensity on the green channel is also sensitive to the blemishes. This is the second important compared to the red channel.
   3. The intensity on the blue is unchanged. The intensity on the blue channel is not important.

In conclusion, the intensity dropped on both red and green channels from a normal orange skin to a blemished orange skin. In this case, the intensity change on the red channel is more important.

1. Intensity Change from B to A

1. The intensity on the red channel dropped from *107* to *81*. The pixel intensity on the red channel is not very high on a green skin area.

2. The pixel intensity on the green channel dropped from *91* to *78.* The intensity change on the green channel is very important when analyze the blemishes on the green skin.

3. The intensity on the blue channel is getting a little bit higher. This is caused by the texture of the blemished skin.

In conclusion, the intensity dropped on both red and green channels from a normal green skin to a blemished skin. In this case, the intensity change on the green channel is more important.

From the above examples, it is clearly to see that the intensity dropped twice on the red channel.

1. The skin color changed from orange to green.

2. The skin color changed from green to blemish.

It is hard to decide which intensity change is from a normal skin and a blemished skin. So the intensity drop on the green channel becomes more important. Whenever the intensity dropped a lot on the green channel for a green skin, then a blemished area is detected.

* + 1. **Summary**

Class mean is the measurement of the central tendency for each RGB Color Class. For a given data set, the class mean is the sum of the measurements divided by the number of the measurements and to compute a number as being the average. The formula of mean is defined as follow:

***M = 1/n* ∑*ai = 1/n (a1 + a2 +.... + an),***

***where* *1 ≤ i ≤ n***

Sometimes a set of numbers might contain outliers. The outlier is the intensity of a pixel which is much lower or much higher than the others. The outliers are erroneous data caused by different reasons, such as leaves on the conveyor. So a more accurate way to compute the class mean is to remove the outliers first, and then compute the arithmetic class mean. There are three steps involved in this computation.

1. Sort the selected data set.

2. Discard an equal amount of data at the high and the low end. For most statistical applications, five to twenty-five percent of the ends are discarded.

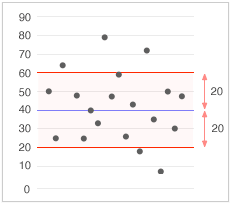
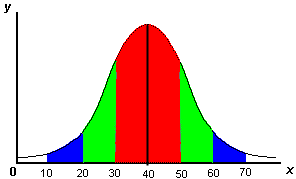
3. Compute the arithmetic mean of the remaining data.

Ten percent of the ends are discarded in the real application. The modified arithmetic mean can improve the accuracy of this algorithm.

After some experiments and data analysis, it is clearly to see that the brightness of an orange image increases from outside to inside. This is the nature of light source on circular object which can be explained in physics.

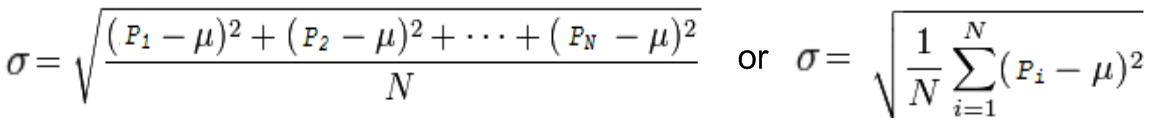
* 1. **Class Standard Deviation** 
     1. **Overview**

In statistics, standard deviation is a simple measure of the spread of a data set. Standard deviation often can find the story behind the data, such as the tightness of data samples that are clustered around the mean. A low standard deviation indicates that all the pixel intensities are very close to the same value(class mean), while the high standard deviation indicates that all the pixel intensities are clearly more spread out across a large range of levels. For example, the average intensity of class one on the red channel is about 40, with a standard deviation of 0. This means that all the pixels have exactly 40 intensity level. If the standard deviation is 20, then pixels have much more variable intensities, with a typical range of about 20 to 60. The charts below represent a normal distribution with a mean of 40 and a standard deviation of 20.

*Fig.10. Standard Deviation*

Normal distribution is also called the Gaussian distribution. A set of data is described as a symmetric bell shaped curve. It provides a visual relationship between probability and area. If the bell-shape curve is steep, then the standard deviation is low and data samples are tightly bunched together. If the bell-shape curve is relatively flat, then the standard deviation is high and data samples are spread apart. The formula of standard deviation is defined as follow:

******

*P* stands for the pixel intensity value, *μ* is the class mean value, and *N* is the number of pixels in the class. There are total four steps to compute the standard deviation.

1. For each pixel intensity value *P*, subtract the class mean value *μ* and compute the square of the difference.
2. Sum up all the squared values and divide by the total number of pixels.
3. Compute the square root of the value in Step 2.

There are eight RGB Color Classes and three color channels in a given RGB image, so twenty-four class standard deviations would be computed in total. Standard deviation is a useful measure of the dispersion of a RGB color class members.

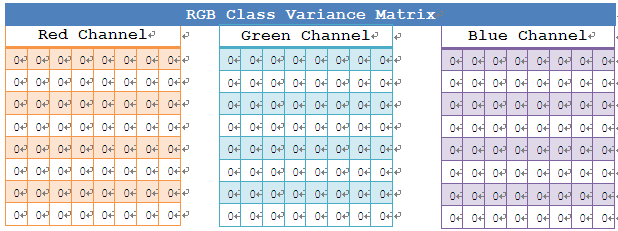
* + 1. **Algorithm**

1. Initialize RGB Class Variance Matrix. RGB Class Variance Matrix is a three-dimensional array which consists of eight rows and eight columns. The elements of a three-dimensional array can be thought of as a set of two-dimensional arrays. The first, second, and third dimensions represent the red, green, and blue channels respectively.
2. Compute class standard deviation for each class on three channels respectively. There are eight classes on each of these three channels, therefore twenty-four class standard deviations need to be computed.
3. Step through each class from class one to eight.
4. Step through each pixel in the RGB image.
5. Map the class number for the pixel using RGB Class Distribution Matrix.
6. Map the intensity values for the pixel using red, green, and blue Channels respectively. In the RGB image, a pixel has three intensity values. One is for the red channel, one is for the green channel, and one is for the blue channel.
7. Compute the difference of the pixel intensity from its corresponding class mean and square the result.
8. Sum up all the squared difference.
9. Compute the class standard deviations for each channel separately.

3. Store the computed class standard deviations in the RGB Class Variance Matrix for further analysis.

* + 1. **Implementation**

1. Initialize RGB Class Variance Matrix.



*Fig.10. RGB Class Variance Matrix*

*For dimension = 1 to 3*

*For rows = 1 to 8*

*For cols = 1 to 8*

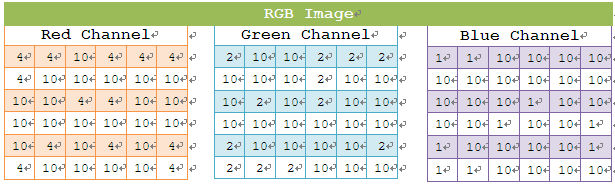
*classVariance(rows, cols, dimension) = 0*

*End*

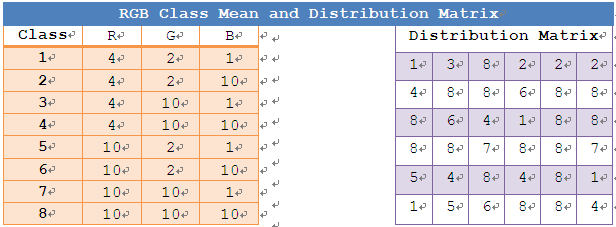
*End*

*End*

1. Compute class standard deviation for each class on three channels separately. The following is a fraction of data extracted from a RGB image for demonstration purpose.



*Fig.10. RGB Image*



*Fig.10. Class Mean and Distribution Matrix*

*For classNo = 1 to 8*

*For i = 1 to rows*

*For j = 1 to cols*

*IF classDistributionMatrix(i, j) = classNo*

*/\**

*Compute the difference of the pixel intensity from the class mean and square the difference. For instance:*

*(RedChannel(i,j) - classMean(classNo,1)).^2*

*(BlueChannel(i,j) - classMean(classNo,3)).^2*

*\*/*

*.........................................*

*//Sum all the squared difference*

*.........................................*

*End*

*End*

*End*

*// Compute within-class standard deviation*

*..........................................*

*End*

The following computation demonstrates how the Class Standard Deviation of class one on the red channel is computed.

* 1. Find all the pixel locations of class one using RGB Class Distribution Matrix.

*P(1,1), P(6,1), P(3,4), P(5,6)*

Find the intensity value for each pixel on the red channel.

*P(1,1) = 4, P(6,1) = 4, P(3,4) = 4, P(5,6) = 4*

* 1. Compute the difference of the pixel intensity from the class mean of class one on the red channel. The value of this class mean is 4.

*P(1,1) – 4 = 4 – 4 = 0, P(6,1) – 4 = 4 – 4 = 0,*

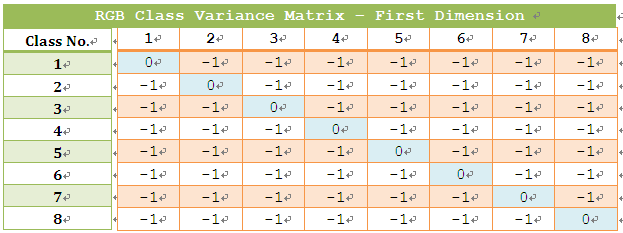
*P(3,4) – 4 = 4 – 4 = 0, P(5,6) – 4 = 4 – 4 = 0*

* 1. Sum all the squared differences and divide by the number of pixels.

*(0.^2 + 0.^2 + 0.^2 + 0.^2) / 4 = 0 / 4 = 0*

* 1. Compute the square root of the value in Step 4.

*sqrt(0) = 0;*

3. Store the computed Class Standard Deviations in the RGB Class Variance Matrix for further analysis. 

*Fig.10. RGB Class Variance Matrix*

The above diagram illustrates the first dimension of the RGB Class Variance Matrix. It is derived based on the pixel intensity values from the red channel. The standard deviation of class one is stored in the first row and first column, the standard deviation of class two is stored in the second row and second column, and so on.

* + 1. **Summary**

Class Standard Deviation is a simple measure of the spread of a data set. A low Class Standard Deviation indicates that all the pixel intensities are very close to the Class Mean, which the high Class Standard Deviation indicates that all the pixel intensities are clearly more spread out across a large range of levels. The Class Standard Deviation is derived from the original standard deviation in statistics. There are four steps to compute the original standard deviation.

1. For each pixel intensity value *P*, subtract the class mean value *μ* and compute the square of the difference.
2. Sum up all the squared values and divide by the total number of pixels.
3. Compute the square root of the value in Step 2.

There are four basic steps involved to compute the Class Standard Deviation.

1. For a given RGB image, extract red, green, and blue channel separately.
2. Categorize pixels using RGB Class Distribution Matrix
3. Compute the Class Standard Deviation for each class.
4. Compute the Class Standard Deviation for each isolated channel.
5. Store all the computed Class Standard Deviations in the RGB Class Variance Matrix.
   1. **Class Distance**

**4.5.1 Overview**

Class Distance is a numerical description of how far apart RGB Color Classes are, e.g., the distance between class one and class two is *35.2*. In mathematics, there are two common methods to compute the distance between two objects, e.g., absolute difference and squared difference. Absolute difference is a numerical value without regard to its sign. The following example shows how the absolute difference works.

*A = 8, B = 6,*

*Diff1 = A – B = 2, Diff2 = B – A = -2,*

*absDiff1 = |A - B| = 2, absDiff2 = |B – A| = 2*

Squared difference is squared numerical value without regard to its sign. The following example shows how the squared difference works.

*A = 8, B = 6,*

*Diff1 = A – B = 2, Diff2 = B – A = -2,*

*sqrtDiff1 = (A – B).^2 = 4, sqrtDiff2 = (B – A).^2 = 4*

Absolute difference and squared difference basically works the same on the way of regarding to its sign. The following example demonstrates that both methods can be used as a comparison method between RGB Color Classes.

*A = 8, B = 6, C = 11,*

*absDiff1 = |A - B| = 2,*

*absDiff2 = |A - C| = 3,*

*sqrtDiff1 = (A – B).^2 = 4, sqrtDiff2 = (A - C).^2 = 9,*

*absDiff1 < absDiff2, sqrtDiff1 < sqrtDiff2*

Both methods show that the distance between A and B is shorter than the distance between A and C, however, the squared difference enlarged the real distance between two objects. The formula of Class Distance is defined as follow:

***Dist = (ClassMeanA – ClassMeanB).^2***

*ClassMeanA* and *ClassMeanB* are class means of two different classes on the same channel. Class Distance is a measure of the squared difference between two class means. A small distance indicates that two classes are close to each other, and a big distance indicates that two classes are far away from each other. If two classes are close enough to each other, then they might be merged together to form a new class.

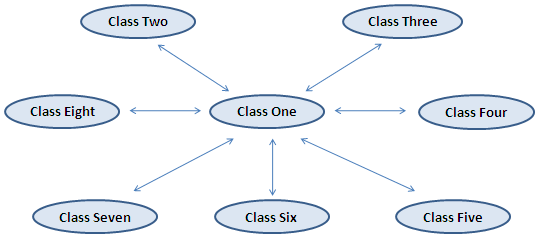
* + 1. **Algorithm**

1. Compute Class Distance for each class on three channels separately.
2. Step through each class from class one to eight. The distances to be computed are listed as follow:

**Class One:**

The distances to be computed are:

* *Class One to Class Two*
* *Class One to Class Three*
* *Class One to Class Four*
* *Class One to Class Five*
* *Class One to Class Six*
* *Class One to Class Seven*
* *Class One to Class Eight*

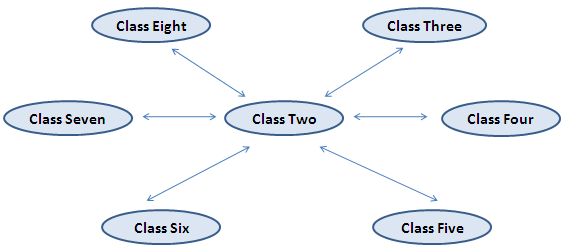


*Fig.10. Distance for Class One*

**Class Two:**

The distances to be computed are:

* *Class Two to Class Three*
* *Class Two to Class Four*
* *Class Two to Class Five*
* *Class Two to Class Six*
* *Class Two to Class Seven*
* *Class Two to Class Eight*

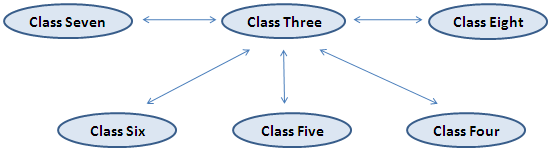


*Fig.10. Distance for Class Two*

**Class Three:**

The distances to be computed are:

* *Class Three to Class Four*
* *Class Three to Class Five*
* *Class Three to Class Six*
* *Class Three to Class Seven*
* *Class Three to Class Eight*

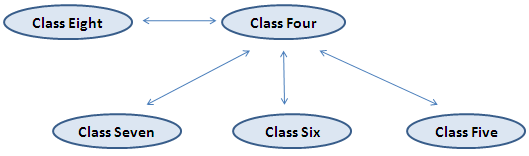


*Fig.10. Distance for Class Three*

**Class Four:**

The distances to be computed are:

* *Class Four to Class Five*
* *Class Four to Class Six*
* *Class Four to Class Seven*
* *Class Four to Class Eight*

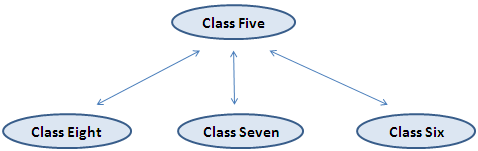


*Fig.10. Distance for Class Four*

**Class Five:**

The distances to be computed are:

* *Class Five to Class Six*
* *Class Five to Class Seven*
* *Class Five to Class Eight*



*Fig.10. Distance for Class Five*

**Class Six:**

The distances to be computed are:

* *Class Six to Class Seven*
* *Class Six to Class Eight*



*Fig.10. Distance for Class Six*

**Class Seven:**

The distance to be computed is:

* *Class Seven to Class Eight*



*Fig.10. Distance for Class Seven*

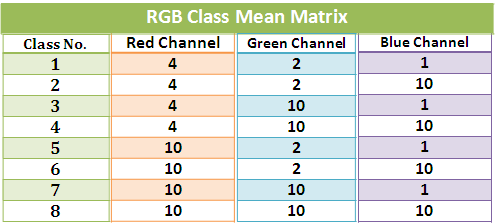
The distance from class one to class eight is the same as the distance from class eight to class one. The distances from class eight to all the other classes have been computed previously. So there is no need to compute any distance apart from class eight. There are total twenty-eight RGB Class Distances to be computed.

1. Step through each channel to compute Class Distances.

2. Store the computed Class Distances in the RGB Class Variance Matrix for further analysis.

* + 1. **Implementation**

1. Compute Class Distance for each class on three channels separately.



*Fig.10. RGB Class Mean Matrix*

*For classNo = 1 to 7*

*For neighbours = classNo + 1 to 8*

*/\**

*Compute RGB Between-Class Distance.*

*For instance:*

*(classMean(classNo,1) - classMean(neighbours,1)).^2*

*(classMean(classNo,2) - classMean(neighbours,2)).^2;*

*\*/*

*.....................................*

*End*

*End*

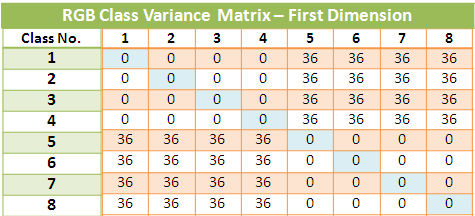
The following computation demonstrates how the RGB Class Distance from class one to class eight on the red channel is calculated.

*classMean(1,1) = 4, classMean(8,1) = 10,*

*Dist = (classMean(1,1) - classMean(8,1)).^2*

*= (4 - 10).^2 = 36*

1. Store the computed Class Distances in the RGB Class Variance Matrix for further analysis.

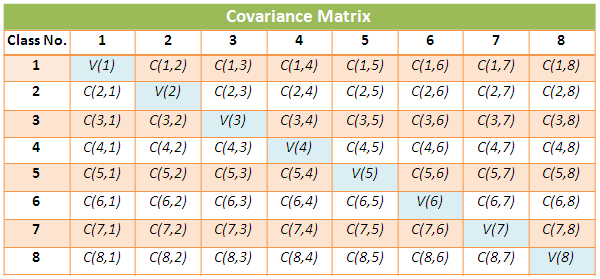


*Fig.10. RGB Class Variance Matrix*

The above diagram illustrates the first dimension of the RGB Class Variance Matrix. It is derived from the class means on the isolated red channel. The distance between class one and class two is stored in the second row and first column, the distance between class one and class three is stored in the third row and first column. The distance in the second row and first column is the same as the distance in the first row and second column, the distance in the third row and first column is the same as the distance in the first row and third column, and so on. In statistics, a matrix of covariances between elements of a vector is called covariance matrix. The formula is defined as follow:

***Dist[i,j] = Cov(classNoi, classNoj), and***

***Dist[i] = Var(classNoi)***



*Fig.10. Covariance Matrix*

*Dist[i]* stands for the Class Standard Deviation, and *Dist[i,j]* stands for the Class Distance. Data along the blue diagonal line in the above covariance matrix follows the symmetry principle, e.g., *C(2,1)* is the same as *C(1,2)* in terms of distance between two objects. The distance between two objects is always a positive value.

* + 1. **Summary**

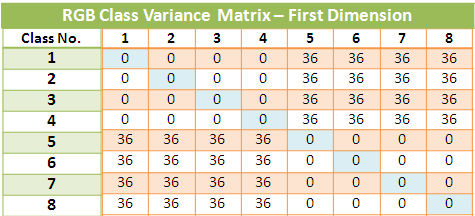
Class Distance is a numerical description of how far apart RGB Color Classes are. Absolute difference and squared difference are two common methods to compute the distance between two objects. However, the squared difference enlarged the real distance between two objects. A small distance indicates that two classes are close to each other, and a big distance indicates that two classes are far away from each other. The formula of Class Distance is defined as follow:

***Dist = (ClassMeanA – ClassMeanB).^2***

There are four basic steps to compute the Class Distances:

1. For a given RGB image, extract red, green, and blue channel separately.
2. Categorize pixels using RGB Class Distribution Matrix.
3. Compute the Class Distance for each class.
4. Compute the Class Distance on each channel.

Store all the computed Class Distances in the RGB Class Variance Matrix for further analysis

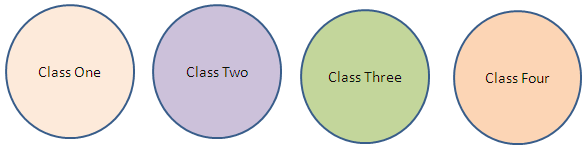


*Fig.10. RGB Class Variance Matrix*

* 1. **Closest Neighbor Class**

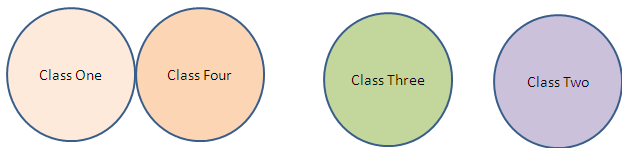
**4.6.1 Overview**

There are total eight RGB Color Classes. Some of them can be very similar to each other, so the target in this section is to identify the classes which are closest to each other. The term neighbor referred in this section means all the other classes except itself, and the term closest means the most similar neighbor compared to all other neighbors. The word similar in terms of color expression means that two classes have closer visual impact. The following example shows the similarity of RGB Color Classes.



*Fig.10. Similarity of RGB Color Classes*

From the above diagram, it is clearly to see that the closest neighbor of class one is class four. Class two looks very different from the other three classes. In another word, class two has three neighbors, but no closest neighbor. It is a standalone class which is close to itself. Class three is the same as class two, and no closest neighbor. The closest neighbor of class four is class one.



*Fig.10. Closest Neighbor Class*

Of course, the similarity among eight RGB Color Classes cannot be measured by human eye in the real application. RGB Class Variance Matrix is the data source which consists of the measurements about Class Distance. The RGB Class Variance Matrix is a three dimensional array with eight rows and eight columns. Compute the average distance of two classes among three channels for further analysis. The following example shows how the average distance from class one to its neighbor class two are calculated.

*distR[1,2] = 80 Red Channel,*

*distG[1,2] = 70 Green Channel,*

*distB[1,2] = 6 Blue Channel,*

*averageDist[1,2] = (80 + 70 + 6) / 3 = 52*

The value *52* is the average distance from class one to class two. In statistics, arithmetic average of a set of values is the one of the most commonly used statistical measurements, however, for skewed distributions, this average is not accurate. For example, the arithmetic average of six values: 20, 18, 17, 16, 10, 1 is:

*(20 + 18 + 17 + 16 + 10 + 1) / 6 = 13.6667*

The average in the above example is skewed downwards by a few numbers with very small values, however, the majority have a value bigger than *10*. The Class Distance among three channels is a set of skewed values. The average distance between class one and class two is skewed downwards by a number with very small value. This small value is the Class Distance between class one and class two in the blue channel. The distance between class one and class two in blue channel is very small compared with the distance in the red channel and green channel. The assumption is that the class distance in the red channel is more important than the one in the green channel, and the Class Distance in the green channel is more important than the one in the blue channel. The assumption is based on the texture of the orange. The color of an orange is more likely to be red, maybe a little bit green, and almost no blue. To compute an average among a set of skewed values is a challenge. In statistics, there are three other main algorithms to compute the mean, e.g., quadratic mean, geometric mean, and harmonic mean.

1. Quadratic Mean

Quadratic mean is also called power mean.

1. Compute the power of each element in the data set and sum the results.

*80.^2 + 70.^2 + 6.^2 = 11336*

2. Divide the sum by the number of elements in the data set.

*6400 / 3 = 3779*

3. Compute the square root of the value in step two.

*sqrt(2133) = 61*

1. Geometric Mean

1. Multiply all the elements in the data set.

*80 \* 70 \* 6 = 33600*

2. Compute the one-third power of the multiplication.

*336001/3 = 32*

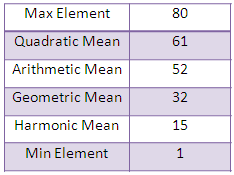
1. Harmonic Mean
2. Divide each element by one and sum the results.

*1/80 + 1/70 + 1/6 = 0.1935*

1. Divide the value in step two by the number of elements.

*3 / 0.1935 = 15*

The quadratic mean is an average value which is more towards to the maximum elements in the data set, harmonic mean is more towards to the minimum elements, and arithmetic mean and geometric mean are in between. The following diagram shows the relationship among these means.



*Fig.10. Mean*

Class Distance between two classes in the red channel is often bigger than the one in the green channel, and much bigger than the one in the blue channel. The Class Distance in the red channel is more considered than the other two channels. Class distance among three channels is a typical skewed distribution case, and quadratic mean is the best algorithm to measure the average distance between classes.

.

**4.6.2 Algorithm**

1. Compute the quadratic mean for each Class Distance in RGB Class Variance Matrix.
2. Store the computed quadratic means in the Mean Variance Matrix with eight rows and eight columns.
3. Find the closest neighbor for each RGB Color Class based on the Mean Variance Matrix.
4. Store the closet neighbor for each RGB Color Class in the Closest Neighbor Class Matrix. Closest Neighbor Class Matrix a two dimensional array with eight rows and eight column.

**4.6.3 Implementation**

1. Compute the quadratic mean for each Class Distance.

*For i = 1 to 8*

*For j = 1 to 8*

*/\**

*Compute quadratic mean.*

*For instance:*

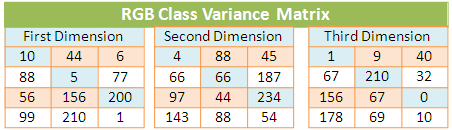
*(classVariance(i,j,1).^2 + classVariance(i,j,2).^2 + classVariance(i,j,3).^2) / 3*

*\*/*

*........................................*

*End*

The sample data set selected is a subset of the RGB Class Variance Matrix.



*Fig.10. RGB Class Variance Matrix*

The following computation demonstrates how the quadratic mean of class one is calculated.

*classVariance(1,1,1) = 10,*

*classVariance(1,1,2) = 4,*

*classVariance(1,1,3) = 1,*

*(10.^2 + 4.^2 + 1.^2) / 3 = 39*

*Dist[1] = sqrt(39) = 6*

The following computation demonstrates how the quadratic mean from class one to class two is calculated.

*classVariance(1,2,1) = 44,*

*classVariance(1,2,2) = 88,*

*classVariance(1,2,3) = 9,*

*(44.^2 + 88.^2 + 9.^2) / 3 = 3254*

*Dist[1,2] = sqrt(3254) = 57*

The following computation demonstrates how the quadratic mean from class one to class three is calculated.

*classVariance(1,3,1) = 6,*

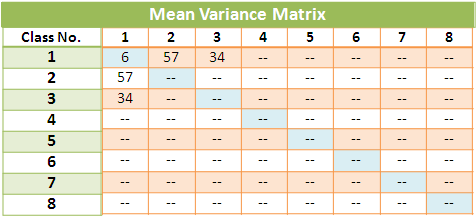
*classVariance(1,3,2) = 45,*

*classVariance(1,3,3) = 40,*

*(6.^2 + 45.^2 + 40.^2) / 3 = 1220*

*Dist[1,3] = sqrt(1220) = 34*

1. Store the computed quadratic means in the Mean Variance Matrix with eight rows and eight columns. The following diagram shows how the *Dist[1]*, *Dist[1,2]*, and *Dist[1,3]* are stored in the Mean Variance Matrix.



*Fig.10. Mean Variance Matrix*

1. Find the Closest Neighbor Class for each RGB Color Class based on the Mean Variance Matrix.

*For i = 1 to 8*

*minVariance = 10000000000;*

*closestNeighbor = 0;*

*For j = 1 to 8*

*If(minVariance > meanVariance[i,j])*

*minVariance = meanVariance[i,j]*

*closestNeigbor = j*

*End*

*End*

*End*

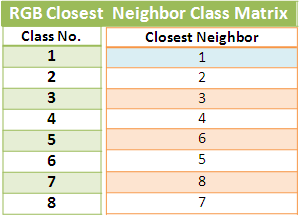
The above algorithm is used to search the Closest Neighbor Class for each RGB Color Class. The *minVariance* is initialized to a very big value for comparison purpose. The closest neighbor of class one according to the above diagram is itself, e.g.,

*Dist[1] > Dist[1,2], 6 > 57*

*Dist[1] > Dist[1,3], 6 > 34*

This means class one is more like to be a standalone class.

1. Store the closet neighbor for each RGB Color Class in the RGB Closest Neighbor Class Matrix. RGB Closest Neighbor class Matrix a two dimensional array with eight rows and eight cols. This matrix shows the closest neighbor class for each RGB Color Class, e.g.,

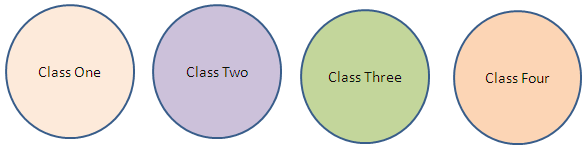


*Fig.10. Closest Neighbor Class Matrix*

From the above diagram, it is clearly to see class one to four are standalone classes, class five and six are Closest Neighbor Classes, and class seven and eight are Closet Neighbor Classes.

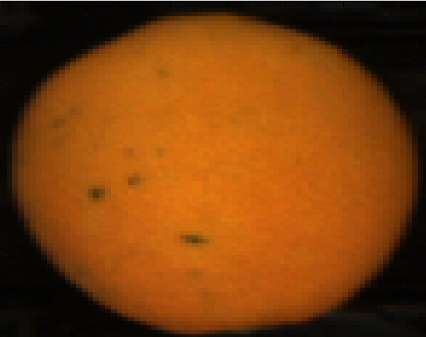
**4.6.4 Experiment and Analysis**

If two RGB Color Classes are similar to each other, then these two classes should have similar visual impact.



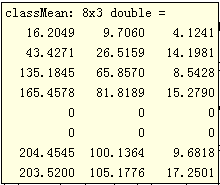
*Fig.10. Similarity of RGB Color Classes*

Class one and four looks similar to each other. In another word, the Class Means of these two classes should be similar as well. The following image is a ripe orange. Ripe orange has orange skin color.



*Fig.10. Ripe Orange*

After computation, the Class Means of the above image are listed as follow:



*Fig.10. Class Mean*

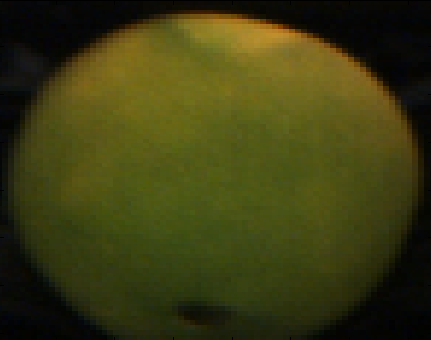
The first column is the Class Means on the red channel, the second column is the Class Means on the green channel, and the third column is the Class Means on the blue channel. The first row is the Class Means for class one, the second row is the Class Means for class two, and so on. Class five and class six are not available, so their Class Means are filled with zeros. It is clearly to see that class seven and eight have very similar Class Means for all three channels. So the Closet Neighbor Class of class seven is class eight, and the Closet Neighbor Class of class eight is class seven. This can be explained easily using the texture of ripe oranges. The following is the structure of RGB Color Classes.



*Fig.10. RGB Color Class*

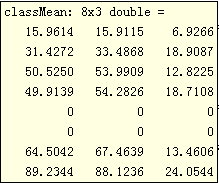
Class seven is a combination of red and green color, and class eight is a combination of red, green and blue color. The blue color for an orange is very minor, so red and green color dominates the Class Means. Class seven and eight both have red and green color components, so they should be very similar in common sense.

The following image is an unripe orange. Unripe orange has green skin color.



*Fig.10. Unripe Orange*

After computation, the Class Means of the above image are listed as follow:



*Fig.10. Class Mean*

Class five and class six are not available, so their Class Means are filled with zeros. It is clearly to see that class three and four have very similar Class Means for all three channels. So the Closet Neighbor Class of class three is class four, and the Closet Neighbor Class of class four is class three. This can be explained easily using the texture of unripe oranges. Class three consists of pure green color, and class four is a combination of green and blue color. The blue color for an orange is very minor, so green color dominates the Class Means. Class three and four both have green color component, so they should be very similar in common sense.

In conclusion, class seven and eight are normally similar to each other for a ripe orange, and class three and four are normally similar to each other for an unripe orange.

**4.6.5 Summary**

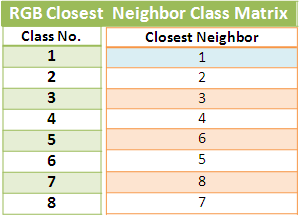
Some of RGB Color Classes could be very similar to each other. It is very important to indentify Closest Neighbor Classes for further analysis. The RGB Class Variance Matrix is the data source which consists of the measurements about Class Distance. The average distance between two classes is computed using quadratic mean. The formula is defined as follow:

*quadMean*

*= (classMeanR.^2 + ClassMeanG.^2 + classMeanB.^2)/3*

The average distance between two classes is skewed downwards by the Class Distance in blue channel. So the Class Distance between two classes is considered as a set of skewed values. The assumption is that the Class Distance in the red channel is more important than the one in the green channel, and the Class Distance in the green channel is more important than the one in the blue channel. The assumption is based on the texture of the orange. The color of an orange is more likely to be red, maybe a little bit green, and almost no blue.

After some experiments, it is clearly to see that class seven and eight are normally similar to each other for a ripe orange, and class three and four are normally similar to each other for an unripe orange.

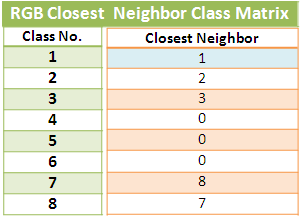


*Fig.10. RGB Closest Neighbor Class Matrix*

* 1. **Class Reclassification**

**4.7.1 Overview**

There are total eight RGB Color Classes. Some of them may be very similar or close to each other. The Closest Neighbor Class Matrix shows the relationship among those classes. There are three different types of classes which will be introduced in this section, such as standalone class, missing class and similar class. The following diagram shows an instance of the RGB Closest Neighbor Class Matrix.



*Fig.10. RGB Closest Neighbor Class Matrix*

1. **Standalone Class**

A class has no closest neighbor. Class one, two and three are standalone classes. Pixels in the standalone class are different from other pixels in terms of color components and brightness.

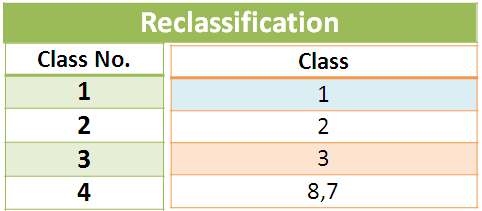
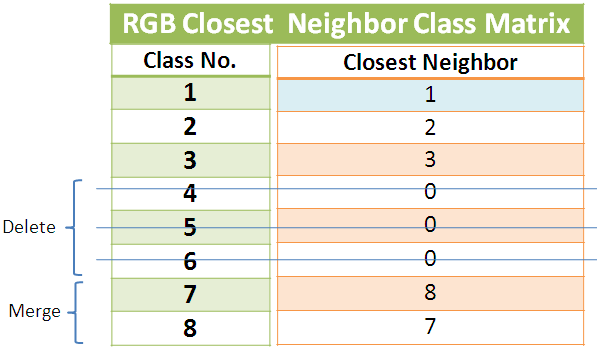
1. **Missing Class**

A class does not exist in a RGB image. Class four, five, and six are missing classes. Zero in the Closest Neighbor Class Matrix indicates that the current class is not available.

1. **Similar Class**

Two classes are close to each other. The closest neighbor of class seven is class eight, and the closet neighbor of class eight is class seven. Class seven and class eight are very similar to each other, so they can be merged to form a new class.

After class reclassification, the new RGB Color Class looks like follow.



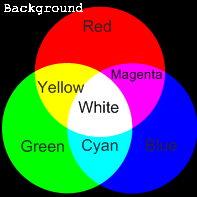
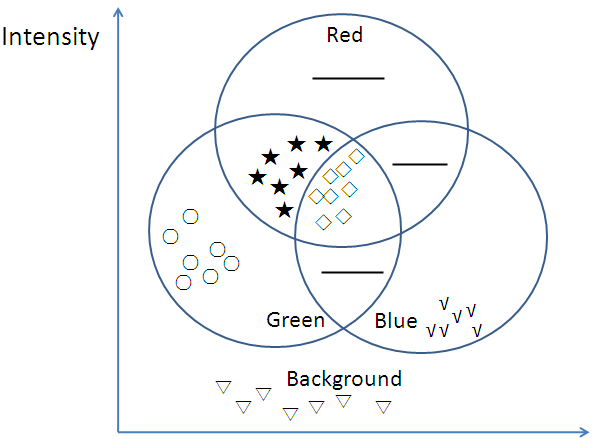
*Fig.10. Class Reclassification*

There are four RGB Color Classes left after class reclassification. Class one, two, and three stay the same, class four, five, and six are deleted, and class seven and eight are merge together. The following diagram shows the relationship among four classes.



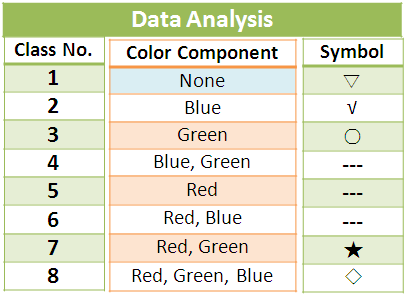
*Fig.10. New RGB Color Classes*

Class one is the RGB Background Class, class two is the RGB Blue Class, class three is the RGB Green Class, and class four is a combination of RGB Yellow Class and RGB White Class. In practice, the blue color in an orange image is very minor, so RGB Yellow Class and RGB White Class are normally similar to each other. In another word, the combination of red and green color is very similar to the combination of red, green, and blue color for orange image only. The RGB Color Classes can also be defined as RGB Color Clusters. Colors within the same cluster are more similar to each other than the colors in a different cluster. The similarity is assessed base on the measurement of the Class Distance. Clustering is a common technique for statistical data analysis.

*Fig.10. RGB Color Clusters*

The left diagram shows the classification of eight RGB Color Classes. The right diagram shows the assignment of colors into clusters. The process of assigning a pattern to one of a number of pre-defined classes is called classification. A pattern is a set of measurements, e.g., the combination of colors and brightness. The following diagram shows the symbols used for each RGB Color Cluster.



*Fig.10. Data Analysis*

There are four distinct clusters in the above diagram.

1. Cluster one contains all the pixels with very low intensity. It is different from all other clusters. In this case, all the pixels in this cluster belong to the background.
2. Cluster two contains all the pixels with natural green color.
3. Cluster three contains all the pixels with natural blue color. Not many pixels belong to this cluster due to the nature of the orange.
4. Cluster four contains all the pixels with the combination of red and green color. Cluster five contains all the pixels with the combination of red, green, and blue color. Cluster four and five are actually very close to each. Two clusters can be merged to form one distinct cluster.

**4.7.2 Algorithm**

1. Reclassify classes according to the RGB Closest Neighbor Class Matrix.

1. Leave standalone classes as it is.

2. Delete all the classes which are not available.

3. Merge all the similar classes together to form a new class.

2. Count the number of classes left after class reclassification.

1. Store the newly derived classes into a matrix called RGB New Class Matrix. RGB New Class Matrix is a two dimensional array with two columns. The number of rows is the same as the number of classes counted in step two.
2. Recompute the Class Mean for each new class in the RGB New Class Matrix. Similar classes may be merged together, so the corresponding Class Means need to be recomputed as well. The following example shows how the new class mean is recalculated from two similar classes.

*classMeanA = 50, numPixelA = 4,*

*classMeanB = 40, numPixleB = 6,*

newClassMean

= (*classMeanA \* numPixelA + classMeanB \* numPixelB) / (numPixelA + numPixelB)*

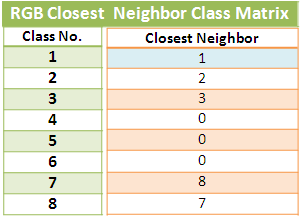
*= (50 \* 4 + 40 \* 6) / (4 + 6) = 44*

There are two classes A and B. The number of pixels in class A is *4*, and its Class Mean is *50*. The number of pixels in class B is *6*, and its Class Mean is *40*. The new Class Mean computed for the new class is *44* which is an arithmetic mean.

1. Stored the newly computed class means in a matrix called RGB New Class Mean Matrix for further analysis.

**4.7.3 Implementation**

1. Reclassify classes according to the RGB Closest Neighbor Class Matrix. The following is an instance of the RGB Closest Neighbor Class Matrix.



*Fig.10. RGB Closest Neighbor Class Matrix*

*For classNo = 1 to 8*

*/\**

*1. Leave standalone classes as it is.*

*2. Delete all the classes which are not available.*

*3. Merge similar classes together to form a new class.*

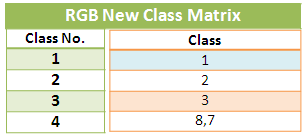
*\*/*

*...................................................*

*End*

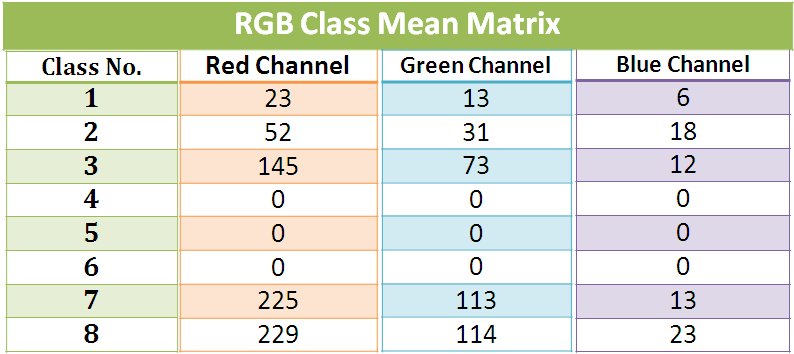
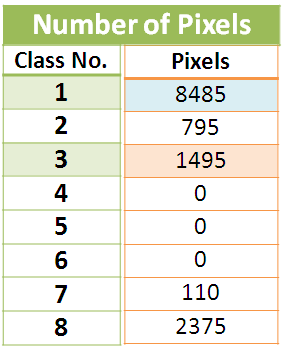
2. Count the number of new classes. In this case, the number of new classes is four. Class one, two, and three are standalone classes and they should stay as they are. Class four, five, and six are not available and they should be deleted. Class seven and eight are Closest Neighbor Classes and they should be merged to form a new class.

1. Store the newly derived classes into a matrix called RGB New Class Matrix. RGB New Class Matrix is a two dimensional array with four rows and two columns.



*Fig.10. RGB New Class Matrix*

1. Recalculate the Class Mean for each new class in the RGB New Class Matrix.

*Fig.10. New Class Mean*

*For classNo = 1 to 8*

*/\**

1. *For a standalone class, leave the mean as it is.*
2. *For Closest Neighbor Classes, calculate the arithmetic mean for them.*

*\*/*

*...................................................*

*End*

The following computation demonstrates how the arithmetic means between class seven and eight are calculated on three channels separately.

1. **On the Red Channel**

*classMean7 = 225, classMean8 = 229,*

*numOfPixels7 = 110, numbOfPixels8 = 2375,*

*arithmeticMean = (225 \* 110 + 229 \* 2375) / (110 + 2375)*

*= 228.82*

1. **On the Green Channel**

*classMean7 = 113, classMean8 = 114,*

*numOfPixels7 = 110, numbOfPixels8 = 2375,*

*arithmeticMean = (113 \* 110 + 114 \* 2375) / (110 + 2375)*

*= 113.95*

1. **On the Blue Channel**

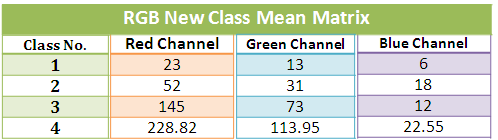
*classMean7 = 13, classMean8 = 23,*

*numOfPixels7 = 110, numbOfPixels8 = 2375,*

*arithmeticMean = (13 \* 110 + 23 \* 2375) / (110 + 2375)*

*= 22.55*

1. Stored the newly computed Class Means in a matrix called RGB New Class Mean Matrix.

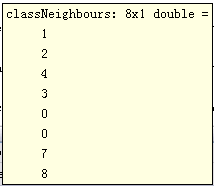


*Fig.10. RGB New Class Mean Matrix*

The new Class Means for class one, two, and three stay the same. Class four is a newly derived class. Its mean was recalculated based on the previous class seven and eight. Its new Class Means are *228.82* on the red channel, *113.95* on the green channel, and *22.55* on the blue channel.

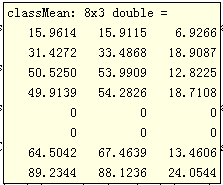
**4.7.4 Experiment and Analysis**

The following diagram shows the RGB Closest Neighbor Matrix from a real application.



*Fig.10. RGB Closest Neighbor Matrix*

It is clearly to see that class one, two, seven and eight are standalone classes. Class three and four are Closest Neighbor Classes. Class five and six are not available. The following diagram shows the RGB Class Mean Matrix of the above classes.



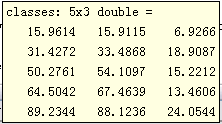
*Fig.10. RGB Class Mean Matrix*

The Class Means between class three and class four are very close to each other compared with other Class Means. Class three and four should be merged together to form a new distinct class. Recalculate the Class Mean for newly derived class.

newClassMean

= (*classMeanA \* numPixelA + classMeanB \* numPixelB) / (numPixelA + numPixelB)*

The following diagram shows the RGB New Class Mean Matrix after recompilation.

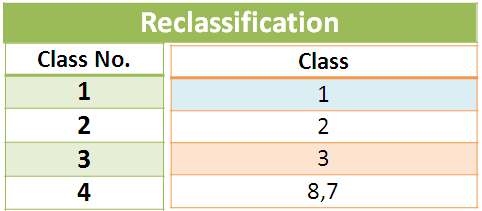
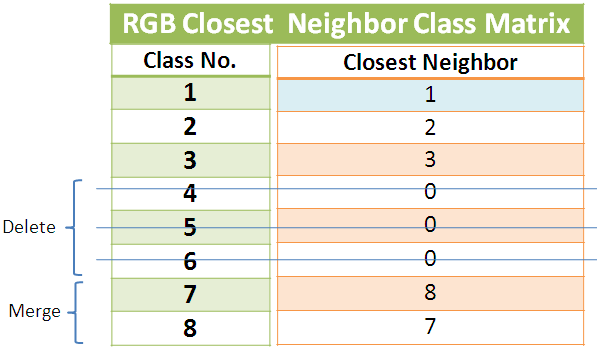


*Fig.10. RGB New Class Mean Matrix*

There are five classes left after class reclassification. Previous class five and six are deleted, and class three and four are merged together.

**4.7.5 Summary**

The following is an example of class reclassification.



*Fig.10. Class Reclassification*

There are three steps involved in class reclassification.

1. Leave standalone classes as they are.

2. Delete all the classes which are not available.

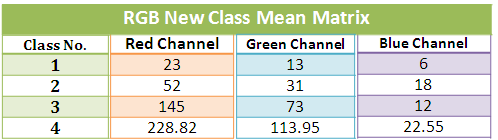
3. Merge the Closest Neighbor Classes together to form a new distinct class.

The Class Mean needs to be recalculated for any newly derived class. The formula is shown as follow:

newClassMean

= (*classMeanA \* numPixelA + classMeanB \* numPixelB) / (numPixelA + numPixelB)*

The newly computed Class Means are stored in a matrix called RGB New Class Mean Matrix. The following is an example of the RGB New Class Mean Matrix.

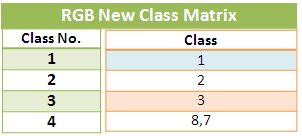


*Fig.10. RGB New Class Mean Matrix*

* 1. **Pixel Reclassification**

**4.8.1 Overview**

Since RGB Color Class has been reclassified, pixels in the RGB image no longer belong to one of the eight original RGB Color Classes. RGB New Class Matrix contains the information about newly derived classes. The property of each pixel in the RGB image should be reclassified accordingly based on certain rules. For instance, *P(1,2)* is a member of class eight before class reclassification, however, there is only four classes left after class reclassification. The following diagram shows the newly derived classes.



*Fig.10. RGB New Class Matrix*

It is clearly to see that *P(1,2)* no longer belongs to class eight, because class eight is not available in the RGB New Class Matrix. Find which new class each pixelbelongs to is the assignment in this algorithm.

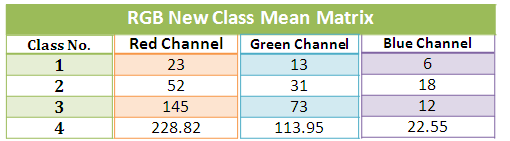
The intensity values of *P(1,2)* on three channels are:

*R(1,2) = 127 Red Channel,*

*G(1,2) = 55 Green Channel,*

*B(1,2) = 16 Blue Channel*

Find that which class *P(1,2)* belongs to. In another word, the intensity value of *P(1,2)* is more similar or close to which new Class Mean.



*Fig.10. RGB New Class Matrix*

In statistics, the word similar or close can be interpreted as variance. A low variance indicates that the data points tend to be very close to the testing class, while high variance indicates that the data points are far away from the testing class. The following example shows how the variance of *P(1,2)* to each class on the red channel is computed.

1. Testing Class One:

*Var[1] = R(1,2)- redChannelMean(1,1) = 127 – 23 = 104*

1. Testing Class Two:

*Var[2] = R(1,2)- redChannelMean(2,1) = 127 – 52 = 75*

1. Testing Class Three:

*Var[3] = R(1,2)- redChannelMean(3,1) = 145 – 127 = 18*

1. Testing Class Four:

*Var[4] = R(1,2)- redChannelMean(4,1) = 228.82 – 127 =101.82*

It is clearly to see that *P(1,2)* on the red channel is more close to class three. *Var[3]* is the smallest variance compared with *Var[1]*, *Var[2]*, and *Var[4]*.

The value between *P(1,2)* and Class Mean could be negative, e.g.,

*Var[1] = R(1,2)- redChannelMean(1,1) = 23 – 127 = -104*

The sign is not important in this case, because the variance will be squared in the next step. So it does not matter if the variance is negative or positive in this step. The following example shows how the variance of *P(1,2)* to each class on the green channels is computed.

1. Testing Class One:

*Var[1] = G(1,2)- greenChannelMean(1,2) = 55 – 13 = 42*

1. Testing Class Two:

*Var[2] = G(1,2)- greenChannelMean(2,2) = 55 – 21 = 34*

1. Testing Class Three:

*Var[3] = G(1,2)- greenChannelMean(3,2) = 73 – 55 = 18*

1. Testing Class Four:

*Var[4] = G(1,2)- greenChannelMean(4,4)*

*=113.95 – 55 = 58.95*

It is clearly to see that *P(1,2)* on the green channel is more close to class three. *Var[3]* is the smallest variance compared with *Var[1]*, *Var[2]*, and *Var[4]*. The following example shows how the variance of *P(1,2)* to each class on the blue channels is computed.

1. Testing Class One:

*Var[1] = B(1,2)- blueChannelMean(1,3) = 16 – 6 = 10*

1. Testing Class Two:

*Var[2] = B(1,2)- blueChannelMean(2,3) = 18 – 16 = 2*

1. Testing Class Three:

*Var[3] = B(1,2)- blueChannelMean(3,3) = 16 – 12 = 4*

1. Testing Class Four:

*Var[4] = B(1,2)- blueChannelMean(4,3) = 22.55 – 16 = 6.55*

It is clearly to see that *P(1,2)* on the blue channel is more close to class two. *Var[2]* is the smallest variance compared with *Var[1]*, *Var[3]*, and *Var[4]*. However, in both red and green channels *Var[3]* is the smallest variance. Which class *P(1,2)* belongs to cannot be decided based on one isolated channel. Compute the average variance among three channels is the solution. The assumption is that the variance in the red channel is more important than the one in the green channel, and the variance in the green channel is more important than the one in the blue channel. This assumption is based on the texture of the orange. The color of an orange is more likely to be red, maybe a little bit green, and almost no blue. So the quadratic mean is the selected method of measurement.

1. Testing Class One:

*Var[1] = sqrt((104.^2 + 42.^2 + 10.^2) / 3) = 65.01*

1. Testing Class Two:

*Var[2] = sqrt((75.^2 + 34.^2 + 2.^2) / 3) = 47.55*

1. Testing Class Three:

*Var[3] = sqrt((18.^2 + 18.^2 + 4.^2) / 3) = 14.87*

1. Testing Class Four:

*Var[4] = sqrt((101.82.^2 + 58.95.^2 + 6.55.^2) / 3)*

*=68.03*

It is clearly to see that *P(1,2)* is more close to class three, so *P(1,2)* is classified as a member of class three. Its intensity value is set to be the same as the Class Mean on three channels separately, e.g.,

*R(1,2) = 145 Red Channel,*

*G(1,2) = 73 Green Channel,*

*B(1,2) = 12 Blue Channel*

All the pixels in the RGB image need to be reclassified.

1. Find which class the pixel belongs to.
2. Set the Class Mean as its intensity value on three channels separately.

**4.8.2 Algorithm**

1. Step through each pixel in the RGB image.

1. Compute the average variance to each RGB new color class separately. The formula is defined as follow:

*Var = sqrt(((Rij – classMeanR).^2 +*

*(Gij – classMeanG).^2 + (Bij – classMeanB).^2)/ 3)*

*Rij,* *Gij,* and *Bij* represent the intensity values of *Pij* on the red, green, and blue channels separately.

*classsMeanR, classMeanG, and classMeanB* represent the RGB new class means on three channels separetely. *i* and *j* are rows and columns of the RGB image.

2. Find the smallest variance. The smallest variance means that *Pij* is more close to that class, so *Pij* should be classified as a member of that class.

1. Assign values to each pixel according to its Class Mean. For instance, if *P(1,2)* is a member of class two, then the value of *P(1,2)* on the red channel will be the Class Mean of class two on the red channel, the value of *P(1,2)* on the green channel will be the Class Mean of class two on the green channel, and so on.
   * 1. **Implementation**
2. Step through each pixel in the RGB image.

*For i = 1 to rows*

*For j = 1 to cols*

*For classNo = 1 to classCount*

*/\**

*Compute the average variance to each RGB new color class. For instance:*

*Var = sqrt(((R(i,j)– classMean(classNo,1)).^2 +*

*(G(i,j)– classMean(classNo,2)).^2 +*

*(B(i,j)– classMean(classNo,3)).^2)/ 3)*

*\*/*

*...........................................*

*/\**

*Find the smallest variance.*

*\*/*

*...........................................*

*End*

*End*

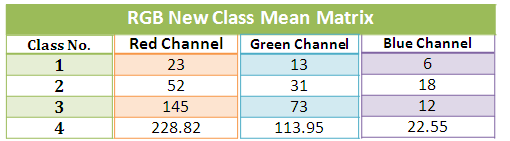
*End*

The following computation demonstrates how the pixel reclassification works. The intensity values of *P(4,2)* on three channels are:

*R(4,2) = 50 Red Channel,*

*G(4,2) = 20 Green Channel,*

*B(4,2) = 7 Blue Channel*



*Fig.10. RGB New Class Mean Matrix*

*Var[1] = sqrt(((23-50).^2 + (13-20).^2 + (6-7).^2) /3)*

*= 16.11*

*Var[2] = sqrt(((52-50).^2 + (31-20).^2 + (18-7).^2) /3)*

*= 9.05*

*Var[3] = sqrt(((145-50).^2 + (73-20).^2 + (12-7).^2) /3)*

*= 62.87*

*Var[4] = sqrt(((228.82-50).^2 + (113.95-20).^2 +*

*(22.55-7).^2) /3) = 116.96*

The smallest variance is *Var[2]*, so *P(4,2)* is classified as a member of class two.

1. Assign values to each pixel according to its class mean. *P(4,2)* is classified as a member of class two in step one. The intensity values of *P(4,2)* on three channels are assigned as follow:

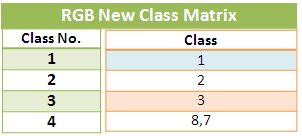
*R(4,2) = 52 Red Channel,*

*G(4,2) = 31 Green Channel,*

*B(4,2) = 18 Blue Channel*

* + 1. **Summary**

Pixels in the RGB image no longer belong to one of the eight original RGB Color Classes after class reclassification. Each pixel in the RGB image should be reclassified to one of the newly derived classes.



*Fig.10. RGB New Class Matrix*

The variance is selected as a reclassification tool. A low variance indicates that the data points tend to be very close to the testing class, while high variance indicates that the data points are far away from the testing class. The formula is defined as follow:

*Var = sqrt(((Rij – classMeanR).^2 +*

*(Gij – classMeanG).^2 + (Bij – classMeanB).^2)/ 3)*

The assumption is that the variance in the red channel is more important than the one in the green channel, and the variance in the green channel is more important than the one in the blue channel. This assumption is based on the texture of the orange.

The intensity value of each reclassified pixel is set to be the same as its corresponding Class Mean on three channels separately.

**Chapter 5**

**Parallel Image Processing**

* 1. **Overview**

An image is a two dimensional array of picture elements. Image processing is any form of signal processing that converts the original image to another image for certain purpose.

******

*Fig.10. Edge Detection Using the Laplacian*

Nowadays large amount of data are required for image processing, such as medical imaging requires the processing of images in the megapixel range. Sometimes computations can’t be completed within a desirable time frame(Grand Challenge Problem). Memory access time slows down the whole process especially with a large amount of data input. Traditional linear approach can’t solve this problem effectively, so a new expandable architecture is required to handle this challenge.

The basic idea of Parallel Image Processing(PIP) is to use multi-processors to perform a single task or multiple tasks simultaneously. Large problems can almost always be divided into smaller ones. The maximum speedup is n with n processors(linear speedup). But in practice, it can’t be achieved according to Amdah’s Law. The serial section is constant such as reading data from a CD. Because the large amount of memory is available, memory access time is much faster than traditional computing system.

Parallel Image Processing has been a topic of interest for many years, but there is no universal supported parallel machine model. Covert existing code into a parallel format is not easy.

* 1. **Algorithm**

There are three main basic steps in the parallel image processing algorithm:

1. Split the original image into even parts according to the number of processors used.



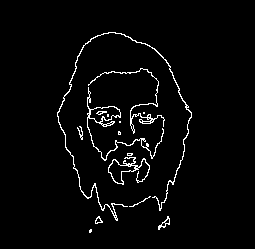
*Fig.10. Split Original Image into Two Even Parts*

1. Each of the processors operates on their own part of image to produce a multiple of partial image processing results.



*Fig.10. Partial Results*

1. Merge the partial results to produce a final output.



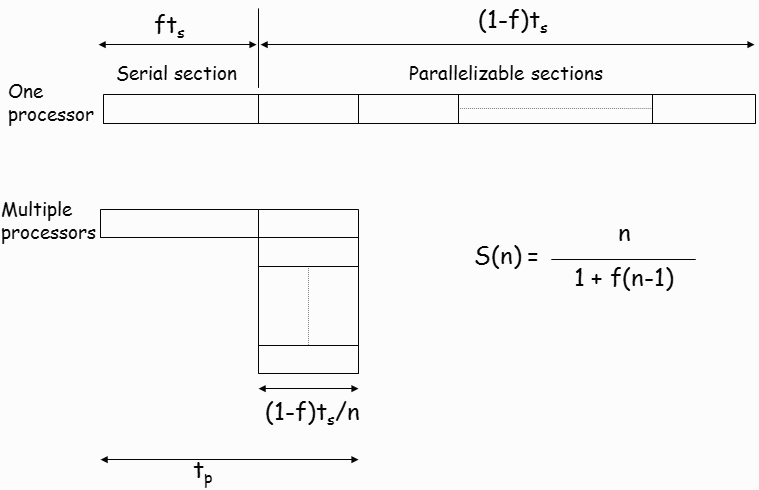
*Fig.10. Output Image*

Parallel system(stcluster) in Massey University:

1. Maximum 8 processors.
2. Maximum memory size 8MB.
3. Using Message Passing Interface(MPI) sending and receiving messages between processors.

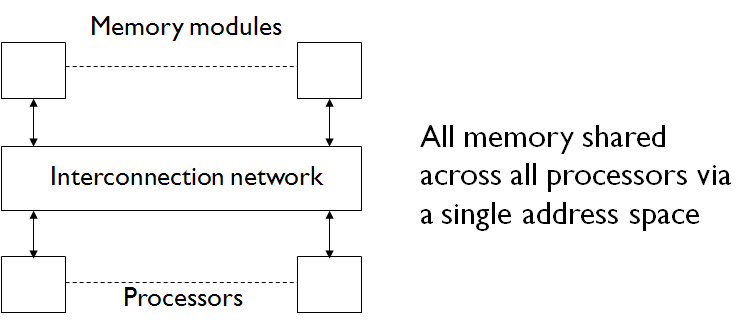
*Fig.10. Master/Slaves Approach*

Parallel allows image processing to be conducted at an increased speed. The maximum speedup is n with n processors(linear speedup). But in practice, it can’t be achieved according to Amdah’s Law. The serial section is constant such as reading data from a CD.

******

*Fig.10. Amdah’s Law*

Communication between processors is very time consuming. It depends on the input image size and the number of processors used. One way to solve this problem is to increase the memory access speed, such as shared memory multiprocessor model.

******

*Fig.10. Shared Memory Multiprocessor Model*

* 1. **Experiment and Analysis**

Edge detection is based on the Lalacian method. The edges in an image correspond to the zero crossings of the Laplacian.

******

*Fig.10. Edge Detection*

This experiment is tested on the stcluster server in Massey University. It doesn’t achieve the desired result, but some useful experience can be shared with all students.

Process:

1. Split the original image into even parts.
2. Each processor loads its own part of image and produces a partial edge detection result***.***
3. Merge all the partial edge detection result, and produce the final output.

**Interesting topics:**

1. Image Size

Image size is not the file size. File size is something like 4Kb and 8MB. We only want to measure the image size such as 456\*245(width and height). The maximum input image size is about 6000\*4000. Because the maximum memory size is 8MB. The larger input image size requires more memory space.

1. Memory Allocation

Memory can’t automatically allocate space for matrix in stcluster, such as int img[10][10]. It works in c code, but doesn’t work in stcluster. You have to manually allocate memory space for matrix.

*int \*\* im=new (int \*, im1->y);*

*im[0]=new(int,im1->x\*im1->y);*

*for( a=1;a < im1->y;a++)*

*im[a]=im[a-1]+im1->x;*

1. Message passing uses the standard send and receive routines. Such as

*MPI\_Send(&im[0][0],(im1->x)\*(im1->y),MPI\_INT,0,0,MPI\_COMM\_WORLD)*

We treat the partial image as a whole block of data. The data size is *(im1->x)\*(im1->y)*, and the starting position is *im[0][0]*.

1. Memory Access

At the beginning, I also measure the time of loading an image on each processor. I found that the time of loading the same image in the second time is much shorter than the first time. That is because the memory access time is much shorter in the second time. So I wouldn’t measure the time of loading and saving an image in this experiment. I only measure the computational and communication time in this experiment.

**Performance:**

For small input image size, there is no point to use parallel system. One processor is fast enough to handle the problem. The more processors you use the longer time is going to be taken. The communication between processors is very time-consuming.

For large input image size, there is an upper bound on the number of processors used. Below the upper bound, the more processors you use, the better result will be achieved. The computational time takes much longer than the communication time. The computational time dominates the whole execution time. In this case, we are happy to say that large problems are divided into smaller ones. Above the upper bound, the communication time dominates the whole execution time. Communication between processors takes much longer than the computational time. At this stage, there is no point to increase the number of processors any more.

**Experiment:**

Time is measured in seconds.

Total Time = Communication Time + Computational Time

Input Image Size 2250\*2222

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Processors | 1 | 2 | 4 | 8 |
| Total Time | 0.268 | 0.246 | 0.307 | 0.336 |
| Communication Time | 0 | 0.113 | 0.242 | 0.303 |
| Computational Time | 0.268 | 0.132 | 0.065 | 0.032 |

Input Image Size 3549\*2463

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Processors | 1 | 2 | 4 | 8 |
| Total Time | 0.431 | 0.499 | 0.521 | 0.582 |
| Communication Time | 0 | 0.243 | 0.411 | 0.526 |
| Computational Time | 0.431 | 0.256 | 0.111 | 0.055 |

Input Image Size 4500\*3000

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Processors | 1 | 2 | 4 | 8 |
| napoleonTotal Time | 0.694 | 0.658 | 0.845 | 0.927 |
| Communication Time | 0 | 0.304 | 0.667 | 0.838 |
| Computational Time | 0.694 | 0.354 | 0.178 | 0.088 |

Input Image Size 6000\*4000

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Processors | 1 | 2 | 4 | 8 |
| Total Time | 2.794 | 2.919 | 3.282 | 3.743 |
| Communication Time | 0 | 1.491 | 2.565 | 3.877 |
| Computational Time | 2.794 | 1.428 | 0.716 | 0.355 |

Each input image is tested 3 times on the same number of processors. All the data used above are average values.

The computational time decreases when the number of processors increases. It is almost linear speed up. But the communication time increases a lot when the number of processors increases. When we use 2 processors, computational and communication time are almost equal. When we use 4 processors, the communication time is much bigger than the computational time. When we use 8 processors, the communication time is almost equal the total execution time.

Communication between processors is very time-consuming. That is the main reason why I can’t achieve the desired result.

* 1. **Summary**

Parallel Image Processing is very efficient for large input image size. Large problems can almost always be divided into smaller ones.

The processing speed can be increased by parallel processing, thus reducing the overall processing time. There is an upper bound on the number of processors used. Using the right number of processors for different tasks can be a challenge.

Communication between processors is very time-consuming. How to reduce the communication time will become a very interesting topic.