**CONTENT BASED SIMILAR IMAGE RETRIEVAL SYSTEM (NATIONAL FLAGS AND VEHICLE LOGOS)**

BY

GOH KIEN CHUN

12ACB01457

PRACTICAL SESSION: T4 MONDAY 8 – 11 A.M.

TEAM 25

A MINI PROJECT REPORT

SUBMITTED TO

PROF. DR. ZEN CHEN

MAY 2017

**Table of Contents**

[CHAPTER 1 INTRODUCTION 1](#_Toc492894792)

[1.1 Problem Statement 1](#_Toc492894793)

[1.2 Motivation 1](#_Toc492894794)

[1.3 Project Background 2](#_Toc492894795)

[1.4 Project Scope 2](#_Toc492894796)

[1.5 Project Objective 3](#_Toc492894797)

[CHAPTER 2 LITERATURE REVIEW 4](#_Toc492894798)

[2.1 Content Based Image Retrieval System Based on Semantic Information Using Colour, Texture and Shape Features 4](#_Toc492894799)

[2.1.1 Introduction 4](#_Toc492894800)

[2.1.2 Related Work 4](#_Toc492894801)

[2.1.3 Proposed Methodology 6](#_Toc492894802)

[2.2 Trademark Image Retrieval by Integrating Shape with Texture Feature 10](#_Toc492894803)

[2.2.1 Introduction 10](#_Toc492894804)

[2.2.2 Related Work 11](#_Toc492894805)

[2.2.3 Proposed Methodology 12](#_Toc492894806)

[2.3 Trademark Image Retrieval Based on Shape and Key Local Colour Features 14](#_Toc492894807)

[2.3.1 Introduction 14](#_Toc492894808)

[2.3.2 Proposed Methodology 14](#_Toc492894809)

[2.4 Content-Based Retrieval of Logo and Trademarks in Unconstrained Colour Image Databases Using Colour Edge Gradient Co-occurrence Histograms 15](#_Toc492894810)

[2.4.1 Introduction 15](#_Toc492894811)

[2.4.2 Related Work 15](#_Toc492894812)

[2.4.3 Proposed Methodology 17](#_Toc492894813)

[2.5 Comparisons of Four Proposed Methods from Literature Surveys 19](#_Toc492894814)

[CHAPTER 3 SYSTEM DESIGN 20](#_Toc492894815)

[3.1 Colour Descriptor 20](#_Toc492894816)

[3.1.1 Colour Quantization 21](#_Toc492894817)

[3.1.2 Dominant Colours Formation 22](#_Toc492894818)

[3.1.3 Symmetry Picture 25](#_Toc492894819)

[3.2 Shape Descriptor 26](#_Toc492894820)

[3.2.1 Foreground Retrieval 27](#_Toc492894821)

[3.2.2 Central Normalized Moment 28](#_Toc492894822)

[3.3 Combined Descriptor 29](#_Toc492894823)

[3.4 Datasets 31](#_Toc492894824)

[3.5 Similarity Score of Computation 33](#_Toc492894825)

[3.5.1 Similarity Score Calculation of Colour Descriptor 33](#_Toc492894826)

[3.5.2 Similarity Score Calculation of Shape Descriptor 40](#_Toc492894827)

[3.5.3 Similarity Score of Calculation of Shape and Colour Descriptor 42](#_Toc492894828)

[CHAPTER 4 SYSTEM IMPLEMENTATION, TESTING, AND VERIFICATION 45](#_Toc492894829)

[4.1 Hardware and Software Environments Used in Project Implementation 45](#_Toc492894830)

[4.1.1 Hardware Specification 45](#_Toc492894831)

[4.1.2 Software Specification 45](#_Toc492894832)

[4.2 System Code Implementation 45](#_Toc492894833)

[4.3 System Code Testing and Verification 49](#_Toc492894834)

[4.3.1 Colour Brightness Image 49](#_Toc492894835)

[4.3.2 Wiggled Image 50](#_Toc492894836)

[4.3.3 Shadow Image 51](#_Toc492894837)

[4.3.4 Rotation Image 52](#_Toc492894838)

[4.3.5 Different Scaling Image 53](#_Toc492894839)

[4.3.6 3D Image 54](#_Toc492894840)

[CHAPTER 5 SYSTEM ANALYSIS AND PERFORMANCE EVALUATION 55](#_Toc492894841)

[5.1 Colour Descriptor Analysis 56](#_Toc492894842)

[5.2 Shape Descriptor Analysis 70](#_Toc492894843)

[5.3 Combined Descriptor Analysis 78](#_Toc492894844)

[CHAPTER 6 CONCLUSIONS 81](#_Toc492894845)

[6.1 Strengths and Weaknesses 81](#_Toc492894846)

[6.2 Preliminary Work 81](#_Toc492894847)

[Bibliography 83](#_Toc492894848)

[Appendix 84](#_Toc492894849)

# CHAPTER 1 INTRODUCTION

## 1.1 Problem Statement

1. To maximize the potential of RGB colour space so that we can use RGB colour space to retrieve lightning image, shadowed image, or wiggled image.

Nowadays many researchers are developing content based image retrieval system mainly based on HSV or La\*b\* colour space, because these 2 colour spaces will be efficient and easy to use during image computation. Therefore, the main concern of this paper is to ensure that the algorithms and methodologies used with RGB colour space can outperform any other researches that used RGB colour space, and achieve the same performance as HSV or La\*b\* colour space can do.

1. To build a colour descriptor that can retrieve similar colours. Besides, to build a shape descriptor that can retrieve similar contours. Finally, we will combine both colour and shape descriptors, so that the new combined descriptor will be more powerful.

Many research papers only study one descriptor, which is either colour, shape or texture. Whereas, in our research project, we will investigate 2 descriptors and combine the descriptors at final stage to get a more precise result.

## 1.2 Motivation

Object classification always be a painful job for human if there is a huge database. Therefore develop an efficient content based image retrieval system will be useful to solve this situation. Other than that, a content based image retrieval system will be helpful to distinguish similar trademark if the third party has apply intellectual property of it. This can assure that every trademark design can have a strong protection that no one can violate and infringe it.

## 1.3 Project Background

In many areas of commerce, government, academia, and hospitals, large collections of digital images are being created. Many of these collections are the product of digitizing existing collections of analogue photographs, diagrams, drawings, paintings, and prints. Thus, an image retrieval system is needed to search, retrieve, and browse similar query images from digital images databases.

However, image retrievals using text-based keywords are tedious as well as churn out results that may be irrelevant. Therefore, content based image retrieval is proposed. Content based image retrieval is a technique used for extracting similar images from an image database using their content or features. The query that may be presented to such a system may be an image that the user has, a rough sketch, a colour or texture layout or a short verbal description.

Characteristics of content based image retrieval include:

1. Image retrieval by image content;
2. Visually similar images to query image;
3. No keywords
4. Low level features like colour, texture and shape are used.

The main challenge involved in image retrieval by content is the need to bridge the semantic gap between low level features and high level concepts. The need to manage huge images and to locate target images in response to user queries poses a significant problem for research in Digital Image Processing. Colour, texture and shape information are the primitive image descriptors in content based image retrieval systems.

## 1.4 Project Scope

In this project, the system will be built on C++ and OpenCV platform. The image datasets will be national flags and vehicle logos. The main methodologies used to retrieve similar images are colour quantization as well as foreground retrieval.

## 1.5 Project Objective

1. To improve the retrieval rate of similar images using RGB colour space.

The first objective of this paper is to improve the retrieval rate using RGB colour space that other researches cannot achieve.

1. To compare the performance of corresponding colour pixel percentage and corresponding rank dominant colour for colour descriptor.

The second objective of this paper is to compare the performance of using corresponding colour pixel percentage and corresponding rank dominant colour to calculate similarity difference. Appropriate statistical analysis and testing will be conducted to multiple query images to find out the algorithm with better performance.

1. To compare the performance of central normalized moments and Hu moments for shape descriptor.

The third objective of this paper is to compare the performance of using central normalized moments and Hu moments to calculate similarity difference. Appropriate statistical analysis and testing will be conducted to multiple query images to find out the algorithm with better performance.

1. To achieve 85% recall rate for combined descriptor.

Lastly, the forth objective of this paper is to achieve at least 85% recall rate of a give query image that can retrieve similar images from the database.

# CHAPTER 2 LITERATURE REVIEW

## 2.1 Content Based Image Retrieval System Based on Semantic Information Using Colour, Texture and Shape Features

### 2.1.1 Introduction

Due to the increasingly in the size of multimedia databases like text, audio, video, and image as well as the vast differences between the human perception and a computer vision known as Semantic Gap, the content based image retrieval has become a challenging task in the event of accessing multimedia databases. From the perception of this paper, the existing methods pay the attention towards the accuracy of the retrieval to extract the objects in an image and describe the interconnection between objects. However, the major shortcoming of all the existing methods is incapable in accessing a huge amount of queries. Hence, the main objective of this paper is to propose a method that can carry out a more accurate result from a huge database. In this paper, Corel image database, Li image database and Caltech-101 image database will be used for CBIR system testing.

### 2.1.2 Related Work

This paper has stated out seven existing methods which are SIFT, Computation Visual Attention Model, Gabor Wavelet Feature, Dynamic Colour Distribution Entropy of Neighbourhoods (D\_CDEN) and Grey Level Co-occurrence Matrix (GLCM), Local Binary Pattern and Gabor Transform Feature, Modified Colour Motif Co-occurrence Matrix (MCMCM), and Wavelet Based Colour Histogram Image Retrieval (WBCHIR).

SIFT:

This method is used to extract the local features in images. In the event of CBIR system related with SIFT, there are two main problems which are time complexity and memory usage. In order to handle these problems, k-means clustering and two kinds of dimensionality are applied on feature component extracted by SIFT algorithm to make an image retrieval problem more efficient.

Computation Visual Attention Model:

This method extracts three features, namely novel visual cue, Grey-level co-occurrence matrices, and saliency structure histogram. This model is an improved version of SIFT, but still does not achieve good performance.

Gabor Wavelet Feature:

In the prior stage, three CBIR algorithms were carried out to compare which was the most effective method. The three algorithms are RGB Colour Histogram, Tamura Texture, and Gabor Wavelet Feature, and at the end Gabor Wavelet Feature is more efficient when compared to the other two methods.

D\_CDEN and GLCM:

The colour and texture features extracted from images are done by D\_CDEN and GLCM respectively. This method has used a k-means clustering algorithm to compute intra cluster similarity of images, but not suitable for computing inter cluster similarity.

Local Binary Pattern and Gabor Transform Feature:

This method has achieved the most effective result in terms of performance evaluation measures when made in comparison with other six methods.

MCMCM:

This method extracts the inter correlation between RGB colour information. Besides that, difference between pixels of scan pattern is added to measure the variation among the pixels within colour motifs of a scan pattern. As a result, it has a better retrieval rate among the respective databases.

WBCHIR:

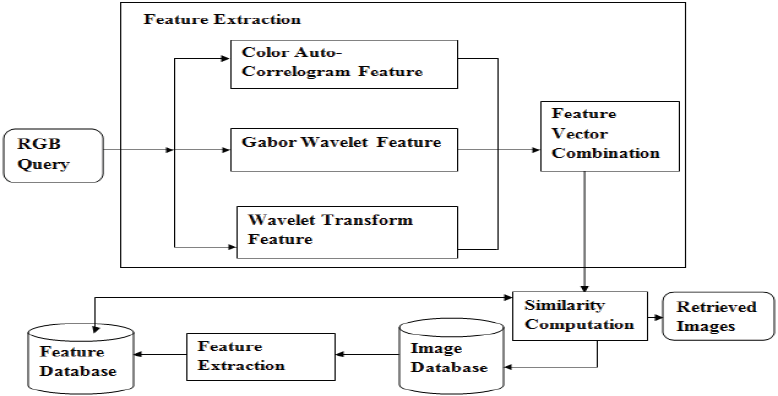
This method extracts the features by integrating colour and texture information.

|  |  |  |
| --- | --- | --- |
| **Method** | **Advantage** | **Disadvantage** |
| SIFT | K-means and two kinds of dimensionality are applied. | Inefficiency in time complexity and memory usage management. |
| Computation Visual Attention Model | Improved version of SIFT. | The performance evaluation is bad. |
| Gabor Wavelet Feature | The performance evaluation is good if compared to RGB Colour Histogram and Tamura Texture. | The three algorithms cannot be integrated. |
| D\_CDEN and GLCM | K-means is applied to compute intra cluster similarity of images. | Unable to compute inter cluster similarity of images. |
| Local Binary Pattern and Gabor Transform Feature | The performance evaluation is the best among all the other methods. | Two algorithms are involved and will be more complex. |
| MCMCM | The retrieval rate is the best among all the other methods. | Only suitable for RGB colour space. |
| WBCHIR | Integrates colour and texture information. | Lacking of shape information. |

*Table 2-1 Advantages and Disadvantages of the Related Works*

### 2.1.3 Proposed Methodology

In this paper, the authors have proposed a new algorithm to generate three image features, namely Colour Auto-Correlogram Feature, Gabor Wavelet Feature, and Wavelet Transform Feature. Colour Auto-Correlogram Feature is associated with colour information of an image which is derived from RGB colour space of an image. The Gabor Wavelet Feature has the texture information to extract the textural features associated with the image. Also, the Wavelet Transform Feature is linked with shape information in the extraction edges in an image. The databases used to test the efficiency of the proposed feature descriptor are Corel image database, Li image database, and Caltech-101 image database.

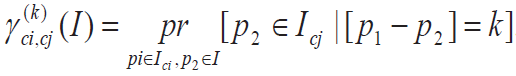


*Figure 2-1 Schematic Diagram of the Proposed Work*

In image processing, a feature can be classified into low-level, middle-level, and high-level feature. Low-level feature indicates colour and textural information of an image, whereas middle-level feature indicates shape information, in addition high-level feature indicates the semantic gap between objects. To bridge this gap, three features are extracted by integrating three different methods as shown in Figure 1. They are Colour Auto-Correlogram which extracts colour feature, Gabor Wavelet which extracts texture feature, and finally Wavelet Transform which extracts shape feature.

Colour Auto-Correlogram Feature:

This feature includes the spatial relationship between colours and it is used to describe the overall distribution of local spatial relationship between colours and it is simple to compute. The computation for the correlogram of an image is as the followed:



Where γ results in the probability that a pixel away from the given pixel of colour cj at distance k. I is the original RGB image of size m x n. The colour information of an image I are quantized into m colours and denoted as c1 to cm. For every pixel p(x, y) in an image, k represents the distance between pixel p1(x1, y1) and pixel p2(x2, y2) in an image.

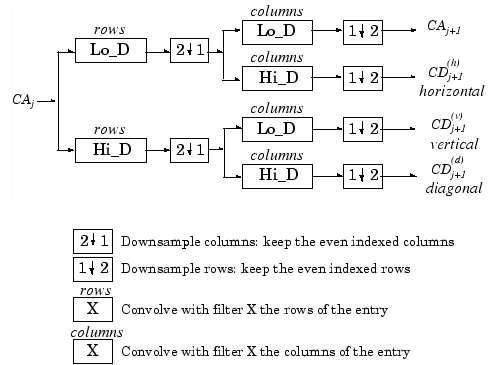
Gabor Wavelet Feature:

This feature is extracted by evaluating the mean information and standard deviation information from the filtered image. Let Gmn(x, y) be the Gabor fileter technique applied on the original image I(x, y) of size m x n by applying divergent orientation at varying scale. Then the magnitudes E(m, n) obtained from the Gabor filter can be formulated as followed:

Where m = 0 to M – 1 and n = 0 to N – 1. These magnitude are computed to measure the energy content at divergent orientation at varying scale applied on the image. Later then, mean and standard deviation are computed from the magnitude array to find the set of images with identical texture. The formula of mean and standard deviation are shown below respectively:

Wavelet Transform Feature:

Wavelet Transform can be used for extracting features based on the edge description of an object. Each level of the discrete signal is proceeded through more filters, where each level is computed by proceeding the previous level approximation coefficient through discrete time low-pass and high-pass filter. Discrete Wavelet Transform results in decomposition of approximation coefficients at level j into four components. The components are approximation made at level j + 1, and the other three components are orientation deployed at horizontal, vertical, and diagonal. The decomposition steps are shown in Figure 2.



*Figure 2-2 Decomposition Steps for an Image*

After the feature vector combination, there will be a similarity match between query image and the set of relevant images from the database. The similarity measure is computed by Manhattan distance metric. At the beginning, the three features extracted from the query image are compared with the features of relevant images from the database. After that, a set of relevant images will be returned and arranged in a required order based on their similarity score computed by Manhattan distance.

The performance of the proposed system is measured by precision and recall rate. Precision gives information about effectiveness of the proposed system, whereas recall gives information about the accuracy of the proposed system. The formulas of Precision and Recall are shown as followed respectively:

As a result, the proposed method in this paper achieves a better retrieval accuracy in comparison with the existing methods through information derived from Colour Auto-Correlogram Feature, Gabor Wavelet Feature, and Wavelet Transform Feature due to better precision rate derived from these three features. The average accuracy rate of this proposed method is 80%, whereas the existing methods obtain an accuracy rate of 71%, 72%, 53%, and 78% respectively.

## 2.2 Trademark Image Retrieval by Integrating Shape with Texture Feature

### 2.2.1 Introduction

Trademark is considered as a marketing tool and intellectual property as well as to differentiate one commercial organization from other. A trademark can be an image, symbol, logo, word, design, phrase or a fusion of these. The number of trademarks in emergent countries varies from thousands to hundreds of thousands, and it is growing gradually. Thus, it is necessary to guarantee that the new trademark is adequately distinct from the existing trademarks to avoid confusion and infringement.

Therefore, a precise and useful trademark retrieval system is required for these issues. Traditionally, trademark image retrieval techniques can be broadly classified into two categories which are description based and content based trademark image retrieval. In description based approach, the query is generated in text form to describe the query image. The database images of all classes along with their subclasses have annotations for describing the image. In content based approach, the query is generated in image form and its features are used as the content for describing the query image.

In this paper, the authors tend to design an accurate and effective trademark retrieval system that computes the similarity between two trademark images because it is challenging to validate the similarity between two trademark images as they can be similar in two manners either in geometrical manner or visual appearance.

The existing trademark retrieval systems were built mostly based on only one feature, which is either shape or texture. Thus, this paper propose a new retrieval method which combines both shape and texture features to retrieve similar trademark images. As a conclusion, results illustrate that the combined approach can bring good retrieval precision and speed simultaneously.

### 2.2.2 Related Work

Candes introduced a new multiscale directional transform that allow an optimal nonadoptive sparse representation of object with singularities. However, this approach has a severe problem because it ignores the geometric properties of object, as wavelet transform can only represent point singularities.

Therefore, the authors of this paper have introduced curvelet transform to overcome these problem. The two important features of curvelet make it possible to represent the sparse and handling image singularity better than other multiscale transform. According to the enhancement sparsity that brought out in this paper, curvelet transform yields the smaller asymptotic error with the same number of term if compared to wavelet transform.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Technology Used** | **Singularities Measurement Algorithm** | **Singularities Measurement** | **Asymptotic Error** |
| **Existing Method** | Wavelet Transform | Multiscale Directional Transform | Bad | Big |
| **Proposed Method** | Curvelet Transform | Geometrical Transform | Excellent | Small |

*Table 2-2 Comparisons between existing and proposed method*

### 2.2.3 Proposed Methodology



*Figure 2-3 Architecture of the Proposed System*

As shown in Figure 3 above, the feature database is obtained from trademark database by extracting features using shape and texture features. Later that, the similarity measure algorithm, Euclidean distance is applied over the feature database with the computed feature obtained through feature extraction process on query trademark image. Finally the retrieval result will be displayed. In this paper, Zernike Moment and Curvelet Transform are used as shape and texture feature extractions respectively.

Zernike Moment:

Zernike moments are rotation invariant, they are defined as a complex number moment based on a set or group of orthogonal polynomials, in addition they used as a complete orthogonal set in circle unit. There are few advantages can be carried out from this moment. Firstly, it has smaller information redundancy and better rotation variance. Secondly, it is easier to estimate and able to build arbitrary high order moments. Thirdly, it is robust against the noise.

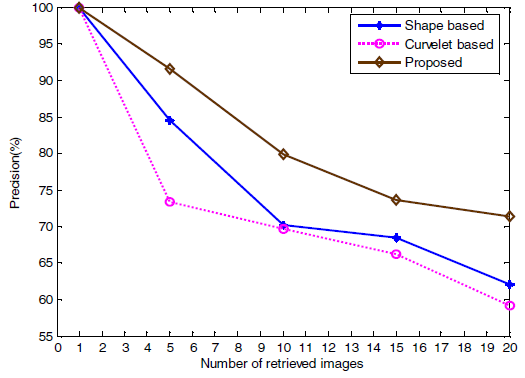
For estimation of Zernike moment of an image, first step is to identify the interested region of an image and then map that region to the origin of unit circle in polar coordinate system. In estimation, the pixels which are internal to the unit circle are considered and rest is excluded, eventually rotation invariance is achieved through this mapping.

Curvelet Transform:

These texture features are computed as the entropy of the sub-bands of the curvelet after obtaining the curvelet coefficients. Entropy is a statistical measure of randomness that can be used to characterize the texture of the input image. The advantage of this feature is it has good capability to describe the curvelet sub-bands.

After all the features are computed, Euclidean distance is used to measure and show the degree of similarity to decide most similar images from the database to the query image.

In order to validate the effectiveness and accuracy of the proposed trademark image retrieval system, an experiment is carried out with a trademark image database which contains 300 images, and they have variations in colour, shape, illumination, and size.



*Figure 2-4 Precision Rate of Shape, Curvelet, and Proposed Methods in Trademark Image Database*

Figure 4 illustrates that the proposed method is effective and accurate for the trademark image retrieval. Furthermore, the precision rate is better when less number of nearest images is obtained. For example, when 20 nearest images are obtained, the precision rate is 62%, 59%, and 71% for shape, curvelet, and proposed feature respectively. Whereas, when five nearest images are obtained, the precision rate is 84%, 73%, and 91%.

## 2.3 Trademark Image Retrieval Based on Shape and Key Local Colour Features

### 2.3.1 Introduction

Due to the increased numbers of registered trademarks which results more and more difficult in manual works on comparing the similarity degree of them, there is a need to establish an accurate and high-effective trademark retrieval system to strengthen the trademark management and protect the private trademark right.

The existing methods are more focus on features extraction and similarity measurement, while ignore concrete applications. Thus, this paper proposes a combined retrieval method for trademark images using shape and colour features to overcome this shortcoming. After the pre-processing of trademark images, Wavelet Modulus Maxima approach is used to extract shape features. This self-adaptive threshold method based on maximum between-cluster variance and simulated annealing algorithm has translation, scale and rotation invariance, and small amount of calculation. After that, retrieve them based on the HSV Colour Space accumulative histogram using key local colour features which improve the retrieval speed. Finally, get results by measuring the integrated similarity degree of shape and colour features. As a conclusion, the proposed algorithm can retrieve similar trademark images fast and efficiently as well as improve the recall and precision ratio.

### 2.3.2 Proposed Methodology

In this paper, the authors propose Wavelet Modulus Maxima approach for shape-based trademark image retrieval. Wavelet Modulus Maxima is a two dimensional smooth function whose integral is equal to one, and that converges to zero at infinity. These functions are defined at horizontal and vertical directions. As a result, the edge of the image which corresponds to the value of the catastrophe point can get by extracting from the wavelet transform. The results achieved from Wavelet Modulus Maxima is effective for shape-based only. However this approach does not include colour and texture information. Therefore, the images with similar contours but different contents were often retrieved.

To overcome this problem, Key Local Colour is used for colour-based trademark image retrieval. HSV colour space is used because HSV model has correlative advantages of naturalness, vision consistency, integrity, and compact, thus HSV model can match better with the visual characteristics than RGB model, because RGB colour space is a non-uniform linear space. Traditionally, people usually think of increasing quantized interval to adapt human visual characteristics, but it also leads to the erroneous inspection. Thus, the accumulative histogram is introduced to overcome the defects in concise quantitative.

As a conclusion, this method is proved to accord with human visual characteristics. Also, the retrieval speed is higher because of using key local colour features.

## 2.4 Content-Based Retrieval of Logo and Trademarks in Unconstrained Colour Image Databases Using Colour Edge Gradient Co-occurrence Histograms

### 2.4.1 Introduction

Even though much research has been put forth on logo and trademark detection and retrieval, most of the existing work already consists of logos and trademarks themselves in the database, and all of the images are of the same scale. The viewpoint of retrieval is most commonly encountered in copyright protection and trademark registration applications. However, the previous attempts do not address the case of unconstrained logo and trademark retrieval, and the acquisition of the images that populate the databases are under controllable conditions. Thus, the main objective in this paper is to propose a method to retrieve logos from unconstrained database. This is a very challenging idea because high amounts of deformation and large variations of colour depending on illumination condition are inevitable in unconstrained images.

### 2.4.2 Related Work

Relevant prior research in logo and trademark detection and retrieval include in the following, each concentrating on a particular feature or approach to solve the problem such as Shape, Neural Networks, Contour-Based, Colour Histograms, Co-occurrence Histogram (CCH), and Colour Edge Co-occurrence Histogram (CECH).

Shape:

Zernike and pseudo-Zernike moments were employed to build the feature set used for the retrieval of trademarks in trademark databases. Other than that, a trademark retrieval system was designed to segment all trademarks in the database into components, and derives all features from these component boundaries. Extracted features such as perimeter, aspect ratio, edges, area, and straightness.

Neural Networks:

Trademark retrieval can be treated as a pattern classification problem. Neural networks can behave as a model-free estimators and the network learned by training with input samples.

Contour-Based:

There have been approaches where user submits a hand drawn sketch using rectangles, polygons, ellipses, and B-spline curves, as opposed to an already prepared logo or trademark to submit to the system. Contour information of these hand drawn curves are used and features are extracted in this manner when they are handled as an input query. An example of algorithm for this approach is Fourier Descriptors.

Colour Histograms:

With regards to colour, much work has been put forth by considering colour as the primary feature. In this case, colour histograms were used for building indices and retrieving images based on these indices. The advantage of this method is it is simple and ease of implementation. In contrast, the colour histogram only captures colour content of an image, creating misclassifications for retrieved images.

CCH:

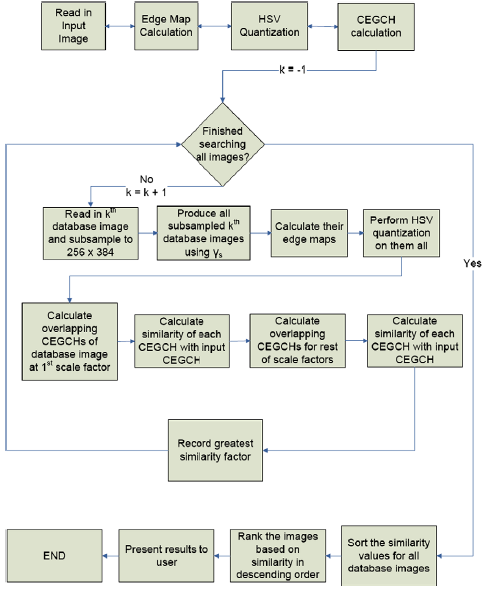
CCH is the modification to the colour histogram. It captures both the colour and spatial relationships between colour pixels. The advantage of this descriptor is it combines colour and texture together and makes it more attractive to use. However, the disadvantage is the detection of the same rigid object amidst cluttered scenes, and experiments were performed under controlled conditions which means the illumination, size, and orientation of the objects were all kept constant.

CECH:

CECH is a modification to CCH. It can eliminate the solid colour contribution problem to the CCH. CECH only captures the separation of pairs of colour pixels at different spatial distances when these pixels lie in edge neighbourhoods, alleviating the disproportionate energy contributions a single colour would have on CCH.

### 2.4.3 Proposed Methodology

This paper extends the CECH objection detection method for use in logo and trademark retrieval in unconstrained colour image databases. The authors introduce colour edge detection to the CECH using vector order statistics, which produces an edge map determining valid edge points in colour images with greater accuracy. Furthermore, Hue-Saturation-Value (HSV) colour quantization method is introduced because it is more suitable for image retrieval in unconstrained colour images. The algorithm of the whole system process is illustrated in Figure 5.



*Figure 2-5 Flowchart of Logo and Trademark Algorithm*

This proposed method generates a higher precision and recall for all logos. However, delineation will fail sometimes due to the logo in this image was smaller than the provided input or the overall lightning.

## 2.5 Comparisons of Four Proposed Methods from Literature Surveys

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Proposed Method 1** | **Proposed Method 2** | **Proposed Method 3** | **Proposed Method 4** |
| **Features Extraction** | Colour  Texture  Shape | Texture  Shape | Colour  Shape | Colour |
| **Technologies Used** | Colour auto-Correlogram feature  Gabor wavelet feature  Wavelet transform feature | Zernike moment  Curvelet transform | Wavelet modulus maxima  Key local colour | CECH  HSV colour quantization method |
| **Databases** | Corel image database  Li image database  Caltech-101 image database | Database that contains 300 trademark images | Database that contains only trademark images in black colour | 5400 unconstrained colour image database |
| **Advantages** | High accuracy and precision rate | High precision rate when less number of nearest images is obtained | High retrieval speed  Accord with human visual characteristics | Can retrieve images from unconstrained database  High precision and recall rate |
| **Disadvantages** | Feature descriptor is difficult to design because it combines three features into one | Lower precision rate when more number of nearest images is obtained especially when more than 20 images are obtained | Only suitable for black colour images | Delineation will fail sometimes due to several factors |

*Table 2-3 A comparison among the 4 survey papers*

# CHAPTER 3 SYSTEM DESIGN

In this project, colour and shape descriptors will be used to retrieve similar images. Colour descriptor consists of colour quantization, GBR colour space extraction, merge similar colours and form dominant colours, sort dominant colours. Whereas shape descriptors consist of background mask filling and calculate central normalized moment of the foreground region. Each of these elements will be further explained and illustrated in this chapter later. The databases we used in this paper are a collection of national flags and vehicle logos. This chapter will be organized as follow: 3.1 Colour Descriptor, 3.2 Shape Descriptor, 3.3 Datasets, and 3.4 Similarity Score of Computation.

## 3.1 Colour Descriptor

*Figure 3-1 Overview of colour descriptor*

Figure 3-1 has illustrated how the colour descriptor worked. First when a colour image is input for example Malaysia Flag, it will collect the BGR colour space information from Malaysia Flag. After that, the whole BGR colour space will be split into 3 channels, which are B channel, G channel, and R channel. Each colour channel can represents 256 bins of colour. In order to reduce the complexity during the mathematics calculation in later processes, we quantized the colour to 8 bins. Furthermore, we extract all the colour pixels and group the same colour pixels together and calculate the colour votes. Then we sort the colour votes in descending order. Next, we merge the similar colour pixels into single colour according to the rank of the colour that has calculated previously. Finally, we recalculate the colour votes and sort them again in descending order.

### 3.1.1 Colour Quantization

Colour Quantization is a method to reduce the colour bins of an image, usually with the intention that the new image should be as visually similar as possible to the possible image. Figure 3-2 shows the difference between original and quantized image.

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |

*Figure 3-2 (a) Original image; (b) Quantized image*

Figure 3-2 (a) is the original image of Malaysia Flag, the colour bins of each BGR channels are ranged from 0 to 255. After quantizing the colour bins 32 times of the original colour bins, Figure 3-2 (b) contains colour bins that are ranged from 0 to 7. In fact, the image should be visually black because colour bins from 0 to 7 are belonged to visually black pixels. Thus, in order to show the image in a colour form, we need to imply the formulas as shown below:

After that, we will extract all the colour pixels from Figure 3-2 (b), the steps are as follow:

1. Extract all the colour pixels;
2. Accumulate same colour pixels together to form a colour vote and store in a vector;
3. Sort the colour votes in descending order.

The result is then plotted as a histogram as shown below.

|  |
| --- |
|  |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 |

*Figure 3-3 Sorted quantized colour histogram*

### 3.1.2 Dominant Colours Formation

From the Figure 3-3 above, we can see that colour 1, colour 9, colour 17, colour 18 and colour 21 are visually similar as white-grey, whereas colour 2, colour 3, colour 5, colour 10, colour 12, and colour 14 are visually similar as red-brown. Therefore, we merge those similar colours into one single colour, the final merged colour is decided by the rank of the quantized colour. For example, colour 9, colour 17, colour 18, and colour 21 will be converted to colour 1, because colour 1 has the highest rank among them.

After few time testing, we have figured out that thresholds with 32 x 3 and 32 x 6 are the most suitable to merge the similar colours. The formula is presented as followed:

I*f the following conditions are true:*

*If those conditions are true, then,*

*Where B is B channel of dominant colour, G is G channel of dominant colour, R is R channel of dominant colour, Bi is B channel of similar colour, Gi is G channel of similar colour, Ri is R channel of similar colour.*

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |

*Figure 3-4 (a) Histogram before merging; (b) Histogram after merging*

Figure 3-4 illustrates that the number of colours is strictly reduced from 167 colours at the beginning to 4 colours at the end. Thus Figure 3-4 (b) is the dominant colours of Malaysia Flag, which are red, grey, blue, and yellow.

Besides that, Figure 3-5 illustrates that after merging similar colours, the colour image has become more evenly and easier to differentiate the colour. This will be an advantage for the similarity score calculation later.

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |

*Figure 3-5 (a) Quantized colour image; (b) Merged colour image*

### 3.1.3 Symmetry Picture

|  |  |  |
| --- | --- | --- |
|  |  |  |
| (a) | (b) | (c) |
|  |  |  |
| (d) | (e) | (f) |
|  |  |  |
| (g) | (h) | (i) |

*Figure 3-6 2 First row is Germany; Second row is Germany2; Third row is Belgium*

For second stage sorting, we may divide each query image and dataset image into symmetry half. This purpose is to further filter those images which have similar colour, but not similar colour region position. For instance, both Germany and Belgium share the same colour of flags. The only difference between 2 flag is the colour region position of each colour is different, for example, in Germany flag, black colour region is on the top, whereas in Belgium flag, black colour region is on the left. Therefore, after we further comparing the colour difference between left and right part of flags, Belgium flag will be taken out if Germany flag is the query image, and Germany 2 will be sorted right behind it, because Figure 3-6 (b) and Figure 3-6 (e) share the same colour pattern, and the same goes to Figure 3-6 (c) and Figure 3-6 (f).

## 3.2 Shape Descriptor

*Figure 3-6 Overview of shape descriptor*

Figure 3-6 has briefly described how to get the outer most contour of an image. For instance, a Ferrari Logo, it is a shield-based shape, thus we need to extract the shield-based shape as a foreground and calculate the central normalized moments. Firstly, we will get a colour image as an input. Second, we will convert the colour image to grey scale image by using the cvtColor library that provided by OpenCV itself. Third, we threshold the grey scale image with the number of 220 so that it can be converted again to the binary black and white image. After that, we find out the outer most contour of the model and flood fill the background with white pixels. Next, we invert the flood-filled image, and apply the bitwise or operation to binary and inverted image so that we can get a white region of foreground image. Finally, we calculate the central normalized moments of the white region for the similarity score computation at later stage.

### 3.2.1 Foreground Retrieval

|  |  |
| --- | --- |
|  |  |
| Step 1: Colour image | Step 2: Grey scale image |
|  |  |
| Step 3: Binary image | Step 4: Flood-filled image |
|  |  |
| Step 5: Inverted image | Step 6: Foreground image |

*Figure 3-7 Step-by-step to retrieve foreground image*

Every logo image has a white background, thus Figure 3-7 demonstrates how to eliminate the white background and retrieve the foreground image. After step 2, we address a binary threshold value of 220 to the grey scale image, that is convert values below 220 to 255 (white pixel), whereas values from 220 to 255 will be converted to 0 (black pixel). Next from point (0, 0) onwards, this coordinate is located at the corner of top left image, flood fill the background with white pixels. The definition of the background is defined by the region outside the largest outer most contour of Ferrari Logo. In step 5, we interchange the white and black pixels, so that we can apply bitwise or operation to binary image as well as inverted image in order to retrieve foreground image in step 6.

### 3.2.2 Central Normalized Moment

Central normalized moment can be computed up to third order, of a vector shape or a rasterized shape. It consists of 7 moment values that are invariant to image shape and scale. This means that a small round shape and a big round shape will return the approximately equal results, this will be illustrated in Figure 3-8 later. The 7 moment values are nu02, nu03, nu11, nu12, nu20, nu21, and nu30. In addition, nu00 and nu10 are not stored because they are zeros.

|  |  |
| --- | --- |
| **Central Normalized Moment** | **Value** |
| nu02 | 0.097626 |
| nu03 | 0.00610753 |
| nu11 | 0.000949622 |
| nu12 | 0.00013218 |
| nu20 | 0.0697717 |
| nu21 | -0.00526489 |
| nu30 | -0.00018855 |

*Table 3-1 An example of central normalized moment of Ferrari logo*

These 7 values are calculated based on the white foreground region. They will be useful in later stage when computing the Euclidean distance for similarity difference. Each image will have its individual and different moment values.

|  |  |
| --- | --- |
|  |  |
| 0.0404318 | 0.121788 |
| -0.0000010315 | 0.00000122791 |
| -0.00000109836 | -0.00000791884 |
| -0.00000018929 | 0.0000164598 |
| 0.156624 | 0.0519969 |
| 0.00000420002 | -0.000000516944 |
| 0.00000729785 | -0.00000700647 |

*Table 3-2 Central normalized moments of Bugatti and Maserati logos*

As mentioned before, central normalized moment is not affected by rotation, thus a horizontal ellipse and vertical ellipse will be treated as different images. For instance, Table 3-2 shows the central normalized moments of both horizontal and vertical ellipse shapes. We can clear notify that, there is a big gap between their central normalized moment values. This method can help us to identify same shape but different rotation variant of logos, for example Bugatti logo and Maserati Logo.

## 3.3 Combined Descriptor

This descriptor combines both colour and shape descriptors. First, it will implement shape descriptor to find out the similar shapes, and then later implement colour descriptor to sort out similar colours. The steps of finding the similar shapes are totally equal as shape descriptor, whereas there will be a bit different for similar colours finding.

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |

*Figure 3-8 (a) Foreground of Ferrari logo; (b) Ferrari logo*

We need to input 2 data variables for colour descriptor, they are original image as well as the foreground of original image as shown in Figure 3-8. After that, we scan through the foreground image, and only retrieve colour value from the original image within the foreground (white pixels). This is because every logo will have a background which is a noise. In addition, most of the time, the background will occupy most of the space and become the leading dominant colour. So, to avoid background colour become the leading dominant colour, we need to eliminate the background, in order to get more accurate value. Figure 3-9 will illustrate the dominant colour histogram after eliminating the background colour. In our case, the background colour is white, thus we will not seeing white colour be in dominant colour histogram.

|  |  |
| --- | --- |
|  | |
| (a) | |
|  |  |
| (b) | (c) |

*Figure 3-9 (a) Proton logo; (b) Dominant colours histogram before merging; (c) Dominant colours histogram after merging*

## 3.4 Datasets

We have 2 separate databases which contain National Flag and Vehicle Logo respectively. All of the datasets within the 2 database were obtained from internet.

For National Flag datasets, the size dimension is controlled around 299 x 168. In addition, there is no any background for the National Flag datasets. Furthermore, they consist of original flags, wiggled flags, as well as shadow flags. Lastly, the total number of national flags is 76.



*Figure 3-10 National flag datasets*

For Vehicle Logo datasets, the size dimensions are controlled around 240 x 240, 320 x 170, and 374 x 374. In addition, there is a white background for all the vehicle logo datasets. Furthermore, they consist of round-based shape, shield-based shape, horizontal ellipse-based shape, vertical ellipse-based shape, wings attached to circle-based shape, as well as 3 attached diamonds-based shape. Lastly, the total number of vehicle logos is 61.

*Figure 3-11 Vehicle logo datasets*

## 3.5 Similarity Score of Computation

In this section, we will be using Euclidean distance to measure the similarity difference between query image and dataset image. Since we are using colour and shape descriptors to retrieve similar images, therefore, Euclidean distance will be applied on both colour and shape descriptors respectively. Similarity score of computation will be further described in 3 sections later, they are 3.5.1 Similarity Score Calculation of Colour Descriptor, 3.5.2 Similarity Score Calculation of Shape Descriptor, and 3.5.3 Similarity Score of Calculation of Combined Descriptor.

For colour descriptor, we will have 2 stages of sorting. First stage, we calculate the Euclidean difference of a whole image and compare it. Second stage, we calculate the Euclidean differences from symmetry left image as well as symmetry right image and compare them.

For shape descriptor, we also have 2 stages of sorting. First stage, we sort the shapes by using central normalized moments. Second stage, we will further sort the shapes by using Hu moments. This is because Hu moments can give more precise values. However, Hu moments have a big disadvantage, which is it will consider rotation variant which is not suitable in our project.

For combined descriptor, we will apply shape descriptor first, and then colour descriptor. This is because our major datasets for this descriptor are vehicle logo datasets. Thus shape has the higher priority than colour variant.

### 3.5.1 Similarity Score Calculation of Colour Descriptor

The similarity score calculation of colour descriptor will be mainly tested on national flag datasets. The way we find compute their similarity score of colours is using corresponding colour of pixels percentage.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  | ? |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |
|  |  |

*Figure 3-12 Find the corresponding colour*

Let’s assume that those 6 colours from the left of Figure 3-12 are the dominant colours from the query image, and those 6 colours from the right of Figure 3-12 are the dominant colours from the dataset image. The first dominant colour from query image is red, so the descriptor will scan all of the dominant colours from dataset image and hopefully a red colour is found. If the red colour is found, then the Euclidean distance of the red pixels percentage from both images will be counted. If red colour is not found in the other case, then we will assign 0 for the red pixel percentage in the dataset image, thus in the Euclidean distance formula, the red pixels percentage from query image will minus 0. After few time of testing, the thresholds are fixed at 2 and 3. The formula is demonstrated as followed:

*Where B is the B channel, G is the G channel, R is the R channel, 2 and 3 are the threshold values for the similar colours.*

*The pseudo codes of finding Euclidean distance of colour pixel percentage can be express as followed:*

*If query colour = dataset colour,*

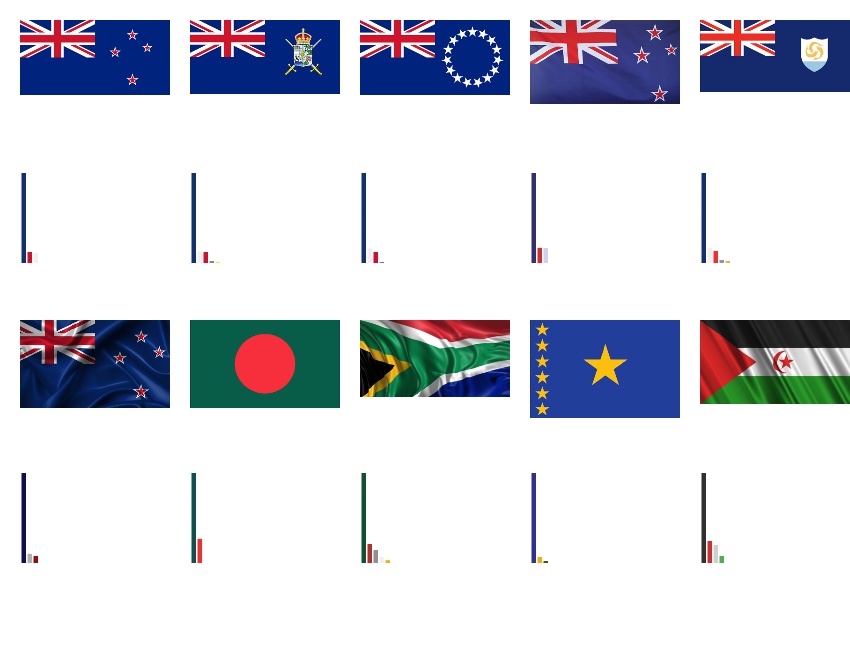
*for (i = 0 to i = n query colour) {  
 for (j = 0 to j = n dataset colour) {  
 if (all conditions above are met)*

*If query has more colour than dataset, then the remaining colour will be rq,*

*If dataset has more colour than query, then the remaining colour will be rd,*

*And lastly, the Euclidean distance will be as followed,*

Until here, we are doing the first stage filtering, where we are going to filter out top 10 most similar images out of the database images.



*Figure 3-13 First stage filtering using pixel percentage*

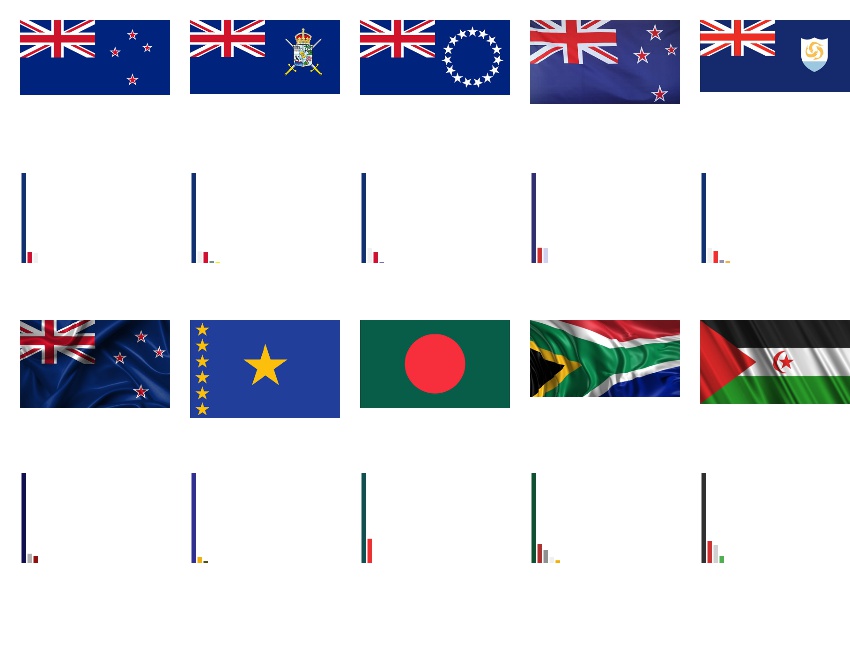
The table of Euclidean distance from the Figure 3-13 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Pixel Percentage** |
| Image 1 | 0 |
| Image 2 | 0.0410021 |
| Image 3 | 0.0492752 |
| Image 4 | 0.0764392 |
| Image 5 | 0.081641 |
| Image 6 | 0.13596 |
| Image 7 | 0.14823 |
| Image 8 | 0.169097 |
| Image 9 | 0.174316 |
| Image 10 | 0.178923 |

*Table 3-3 Euclidean distance table of Figure 3-13*

For second stage filtering, we will use the location information in similarity searching so we divide the image into symmetry half, and then, compute the Euclidean distance between symmetry left from query image and symmetry left from dataset image, as well as symmetry right from query image and symmetry right from dataset image. After that we find the average value of Euclidean distance between left and right image.

The sample result will be illustrated in Figure 3-14 below using Euclidean distance of average pixel percentage between image from left half of the original image and image from right half of the original image.



*Figure 3-14 Second stage filtering using average pixel percentage*

The table of Euclidean distance from the Figure 3-14 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Average Pixel Percentage** |
| Image 1 | 0 |
| Image 2 | 0.0446361 |
| Image 3 | 0.0479985 |
| Image 4 | 0.0750537 |
| Image 5 | 0.0932345 |
| Image 6 | 0.103566 |
| Image 7 | 0.179762 |
| Image 8 | 0.250182 |
| Image 9 | 0.409341 |
| Image 10 | 0.595014 |

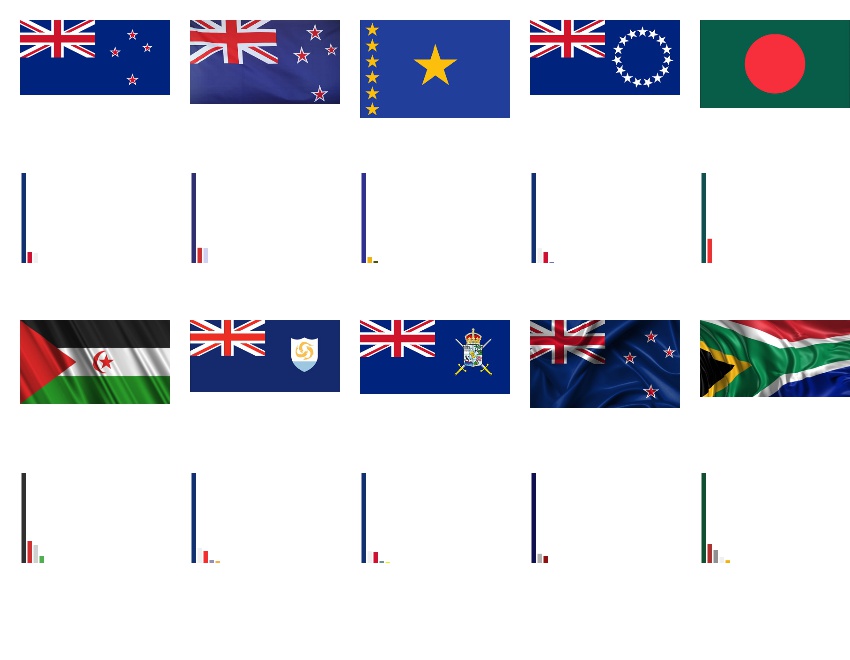
*Table 3-4 Euclidean distance table of Figure 3-14*

Moreover, we have 2 types of method in second stage filtering, we have presented the first one, the second one is to find the moments of the corresponding colour region. For example, in Figure 3-15 (a), it is corresponded to blue colour, (b) is corresponded to red colour, and (c) is corresponded to white colour. The 7 moment values are nu02, nu03, nu11, nu12, nu20, nu21, and nu30 which are computed for the three binary mask images.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| (a) | (b) | (c) |

*Figure 3-15 Binary images from each corresponding colour in New Zealand flag*

The sample result that uses Euclidean distance of colour region moment will be demonstrated below.



*Figure 3-16 Second stage filtering using colour region moment*

The table of Euclidean distance from the Figure 3-16 will be listed belowThe 7 moment values are nu02, nu03, nu11, nu12, nu20, nu21, and nu30

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Colour Region Moment** |
| Image 1 | 0 |
| Image 2 | 0.154447 |
| Image 3 | 0.30573 |
| Image 4 | 0.645115 |
| Image 5 | 1.02445 |
| Image 6 | 1.0617 |
| Image 7 | 1.87079 |
| Image 8 | 1.93254 |
| Image 9 | 2.19036 |
| Image 10 | 2.26103 |

*Table 3-5 Euclidean distance table of Figure 3-16*

### 3.5.2 Similarity Score Calculation of Shape Descriptor

In this section, 2 types of inputs are taken to measure the Euclidean distance for shape descriptor. They are central normalized moments and Hu moments. Both central normalized moments and Hu moments contain 7 values to define a shape, the only difference is Hu moments involve in rotation variant. The similarity score calculation of shape descriptor is mainly done within Vehicle Logo Database. We used the matchShapes library that provided by OpenCV to calculate the similarity difference value between query image and dataset image. Whereas the formula of computing Euclidean distance from central normalized moments is show below:

The first stage of filtering is central normalized moments, whereas the second stage of filtering is Hu moments. The following tables will show the results after computing the Euclidean distance with central normalized moments and Hu moments respectively. The type of shape will be circle-based shape, horizontal ellipse-based shape, vertical ellipse-based shape, shield-based shape, wings attached to circle-based shape, and 3 attached diamonds-based shape.

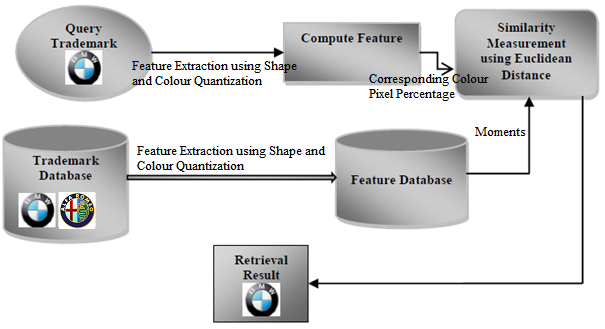
|  |  |
| --- | --- |
|  | |
| Image 1 | 0 |
| Image 2 | 0.000458483 |
| Image 3 | 0.000745379 |
| Image 4 | 0.000777143 |
| Image 5 | 0.00081234 |
| Image 6 | 0.000815794 |
| Image 7 | 0.000903516 |
| Image 8 | 0.000935308 |
| Image 9 | 0.000959295 |
| Image 10 | 0.000996638 |

*Table 3-6 First stage filtering using central normalized moments*

|  |  |
| --- | --- |
|  | |
| Image 1 | 0 |
| Image 2 | 0.00000498457 |
| Image 3 | 0.0000526338 |
| Image 4 | 0.0000960546 |
| Image 5 | 0.0000964921 |
| Image 6 | 0.0000965197 |
| Image 7 | 0.0000965495 |
| Image 8 | 0.0000966039 |
| Image 9 | 0.0000966624 |
| Image 10 | 0.0000966697 |

*Table 3-7 Second stage filtering using Hu moments*

### 3.5.3 Similarity Score of Calculation of Shape and Colour Descriptor



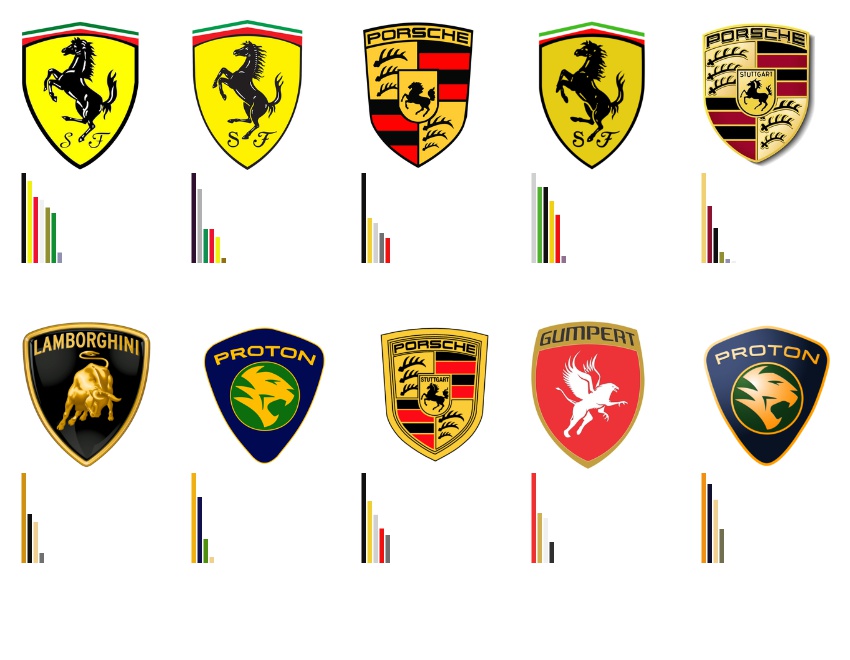
*Figure 3-17 Overview of combined descriptor*

Shape and colour descriptor also called the combined descriptor, it involves 2 stages. First stage, we apply shape descriptor to find the Euclidean distance of moments’ value to sort out similar shapes. At second stage, we apply colour descriptor to find out the Euclidean distance of corresponding colour pixel percentage to group the similar colour images together.



*Figure 3-18 First stage of sorting*

As we can see from Figure 3-18, after first stage of sorting using shape descriptor, we manage to find out all shield-based shape. Next, we would like to group the similar colours together, so that Ferrari Logos, Proton Logos, and Porsche Logos can be grouped together respectively. Before we apply colour descriptor to Figure 3-18, we need to eliminate background colour, because most of the time, the background colour will occupy a lot of space and the leading dominant colour will be background colour. The result will be inaccurate if we consider the background as one of the dominant colour. Thus, we need to eliminate this unstable factor before we proceed to the next step. This step will be illustrated in Figure 3-19 after eliminating background colour and the second stage of sorting.



*Figure 3-19 Second stage of sorting*

Take a note that, the background must be a plain colour, which is either white, black, green, blue, etc. The background cannot contain 2 or more combination of colours. For example, Figure 3-20 (a) is an ideal background, whereas Figure 3-20 (b) is not an ideal background.

|  |  |
| --- | --- |
|  |  |
|  |
| (a) | (b) |

*Figure 3-20 (a) Ideal background; (b) Inappropriate background*

# CHAPTER 4 SYSTEM IMPLEMENTATION, TESTING, AND VERIFICATION

## 4.1 Hardware and Software Environments Used in Project Implementation

### 4.1.1 Hardware Specification

|  |  |
| --- | --- |
| **Operating System** | Windows 8.1 (64-bit) |
| **Memory** | 8 GB |
| **Hard Disk** | 1 TB, 5400 RPM (SATA) |
| **Processor** | Intel® Core™ i7-5500U CPU @ 2.4 GHz |
| **Graphic Card** | NVIDIA GeForce 840M |
| **Mouse** | 3-button wheel mouse, optical |
| **Web Browser** | Google Chrome (32-bit) |
| **Backup Equipment** | External hard drive |
| **Network Adapter (Multi-user Functionality)** | 2 MBps |

*Table 4-1 Hardware environments used in project implementation*

### 4.1.2 Software Specification

|  |  |
| --- | --- |
| Compiler | Microsoft Visual Studio 2013 |
| Integrated Development Environment | Visual Studio |
| Languages Used | C++, OpenCV |

*Table 4-2 Software environments used in project implementation*

## 4.2 System Code Implementation

In this project, we have a total of 15 functions, among them, there are 8 main functions and 7 secondary functions.

The 8 main functions are as followed:

1. int sortDominantColour(vector<int> &colourVote, vector<pair<int, int>> &bin\_sort, int &max\_fre);
2. int wholeImage(Mat &image, Mat &resultImage, int \*topRankColour, int \*pixelPercentage, Mat &histogram);
3. void centralNormalizedMoment(Mat &binaryImage, double \*nuMoment);
4. double euclideanDistance\_colourRegion(Mat &image, Mat &trainImage, int \*colour\_query, int \*colour\_dataset, int totalColour\_img, int totalColour\_timg);
5. void findContour(Mat &image, Mat &resultImage, double \*nuMoment);
6. int eliminateBackground(Mat &image, Mat &processedImage, int \*topRankColour, int \*pixelPercentage, Mat &histogram);
7. double euclideanDistance\_nuMoment(double \*param\_query, double \*param\_dataset, int arrSize);
8. double euclideanDistance\_pixelPercentage(int \*colour\_query, int \*colour\_dataset, int \*pp\_query, int \*pp\_dataset, int arrSize\_query, int arrSize\_dataset, int imgSize\_query,

int imgSize\_dataset);

The sortDominantColours function will be drawing dominant colours histogram and calculating total number of dominant colours. The input data variable will be vector<int> &colourVote which is the pixel percentage of each corresponding dominant colour, after the function is executed, it will return total number of dominant colours. The vector <int, int> &bin\_sort will be storing the sorted colour vote of each corresponding dominant colour.

The wholeImage function will be quantizing the colours to 8 bins, and then merging similar colours of whole image. This function is suitable for national flag datasets which do not have background. The input data variable will be Mat &image which is the original image.

The findContour function will be doing foreground retrieval and calculating central normalized moments. The input data variable will be Mat &image which is the original image, after the function is executed, the foreground binary image will be store in Mat &resultImage, and the 7 central normalized moments will be stored in double \*nuMoment.

The eliminateBackground function is the same as wholeImage function. However, it only retrieves colours from foreground region. This function is suitable for vehicle logo datasets which do have background. The input data variable will be Mat &image which is the original image and Mat &processImage which is the foreground of the original image.

The euclideanDistance\_nuMoment function will be doing Euclidean distance calculation with central normalized moments. The input data variables are double \*param\_query which stores central normalized moments from query image, and double \*param\_dataset which stores central normalized moments from dataset image. The int arrSize is the array size and it will always be 7 because there are only 7 values from central normalized moments as discussed in the previous chapter. Finally, after the function is executed, this function will return a Euclidean distance value.

The euclideanDistance\_pixelPercentage function will be doing Euclidean distance calculation with colour pixel percentage. The input data variables are int \*pp\_query which is colour pixel percentage from query image, and int \*pp\_dataset which is colour pixel percentage from dataset image. The int \*colour\_query is the colour value from query image, and int \*colour\_dataset is the colour value from dataset image, both of them will be computed to find the similar colours. The int arrSize\_query is total dominant colour from query image, whereas int arrSize\_dataset is total dominant colour from dataset image. Next, int imgSize\_query is the image size dimension from query image, and int imgSize\_dataset is the image size dimension from dataset image. The image size dimension is considered for the purpose of normalizing the colour pixel percentage. Finally, after the function is executed, this function will return a Euclidean distance value.

The 7 secondary functions are as followed:

1. void displayQueryImage(Mat &image);
2. int calcColourBin(Vec3b colour);
3. Vec3b convert2Bgr(int colourBin);
4. void roi(Mat &image, Mat &roi\_left, Mat &roi\_right);
5. void sort2DArray(double arr[][2], int row);
6. Mat attachResults2AWhiteBgWithHist(double arr[][2], vector<Mat> &trainImage, vector<Mat> &histogram, int number);
7. Mat attachResults2AWhiteBg(double arr[][2], vector<Mat> &trainImage, int number);The displayQueryImage function will prompt a window out, and display the original query image in it.

The calcColourBin function is a sub function of findQuanDominantColours, it convert the BGR colour space to a decimal value.

The convert2Bgr function works the other way of calcColourBin that it converts the decimal colour value to BGR 3 channels.

The roi function will separate the original image to symmetry left and right images.

The sort2DArray will sort the 2 dimensional array in ascending order.

The attachResults2AWhiteBg will attach the 7 sorted results on a white background and then display.



*Figure 4-1 Flow diagram of the implemented system*

After user input the query image, the system will ask user for 3 options: Colour Descriptor, Shape Descriptor, and Combined Descriptor. The flow path of each descriptor will be shown as followed:

For Colour Descriptor:

1. findQuanDominantColours() -> euclideanDistance\_Bgr()-> Display Result;
2. findQuanDominantColours() -> euclideanDistance\_pixelPercentage()-> Display Result

For Shape Descriptor:

1. findContour() -> euclideanDistance\_nuMoment() -> Display Result

For Combined Descriptor:

1. findContour() -> euclideanDistance\_nuMoment() -> findQuanDominantColours() -> euclideanDistance\_pixelPercentage()-> Display Result

## 4.3 System Code Testing and Verification

In this section, we will test and verify which variant of images our proposed system can handle. For example, colour brightness image, wiggled image, shadow image, rotation image, different scaling image, and 3D image.

### 4.3.1 Colour Brightness Image

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
| Query Images | Retrieved Images |

*Figure 4-3 Dominant colours histograms of colour brightness variant*

From Figure 4-3, we can discover that although some of the images are same designs but with different colour brightness, nevertheless our system can bypass this condition and retrieve the correct result.

### 4.3.2 Wiggled Image

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
| Query Images | Retrieved Images |

*Figure 4-4 Dominant colours histograms of wiggled variant*

From Figure 4-4, we can discover that although some of the images are same designs but wiggled, nevertheless our system can bypass this condition and retrieve the correct results.

### 4.3.3 Shadow Image

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
| Query Images | Retrieved Images |

*Figure 4-5 Dominant colours histograms of shadow variant*

From Figure 4-5, we can discover that although some of the images are same designs but with shadow, nevertheless our system can bypass this condition and retrieve the correct results.

### 4.3.4 Rotation Image

|  |  |
| --- | --- |
|  |  |
| 0.0010817 | |
|  |  |
| 0.155178 | |
| Query Images | Retrieved Images |

*Figure 4-6 Euclidean distance of rotation variant*

From Figure 4-6, we can discover that although some of the images are same designs but with rotation variant, nevertheless our system can bypass this condition and retrieve the correct results.

### 4.3.5 Different Scaling Image

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
| Query Images | Retrieved Images |

*Figure 4-7 Dominant colours histograms of different scaling variant*

From Figure 4-7, we can discover that although some of the images are same designs but with different scaling or proportion, nevertheless our system can bypass this condition and retrieve the correct results.

### 4.3.6 3D Image

|  |  |
| --- | --- |
|  |  |
| 0.055595 | |
| Query Images | Retrieved Images |

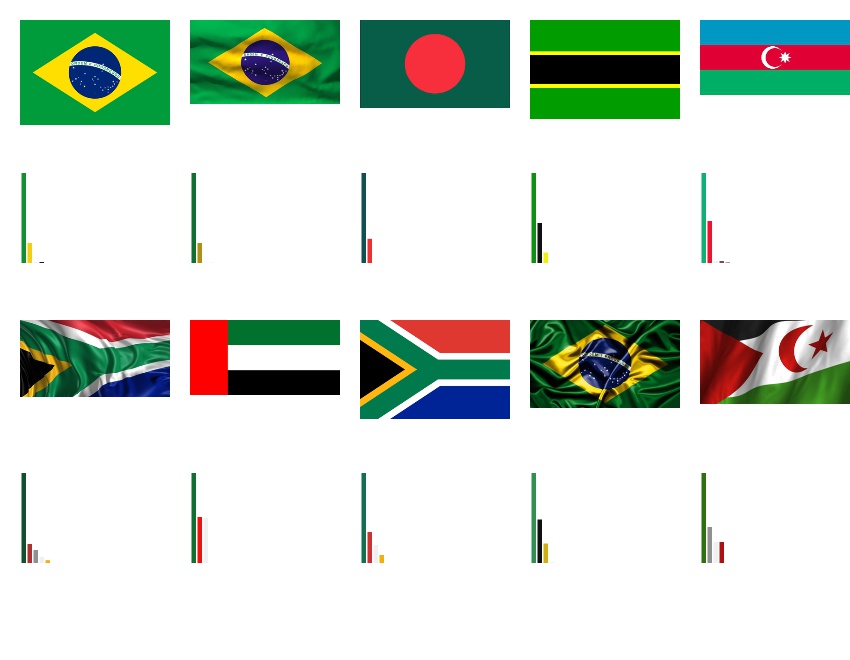
*Figure 4-8 Euclidean distance of 3D variant*

From Figure 4-8, we can discover that our system can recognize similar 2D and 3D image and lastly retrieve the correct results.

# CHAPTER 5 SYSTEM ANALYSIS AND PERFORMANCE EVALUATION

In this section, we will be testing out several types of query images, along with their retrieval results as well as Euclidean distance between query image and dataset image in a table. We will categorize the comparisons into 3 part which is 5.1 Colour Descriptor Analysis, 5.2 Shape Descriptor Analysis, and 5.3 Combined Descriptor Analysis. For colour descriptor, we will be testing on several kinds of query images like normal standard national flags, waving national flags, shadow national flags, etc. Besides, each national flag will have 2 sets of comparison tables, which are first stage sorting table and second stage sorting table. For shape descriptor, we will be the comparison on different kind of shape like circle-based shaped, ellipse-based shape, shield-based shaped, etc. In addition, each type of shape will have 2 comparison tables, they are first stage sorting with central normalized moments table and also second stage sorting with Hu moments table. Lastly, for the combined descriptor, we will be combining 2 descriptors and let see how is the result.

### 5.1 Colour Descriptor Analysis

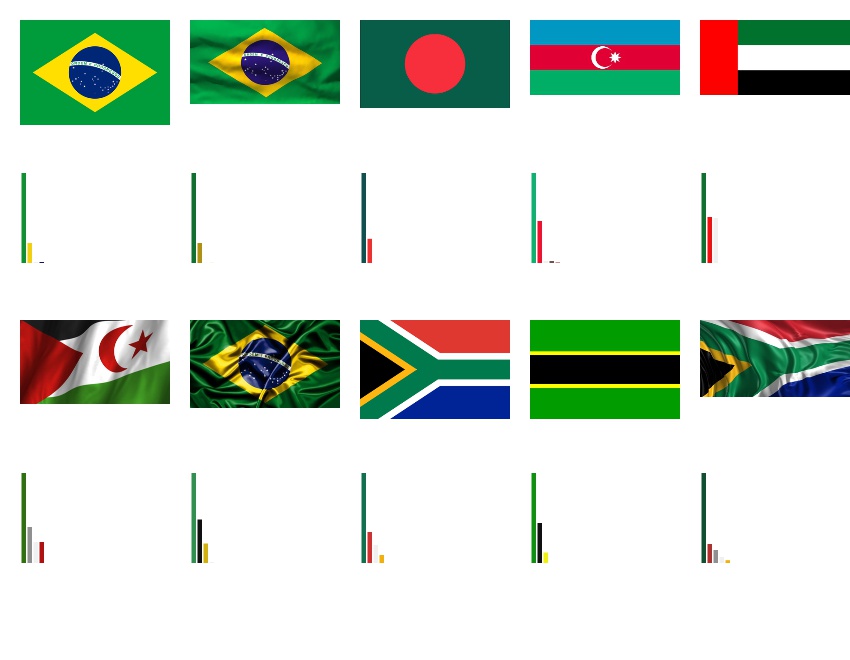


*Figure 5-1 Average Pixel Percentage*

The table of Euclidean distance from the Figure 5-1 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Average Pixel Percentage** |
| Image 1 | 0 |
| Image 2 | 0.150743 |
| Image 3 | 0.277731 |
| Image 4 | 0.345637 |
| Image 5 | 0.38019 |
| Image 6 | 0.523074 |
| Image 7 | 0.569647 |
| Image 8 | 0.617905 |
| Image 9 | 0.703828 |
| Image 10 | 0.768302 |

*Table 5-1 Euclidean distance table of Figure 5-1*

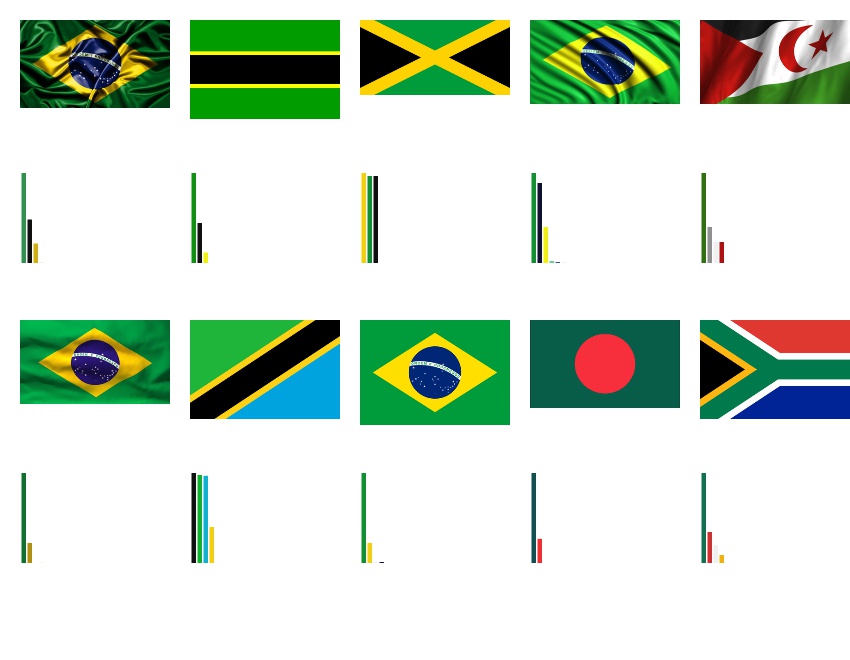


*Figure 5-2 Colour Region Moments*

The table of Euclidean distance from the Figure 5-2 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Colour Region Moments** |
| Image 1 | 0 |
| Image 2 | 0.291686 |
| Image 3 | 0.304885 |
| Image 4 | 0.340315 |
| Image 5 | 0.368557 |
| Image 6 | 0.599056 |
| Image 7 | 0.694131 |
| Image 8 | 0.972294 |
| Image 9 | 2.13358 |
| Image 10 | 3.10207 |

*Table 5-2 Euclidean distance table of Figure 5-2*

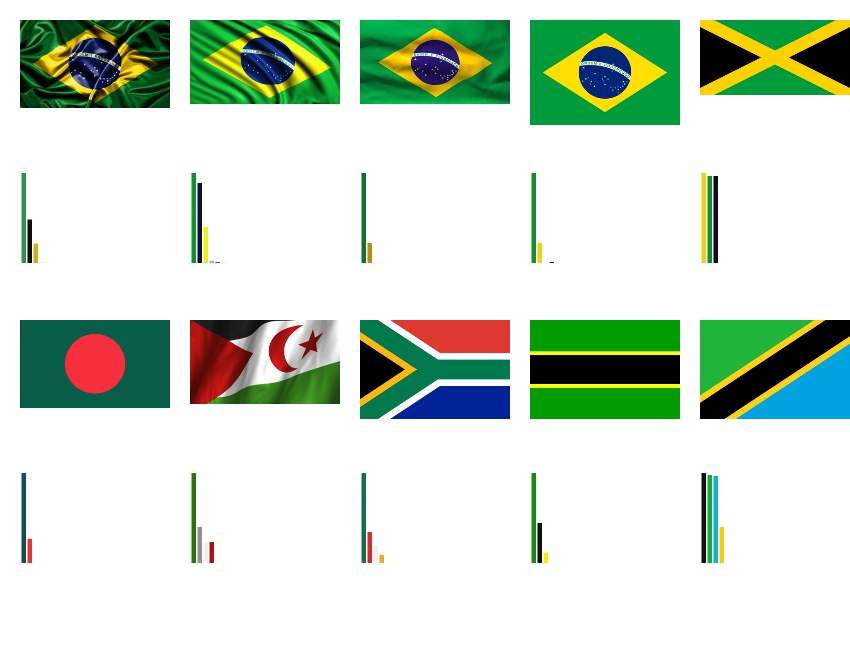


*Figure 5-3 Average Pixel Percentage*

The table of Euclidean distance from the Figure 5-3 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Average Pixel Percentage** |
| Image 1 | 0 |
| Image 2 | 0.404849 |
| Image 3 | 0.482863 |
| Image 4 | 0.523629 |
| Image 5 | 0.628531 |
| Image 6 | 0.645614 |
| Image 7 | 0.663299 |
| Image 8 | 0.703828 |
| Image 9 | 0.711502 |
| Image 10 | 0.924977 |

*Table 5-3 Euclidean distance table of Figure 5-3*

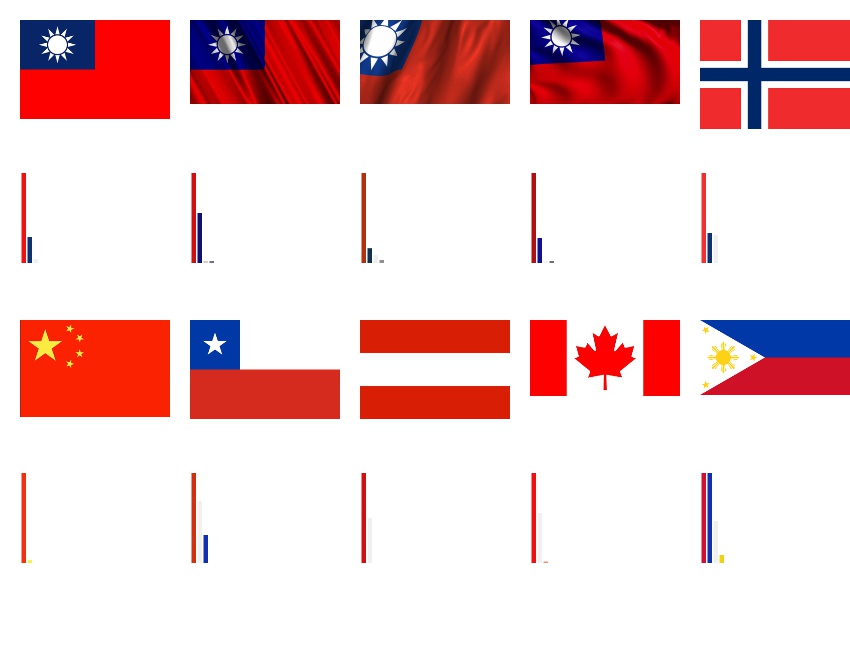


*Figure 5-4 Colour Region Moments*

The table of Euclidean distance from the Figure 5-4 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Colour Region Moments** |
| Image 1 | 0 |
| Image 2 | 0.290604 |
| Image 3 | 0.387024 |
| Image 4 | 0.405608 |
| Image 5 | 0.456505 |
| Image 6 | 0.480712 |
| Image 7 | 0.69203 |
| Image 8 | 1.05181 |
| Image 9 | 1.07556 |
| Image 10 | 1.07901 |

*Table 5-4 Euclidean distance table of Figure 5-4*

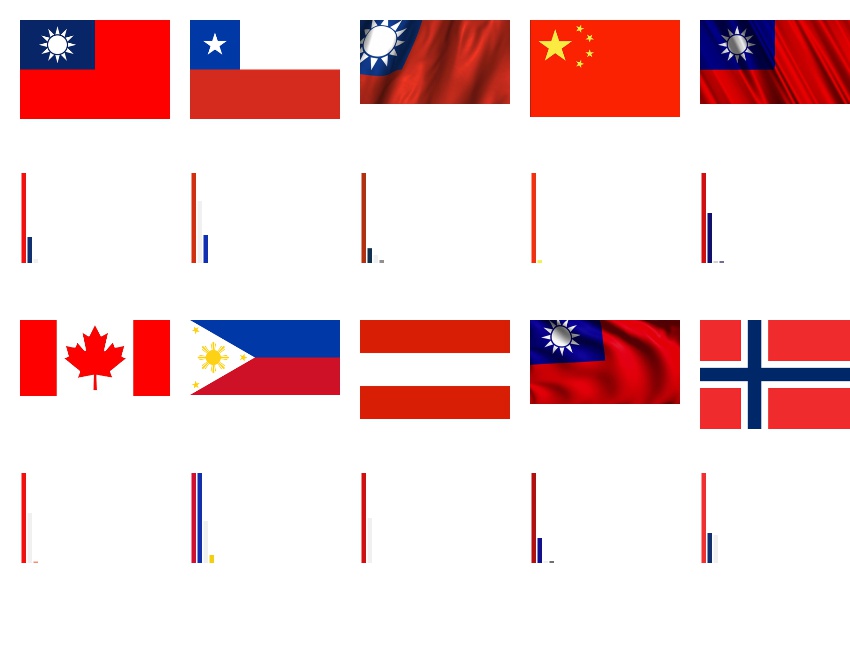


*Figure 5-5 Average Pixel Percentage*

The table of Euclidean distance from the Figure 5-5 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Average Pixel Percentage** |
| Image 1 | 0 |
| Image 2 | 0.0863913 |
| Image 3 | 0.102188 |
| Image 4 | 0.258614 |
| Image 5 | 0.273076 |
| Image 6 | 0.309323 |
| Image 7 | 0.442767 |
| Image 8 | 0.502685 |
| Image 9 | 0.520226 |
| Image 10 | 0.556836 |

*Table 5-5 Euclidean distance table of Figure 5-5*

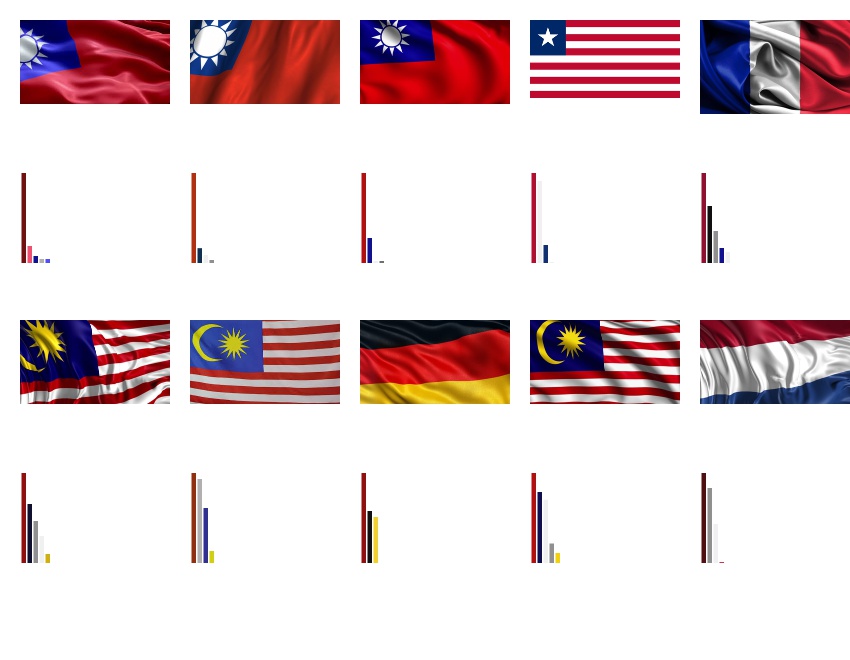


*Figure 5-6 Colour Region Moments*

The table of Euclidean distance from the Figure 5-6 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Colour Region Moments** |
| Image 1 | 0 |
| Image 2 | 0.175382 |
| Image 3 | 0.227792 |
| Image 4 | 0.25074 |
| Image 5 | 0.267022 |
| Image 6 | 0.278881 |
| Image 7 | 0.284307 |
| Image 8 | 0.341614 |
| Image 9 | 0.37105 |
| Image 10 | 0.478626 |

*Table 5-6 Euclidean distance table of Figure 5-6*

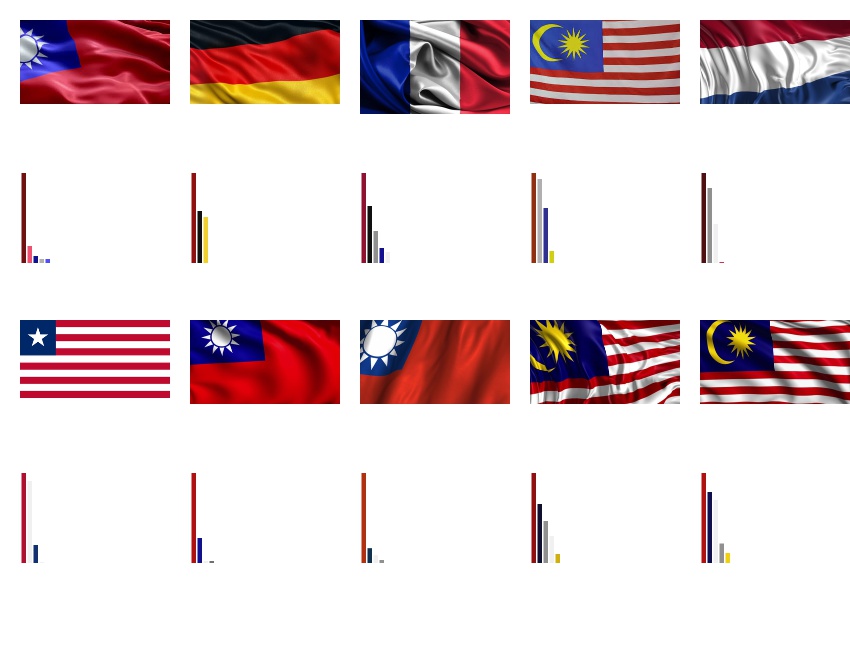


*Figure 5-7 Average Pixel Percentage*

The table of Euclidean distance from the Figure 5-7 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Average Pixel Percentage** |
| Image 1 | 0 |
| Image 2 | 0.229257 |
| Image 3 | 0.388254 |
| Image 4 | 0.531195 |
| Image 5 | 0.598147 |
| Image 6 | 0.602399 |
| Image 7 | 0.603098 |
| Image 8 | 0.669876 |
| Image 9 | 0.723976 |
| Image 10 | 0.897006 |

*Table 5-7 Euclidean distance table of Figure 5-7*

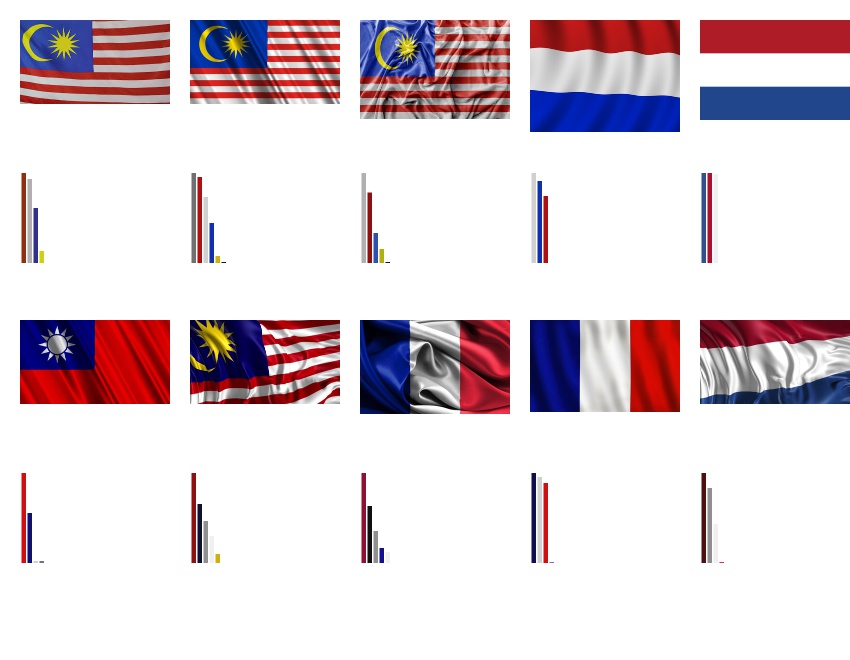


*Figure 5-8 Colour Region Moments*

The table of Euclidean distance from the Figure 5-8 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Colour Region Moments** |
| Image 1 | 0 |
| Image 2 | 0.608216 |
| Image 3 | 0.670681 |
| Image 4 | 0.691883 |
| Image 5 | 0.704086 |
| Image 6 | 0.750356 |
| Image 7 | 0.765652 |
| Image 8 | 0.945757 |
| Image 9 | 1.72449 |
| Image 10 | 4.06524 |

*Table 5-8 Euclidean distance table of Figure 5-8*

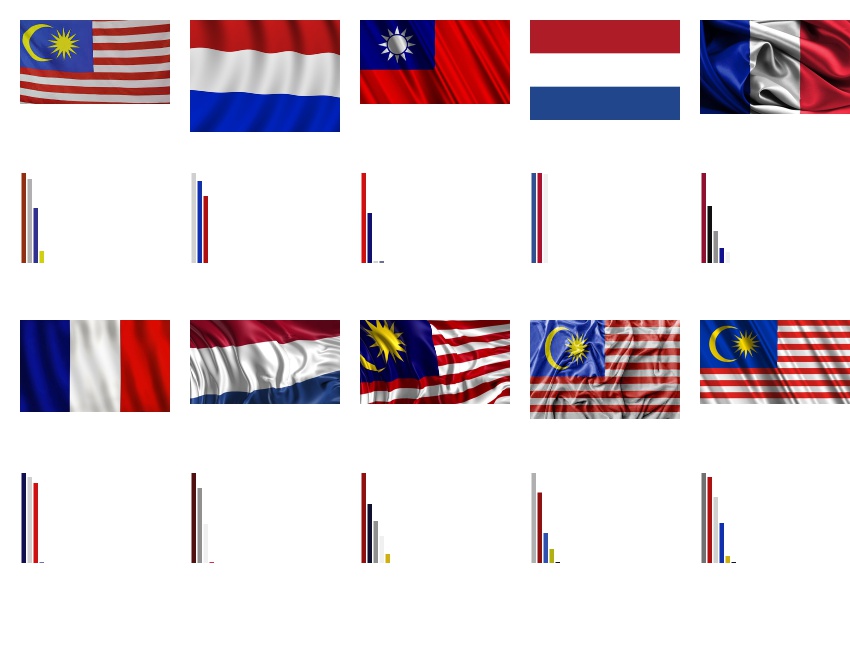


*Figure 5-9 Average Pixel Percentage*

The table of Euclidean distance from the Figure 5-9 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Average Pixel Percentage** |
| Image 1 | 0 |
| Image 2 | 0.227817 |
| Image 3 | 0.297944 |
| Image 4 | 0.326553 |
| Image 5 | 0.477787 |
| Image 6 | 0.496661 |
| Image 7 | 0.532039 |
| Image 8 | 0.596492 |
| Image 9 | 0.655738 |
| Image 10 | 0.750526 |

*Table 5-9 Euclidean distance table of Figure 5-9*

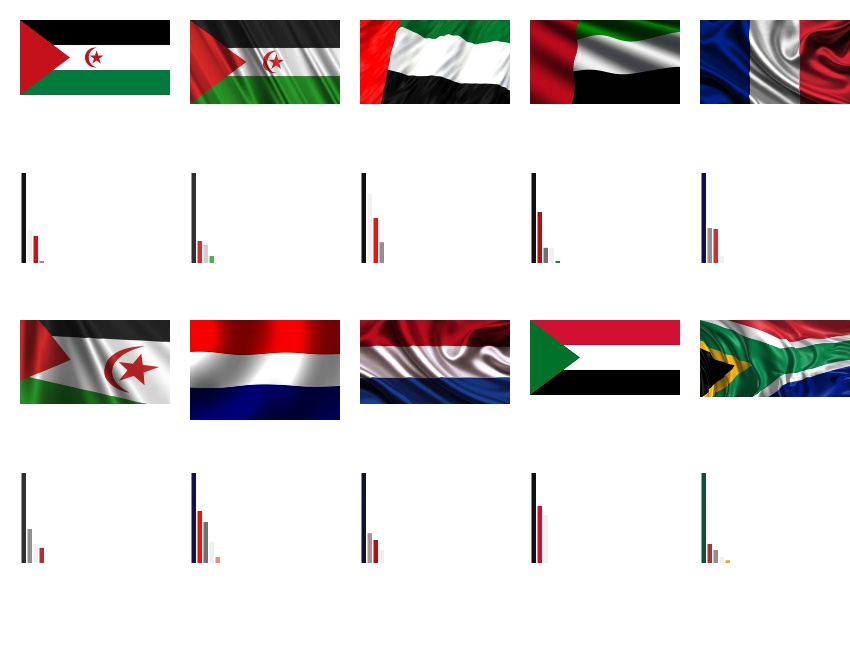


*Figure 5-10 Colour Region Moments*

The table of Euclidean distance from the Figure 5-10 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Colour Region Moments** |
| Image 1 | 0 |
| Image 2 | 0.362769 |
| Image 3 | 0.513621 |
| Image 4 | 0.642028 |
| Image 5 | 0.755431 |
| Image 6 | 0.772109 |
| Image 7 | 0.780799 |
| Image 8 | 1.21058 |
| Image 9 | 2.66047 |
| Image 10 | 9.78698 |

*Table 5-10 Euclidean distance table of Figure 5-10*

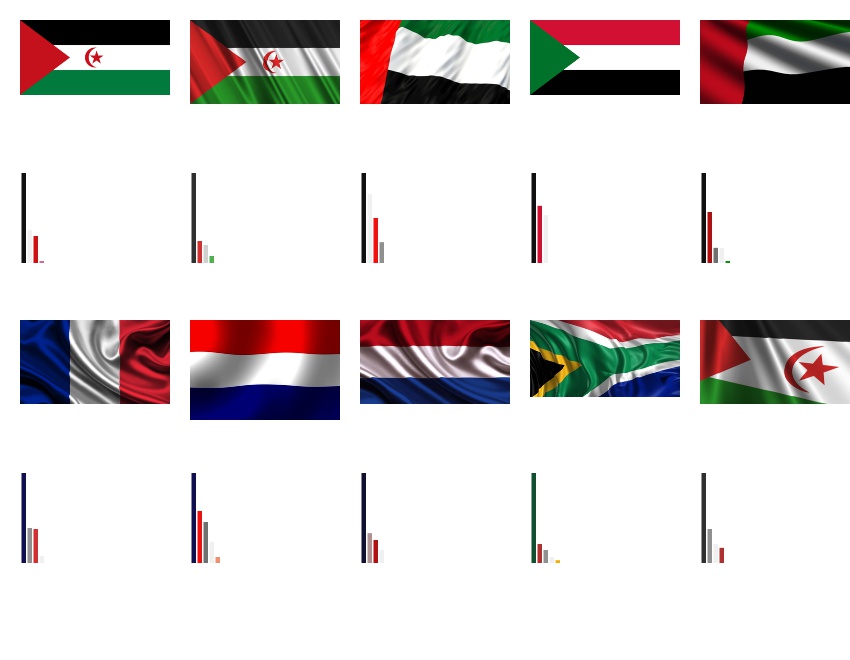


*Figure 5-11 Average Pixel Percentage*

The table of Euclidean distance from the Figure 5-11 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Average Pixel Percentage** |
| Image 1 | 0 |
| Image 2 | 0.155443 |
| Image 3 | 0.237338 |
| Image 4 | 0.353418 |
| Image 5 | 0.556879 |
| Image 6 | 0.593284 |
| Image 7 | 0.594542 |
| Image 8 | 0.598393 |
| Image 9 | 0.629299 |
| Image 10 | 0.662949 |

*Table 5-11 Euclidean distance table of Figure 5-11*

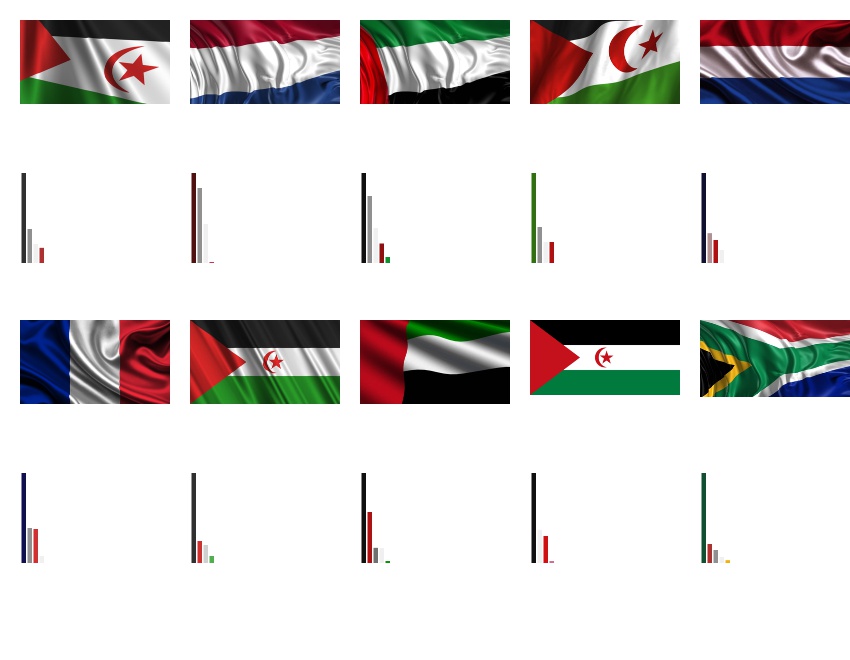


*Figure 5-12 Colour Region Moments*

The table of Euclidean distance from the Figure 5-12 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Colour Region Moments** |
| Image 1 | 0 |
| Image 2 | 0.307516 |
| Image 3 | 0.35199 |
| Image 4 | 0.40931 |
| Image 5 | 0.65204 |
| Image 6 | 0.968148 |
| Image 7 | 0.987407 |
| Image 8 | 1.74675 |
| Image 9 | 1.88758 |
| Image 10 | 2.85913 |

*Table 5-12 Euclidean distance table of Figure 5-12*

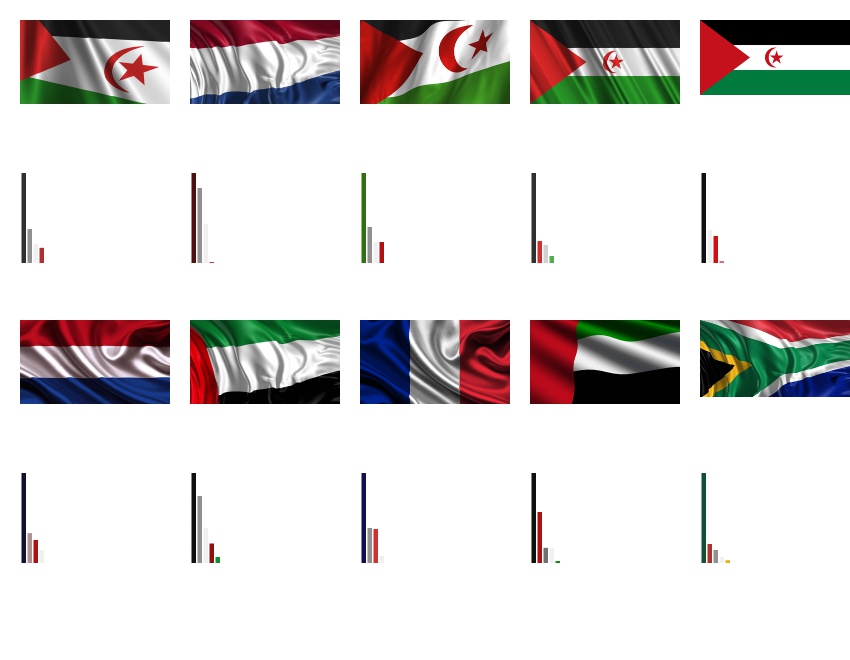


*Figure 5-13 Average Pixel Percentage*

The table of Euclidean distance from the Figure 5-13 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Average Pixel Percentage** |
| Image 1 | 0 |
| Image 2 | 0.163495 |
| Image 3 | 0.211377 |
| Image 4 | 0.334284 |
| Image 5 | 0.531957 |
| Image 6 | 0.562168 |
| Image 7 | 0.572436 |
| Image 8 | 0.578346 |
| Image 9 | 0.593284 |
| Image 10 | 0.83062 |

*Table 5-13 Euclidean distance table of Figure 5-13*



*Figure 5-14 Colour Region Moments*

The table of Euclidean distance from the Figure 5-14 will be listed below.

|  |  |
| --- | --- |
| **Retrieval Result** | **Euclidean Distance of Colour Region Moments** |
| Image 1 | 0 |
| Image 2 | 0.717973 |
| Image 3 | 1.47679 |
| Image 4 | 1.90811 |
| Image 5 | 1.9568 |
| Image 6 | 1.99627 |
| Image 7 | 2.12701 |
| Image 8 | 2.21718 |
| Image 9 | 2.22021 |
| Image 10 | 2.5191 |

*Table 5-14 Euclidean distance table of Figure 5-14*

### 5.2 Shape Descriptor Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **Euclidean Distance** | 0 | 0 | 0 |
| **Retrieval Result 2** |  |  |  |
| **Euclidean Distance** | 0.000458483 | 0.000592815 | 0.000048022 |
| **Retrieval Result 3** |  |  |  |
| **Euclidean Distance** | 0.000555066 | 0.00060851 | 0.0000502063 |
| **Retrieval Result 4** |  |  |  |
| **Euclidean Distance** | 0.000561136 | 0.000636909 | 0.000202711 |
| **Retrieval Result 5** |  |  |  |
| **Euclidean Distance** | 0.000590953 | 0.000650105 | 0.000342137 |
| **Retrieval Result 6** |  |  |  |
| **Euclidean Distance** | 0.000605487 | 0.000654315 | 0.000590953 |
| **Retrieval Result 7** |  |  |  |
| **Euclidean Distance** | 0.00078834 | 0.000956662 | 0.000636909 |

*Table 5-15 Comparison table of central normalized moments among different kinds of circle-based shape*

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **Euclidean Distance** | 0 | 0 | 0 |
| **Retrieval Result 2** |  |  |  |
| **Euclidean Distance** | 0.000000372195 | 0.0000103414 | 0.00000000730453 |
| **Retrieval Result 3** |  |  |  |
| **Euclidean Distance** | 0.00000054647 | 0.0000107136 | 0.0000000685894 |
| **Retrieval Result 4** |  |  |  |
| **Euclidean Distance** | 0.000000607755 | 0.0000108879 | 0.000000242864 |
| **Retrieval Result 5** |  |  |  |
| **Euclidean Distance** | 0.000000615059 | 0.0000109492 | 0.000000615059 |
| **Retrieval Result 6** |  |  |  |
| **Euclidean Distance** | 0.000000960546 | 0.0000109565 | 0.0000109565 |
| **Retrieval Result 7** |  |  |  |
| **Euclidean Distance** | 0.0000960546 | 0.0000857132 | 0.000096697 |

*Table 5-16 Comparison table of Hu moments among different kinds of circle-based shape*

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **Euclidean Distance** | 0 | 0 | 0 |
| **Retrieval Result 2** |  |  |  |
| **Euclidean Distance** | 0.00103703 | 0.0170125 | 0.00717931 |
| **Retrieval Result 3** |  |  |  |
| **Euclidean Distance** | 0.0010817 | 0.0246436 | 0.0132721 |
| **Retrieval Result 4** |  |  |  |
| **Euclidean Distance** | 0.0111427 | 0.0259356 | 0.0155178 |
| **Retrieval Result 5** |  |  |  |
| **Euclidean Distance** | 0.0198065 | 0.0359782 | 0.0185495 |
| **Retrieval Result 6** |  |  |  |
| **Euclidean Distance** | 0.0200719 | 0.0370574 | 0.0212763 |
| **Retrieval Result 7** |  |  |  |
| **Euclidean Distance** | 0.0370574 | 0.0380915 | 0.0219429 |

*Table 5-17 Comparison table of central normalized moments among different kinds of ellipse-based shape*

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **Euclidean Distance** | 0 | 0 | 0 |
| **Retrieval Result 2** |  |  |  |
| **Euclidean Distance** | 0.000169693 | 0.0000462822 | 0.0000462822 |
| **Retrieval Result 3** |  |  |  |
| **Euclidean Distance** | 0.000176552 | 0.000761346 | 0.000807629 |
| **Retrieval Result 4** |  |  |  |
| **Euclidean Distance** | 0.00179198 | 0.000862783 | 0.000909065 |
| **Retrieval Result 5** |  |  |  |
| **Euclidean Distance** | 0.00317238 | 0.000895369 | 0.000941651 |
| **Retrieval Result 6** |  |  |  |
| **Euclidean Distance** | 0.0037877 | 0.00108422 | 0.000113051 |
| **Retrieval Result 7** |  |  |  |
| **Euclidean Distance** | 0.00514371 | 0.00110217 | 0.00114845 |

*Table 5-18 Comparison table of Hu moments among different kinds of ellipse-based shape*

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **Euclidean Distance** | 0 | 0 | 0 |
| **Retrieval Result 2** |  |  |  |
| **Euclidean Distance** | 0.000403262 | 0.0027767 | 0.00200202 |
| **Retrieval Result 3** |  |  |  |
| **Euclidean Distance** | 0.00476533 | 0.00351503 | 0.00403262 |
| **Retrieval Result 4** |  |  |  |
| **Euclidean Distance** | 0.00573411 | 0.00739547 | 0.00947031 |
| **Retrieval Result 5** |  |  |  |
| **Euclidean Distance** | 0.00980098 | 0.00757534 | 0.0128797 |
| **Retrieval Result 6** |  |  |  |
| **Euclidean Distance** | 0.0101147 | 0.0120389 | 0.013378 |
| **Retrieval Result 7** |  |  |  |
| **Euclidean Distance** | 0.012745 | 0.012745 | 0.0161086 |

*Table 5-19 Comparison table of central normalized moments among different kinds of shield-based shape*

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **Euclidean Distance** | 0 | 0 | 0 |
| **Retrieval Result 2** |  |  |  |
| **Euclidean Distance** | 0.0000125557 | 0.0000325865 | 0.0000125377 |
| **Retrieval Result 3** |  |  |  |
| **Euclidean Distance** | 0.0000250934 | 0.000134023 | 0.0000125557 |
| **Retrieval Result 4** |  |  |  |
| **Euclidean Distance** | 0.000167527 | 0.000188856 | 0.000154971 |
| **Retrieval Result 5** |  |  |  |
| **Euclidean Distance** | 0.000178652 | 0.000206798 | 0.000166096 |
| **Retrieval Result 6** |  |  |  |
| **Euclidean Distance** | 0.000465061 | 0.000671859 | 0.000477616 |
| **Retrieval Result 7** |  |  |  |
| **Euclidean Distance** | 0.000483003 | 0.000684415 | 0.000495559 |

*Table 5-20 Comparison table of Hu moments among different kinds of shield-based shape*

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **Euclidean Distance** | 0 | 0 | 0 |
| **Retrieval Result 2** |  |  |  |
| **Euclidean Distance** | 0.000399203 | 0.000458483 | 0.00317987 |
| **Retrieval Result 3** |  |  |  |
| **Euclidean Distance** | 0.0142113 | 0.000903516 | 0.00563677 |
| **Retrieval Result 4** |  |  |  |
| **Euclidean Distance** | 0.0357915 | 0.000959295 | 0.147035 |
| **Retrieval Result 5** |  |  |  |
| **Euclidean Distance** | 0.0417874 | 0.000965096 | 0.151494 |
| **Retrieval Result 6** |  |  |  |
| **Euclidean Distance** | 0.0427979 | 0.000996638 | 0.151512 |
| **Retrieval Result 7** |  |  |  |
| **Euclidean Distance** | 0.043852 | 0.00118699 | 0.151658 |

*Table 5-21 Comparison table of central normalized moments among different kinds of other-based shape*

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **Euclidean Distance** | 0 | 0 | 0 |
| **Retrieval Result 2** |  |  |  |
| **Euclidean Distance** | 0.00000820939 | 0.00000498457 | 0.000543147 |
| **Retrieval Result 3** |  |  |  |
| **Euclidean Distance** | 0.000865334 | 0.0000857132 | 0.000846585 |
| **Retrieval Result 4** |  |  |  |
| **Euclidean Distance** | 0.00298207 | 0.0000960546 | 0.0189623 |
| **Retrieval Result 5** |  |  |  |
| **Euclidean Distance** | 0.00795609 | 0.0000964268 | 0.0198194 |
| **Retrieval Result 6** |  |  |  |
| **Euclidean Distance** | 0.00812578 | 0.0000966011 | 0.0198276 |
| **Retrieval Result 7** |  |  |  |
| **Euclidean Distance** | 0.00830234 | 0.0000966624 | 0.0228097 |

*Table 5-22 Comparison table of Hu moments among different kinds of other-based shape*

### 5.3 Combined Descriptor Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **ED of Shape** | 0 | 0 | 0.000399203 |
| **ED of Colour** | 0 | 0 | 0.0173271 |
| **Retrieval Result 2** |  |  |  |
| **ED of Shape** | 0 | 0.0027767 | 0.0357915 |
| **ED of Colour** | 0.111018 | 0.0850423 | 0.0215484 |
| **Retrieval Result 3** |  |  |  |
| **ED of Shape** | 0.0101147 | 0.00947031 | 0 |
| **ED of Colour** | 0.165398 | 0.315032 | 0.176673 |
| **Retrieval Result 4** |  |  |  |
| **ED of Shape** | 0.012745 | 0.0161086 | 0.0417874 |
| **ED of Colour** | 0.228845 | 0.317096 | 0.405598 |
| **Retrieval Result 5** |  |  |  |
| **ED of Shape** | 0.00573411 | 0.013378 | 0.0427979 |
| **ED of Colour** | 0.392138 | 0.365425 | 0.620698 |
| **Retrieval Result 6** |  |  |  |
| **ED of Shape** | 0.00476533 | 0.00403262 | 0.043852 |
| **ED of Colour** | 0.494078 | 0.402486 | 0.622039 |
| **Retrieval Result 7** |  |  |  |
| **ED of Shape** | 0.00403262 | 0.0128797 | 0.0142113 |
| **ED of Colour** | 0.496465 | 0.436304 | 0.64729 |

*Table 5-23 Comparison table 1 of combined descriptor*

|  |  |  |  |
| --- | --- | --- | --- |
| **Query Image** |  |  |  |
| **Retrieval Result 1** |  |  |  |
| **ED of Shape** | 0 | 0 | 0 |
| **ED of Colour** | 0 | 0 | 0 |
| **Retrieval Result 2** |  |  |  |
| **ED of Shape** | 0.0010817 | 0.000590953 | 0.000592816 |
| **ED of Colour** | 0.1559 | 0.00044125 | 0.00112987 |
| **Retrieval Result 3** |  |  |  |
| **ED of Shape** | 0.0370574 | 0.000525941 | 0.000616415 |
| **ED of Colour** | 0.285067 | 0.000614414 | 0.00126312 |
| **Retrieval Result 4** |  |  |  |
| **ED of Shape** | 0.0200719 | 0.000406289 | 0.00060851 |
| **ED of Colour** | 0.405353 | 0.000668195 | 0.0012935 |
| **Retrieval Result 5** |  |  |  |
| **ED of Shape** | 0.00103703 | 0.000399838 | 0.000650105 |
| **ED of Colour** | 0.413718 | 0.000675727 | 0.00132767 |
| **Retrieval Result 6** |  |  |  |
| **ED of Shape** | 0.0198065 | 0.000555066 | 0.000956662 |
| **ED of Colour** | 0.548422 | 0.00071192 | 0.00133042 |
| **Retrieval Result 7** |  |  |  |
| **ED of Shape** | 0.0111427 | 0.00096329 | 0.000730399 |
| **ED of Colour** | 0.769603 | 0.000871459 | 0.00136105 |

*Table 5-24 Comparison table 2 of combined descriptor*

Conclusion, as we can observe from those tables above, we discover that corresponding colour pixel percentage and central normalized moments will be more stable and return more precise results. This is because corresponding colour pixel percentage compares the colour pixel percentage with same or similar colour value. Moreover, central normalized moments can be applied in any shape, however Hu moments just be useful for some restricted shapes, for example, Hu moments will return the wrong results if the database consists of horizontal ellipse-based shape, vertical ellipse-based shape, and shield-based shape of images. Thus, central normalized moments will be more reliable if the database consists of any kind of shape.

# CHAPTER 6 CONCLUSIONS

## 6.1 Strengths and Weaknesses

Strengths:

1. The retrieval rate will be more precise because we have 2 stages of sorting.
2. Colour descriptor can work on images with shadowed, wiggled, different scaling, and different colour brightness.
3. Shape descriptor can work on any type of shape.
4. The retrieval speed is fast.
5. Can compute image with a plain background.

Weaknesses:

1. The accuracy of national flags for colour descriptor still can be improved.
2. Can only work in constrained database.
3. The background needed to be plain colour, i.e. either white, black, red, etc.

## 6.2 Preliminary Work

For colour descriptor, we have partitioned the image into 2 parts which are left and right. In the future, we can further portioned our image into 4 parts which are top left, top right, bottom left, and bottom left, to improve the accuracy rate.

Besides, we can try to implement other colour spaces like HSV and L\*a\*b\* which have more advantages if compared to RGB colour space in content based image retrieval.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

*Figure 6-1 L\*a\*b\* colour matrixes and dominant colours histogram*

# Bibliography

1. Anandh, K. Mala, and S. Suganya, “Content based image retrieval system based on semantic information using color, texture and shape features”, IEEE, pp. 1-8, 7-9 Jan. 2016.
2. Deepti Agrawal, Anand Singh Jalal, and Rajesh Tripathi, “Trademark image retrieval by integrating shape with texture feature”, IEEE, pp. 30-33, 9-10 March 2013.
3. Yong-jiao Wang and Chun-feng Zheng, “Trademark Image Retrieval Based on Shape and Key Local Color Features”, IEEE, vol. 2, pp. 325-328, 21-22 May 2009.
4. Raymond Phan and Dimitrios Androutsos, “Content-based retrieval of logo and trademarks in unconstrained color image databases using Color Edge Gradient Co-occurrence Histograms”, Computer Vision and Image Understanding, vol. 114, pp. 66-84, 1 January 2010.
5. "OpenCV Tutorials — OpenCV 3.0.0-dev documentation", *Docs.opencv.org*, 2017. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/tutorials/tutorials.html. [Accessed: 03- Sep- 2017].
6. *The OpenCV Reference Manual Release 3.0.0-dev*. 2014.

# Appendix

Header.h

#define LOGO\_TRAINING\_IMAGES 61

#define FLAG\_TRAINING\_IMAGES 76

#define SIZE\_OF\_ARRAY 100

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace cv;

using namespace std;

void displayQueryImage(Mat &image);

int calcColourBin(Vec3b colour);

Vec3b convert2Bgr(int colourBin);

void roi(Mat &image, Mat &roi\_left, Mat &roi\_right);

int sortDominantColour(vector<int> &colourVote, vector<pair<int, int>> &bin\_sort, int &max\_fre);

int wholeImage(Mat &image, Mat &resultImage, int \*topRankColour, int \*pixelPercentage, Mat &histogram);

void centralNormalizedMoment(Mat &binaryImage, double \*nuMoment);

double euclideanDistance\_colourRegion(Mat &image, Mat &trainImage, int \*colour\_query, int \*colour\_dataset, int totalColour\_img, int totalColour\_timg);

void findContour(Mat &image, Mat &resultImage, double \*nuMoment);

int eliminateBackground(Mat &image, Mat &processedImage, int \*topRankColour, int \*pixelPercentage, Mat &histogram);

double euclideanDistance\_nuMoment(double \*param\_query, double \*param\_dataset, int arrSize);

double euclideanDistance\_pixelPercentage(int \*colour\_query, int \*colour\_dataset, int \*pp\_query, int \*pp\_dataset, int arrSize\_query, int arrSize\_dataset, int imgSize\_query,

int imgSize\_dataset);

void sort2DArray(double arr[][2], int row);

Mat attachResults2AWhiteBgWithHist(double arr[][2], vector<Mat> &trainImage, vector<Mat> &histogram, int number);

Mat attachResults2AWhiteBg(double arr[][2], vector<Mat> &trainImage, int number);

ImageProcessing.cpp

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

#include <cmath>

#include "Header.h"

using namespace cv;

using namespace std;

void displayQueryImage(Mat &image) {

cout << "Image Height:\t" << image.rows << endl;

cout << "Image Width :\t" << image.cols << endl;

namedWindow("Query Image", WINDOW\_AUTOSIZE);

imshow("Query Image", image);

}

int calcColourBin(Vec3b colour) {

int colourBin; //colour range from 0 - 511

int b, g, r; //bgr range from 0 - 7

b = colour[0] / 32;

g = colour[1] / 32;

r = colour[2] / 32;

colourBin = b \* pow(8, 2) + g \* 8 + r;

return colourBin;

}

Vec3b convert2Bgr(int colourBin) {

int b, g, r;

b = colourBin / 64 \* 32 + 16; //16 is center intensity value of colour bin

g = (colourBin / 8) % 8 \* 32 + 16;

r = colourBin % 8 \* 32 + 16;

return Vec3b(b, g, r);

}

void roi(Mat &image, Mat &roi\_left, Mat &roi\_right) {

Rect left(0, 0, image.cols / 2, image.rows); //symmetry left part of the image

Rect right(left.width, 0, image.cols - left.width, image.rows); //symmetry right part of the image

roi\_left = image(left);

roi\_right = image(right);

//display images

/\*namedWindow("Left Part", WINDOW\_AUTOSIZE);

imshow("Left Part", roi\_left);

imwrite("Left Part.jpg", roi\_left);

namedWindow("Right Part", WINDOW\_AUTOSIZE);

imshow("Right Part", roi\_right);

imwrite("Right Part.jpg", roi\_right);\*/

}

int sortDominantColour(vector<int> &colourVote, vector<pair<int, int>> &bin\_sort, int &max\_fre) {

int totalColour = 0;

for (int i = 0; i < 512; i++) {

if (colourVote[i] > 0) {

bin\_sort.push\_back(pair<int, int>(colourVote[i], i));

max\_fre = max(max\_fre, colourVote[i]);

totalColour++;

}

}

sort(bin\_sort.rbegin(), bin\_sort.rend()); //sort the quantized bin vector according to its vote

return totalColour;

}

int wholeImage(Mat &image, Mat &resultImage, int \*topRankColour, int \*pixelPercentage, Mat &histogram) {

//colour quantization

Mat quanImage(image.size(), image.type());

vector<int> colourVote(512);

//for each pixel, compute the 8 bins value and store into vector

for (int r = 0; r < image.rows; r++){

for (int c = 0; c < image.cols; c++){

int colourValue = calcColourBin(image.at<Vec3b>(r, c)); //quantized colour bin

colourVote[colourValue]++; //calculate vote of each colour

//draw quantized image

for (int channel = 0; channel < 3; channel++)

quanImage.at<Vec3b>(r, c)[channel] = convert2Bgr(colourValue)[channel];

}

}

/\*namedWindow("Quantized Image", WINDOW\_AUTOSIZE);

imshow("Quantized Image", quanImage);

imwrite("Quantized Image.jpg", quanImage);\*/

//sort quantized dominant colours

vector<pair<int, int>> bin\_sort; //a vector to store the quantized bin and sort it

int max\_fre = 0;

int totalColour; //to store the total number of contained colours

totalColour = sortDominantColour(colourVote, bin\_sort, max\_fre);

//draw the histogram in canvas

Mat canvas(620, 1000, CV\_8UC3, Scalar(255, 255, 255));

for (int i = 0; i < totalColour; i++)

rectangle(canvas, Point((30 + 10) \* i + 10, 620), Point((30 + 10) \* i + 40, 620 - bin\_sort[i].first \* 600 / max\_fre), convert2Bgr(bin\_sort[i].second), -1);

/\*namedWindow("Sorted Quantized Dominant Colours Histogram", WINDOW\_AUTOSIZE);

imshow("Sorted Quantized Dominant Colours Histogram", canvas);

imwrite("Sorted Quantized Dominant Colours Histogram.jpg", canvas);\*/

//merge similar colour pixel

int threshold = 32, colourIndex = 0;

Mat mergedColourImage(image.size(), image.type());

//convert similar colour into same colour within bin\_sort vector according to colour rank

while (colourIndex < totalColour) {

for (int cnt = 0; cnt < totalColour; cnt++) {

if (abs(convert2Bgr(bin\_sort[colourIndex].second)[0] - convert2Bgr(bin\_sort[cnt].second)[0]) <= threshold \* 3 &&

abs(convert2Bgr(bin\_sort[colourIndex].second)[1] - convert2Bgr(bin\_sort[cnt].second)[1]) <= threshold \* 3 &&

abs(convert2Bgr(bin\_sort[colourIndex].second)[2] - convert2Bgr(bin\_sort[cnt].second)[2]) <= threshold \* 3 &&

(abs(convert2Bgr(bin\_sort[colourIndex].second)[0] - convert2Bgr(bin\_sort[cnt].second)[0]) +

abs(convert2Bgr(bin\_sort[colourIndex].second)[1] - convert2Bgr(bin\_sort[cnt].second)[1]) +

abs(convert2Bgr(bin\_sort[colourIndex].second)[2] - convert2Bgr(bin\_sort[cnt].second)[2])) <= threshold \* 6)

bin\_sort[cnt].second = bin\_sort[colourIndex].second;

}

colourIndex++;

}

colourIndex = 0; //reset flag

//draw merged colour image

while (colourIndex < totalColour) {

for (int r = 0; r < image.rows; r++) {

for (int c = 0; c < image.cols; c++) {

if (abs(convert2Bgr(bin\_sort[colourIndex].second)[0] - quanImage.at<Vec3b>(r, c)[0]) <= threshold \* 3 &&

abs(convert2Bgr(bin\_sort[colourIndex].second)[1] - quanImage.at<Vec3b>(r, c)[1]) <= threshold \* 3 &&

abs(convert2Bgr(bin\_sort[colourIndex].second)[2] - quanImage.at<Vec3b>(r, c)[2]) <= threshold \* 3 &&

(abs(convert2Bgr(bin\_sort[colourIndex].second)[0] - quanImage.at<Vec3b>(r, c)[0]) +

abs(convert2Bgr(bin\_sort[colourIndex].second)[1] - quanImage.at<Vec3b>(r, c)[1]) +

abs(convert2Bgr(bin\_sort[colourIndex].second)[2] - quanImage.at<Vec3b>(r, c)[2])) <= threshold \* 6) {

for (int channel = 0; channel < 3; channel++)

mergedColourImage.at<Vec3b>(r, c)[channel] = convert2Bgr(bin\_sort[colourIndex].second)[channel];

}

}

}

colourIndex++;

}

/\*namedWindow("Merged Colour Image", WINDOW\_AUTOSIZE);

imshow("Merged Colour Image", mergedColourImage);

imwrite("Merged Colour Image.jpg", mergedColourImage);\*/

resultImage = mergedColourImage;

vector<int> merge\_colourVote(512);

//for each pixel, compute the 8 bins value and store into vector

for (int r = 0; r < mergedColourImage.rows; r++){

for (int c = 0; c < mergedColourImage.cols; c++){

int colourValue = calcColourBin(mergedColourImage.at<Vec3b>(r, c));

merge\_colourVote[colourValue]++; //calculate vote of each colour

}

}

vector<pair<int, int>> merge\_bin\_sort; //a vector to store the quantized bin and sort it

int merge\_max\_fre = 0;

int merge\_totalColour; //to store the total number of contained colours

merge\_totalColour = sortDominantColour(merge\_colourVote, merge\_bin\_sort, merge\_max\_fre);

//draw the histogram in merge\_canvas

Mat merge\_canvas(620, 1000, CV\_8UC3, Scalar(255, 255, 255));

for (int i = 0; i < merge\_totalColour; i++)

rectangle(merge\_canvas, Point((30 + 10) \* i + 10, 620), Point((30 + 10) \* i + 40, 620 - merge\_bin\_sort[i].first \* 600 / merge\_max\_fre),

convert2Bgr(merge\_bin\_sort[i].second), -1);

/\*namedWindow("Dominant Colours Histogram After Merging", WINDOW\_AUTOSIZE);

imshow("Dominant Colours Histogram After Merging", merge\_canvas);

imwrite("Dominant Colours Histogram After Merging.jpg", merge\_canvas);\*/

histogram = merge\_canvas;

//store the dominant colours

int domColour\_no = 1; //number of dominant colour after neglecting noices

\*topRankColour = merge\_bin\_sort[0].second;

\*pixelPercentage = merge\_bin\_sort[0].first;

for (int i = 1; i < merge\_totalColour; i++) {

if (merge\_bin\_sort[i].first > merge\_bin\_sort[i - 1].first \* 0.1) {

\*(topRankColour + i) = merge\_bin\_sort[i].second;

\*(pixelPercentage + i) = merge\_bin\_sort[i].first;

domColour\_no++;

}

else

break;

}

return domColour\_no;

}

void centralNormalizedMoment(Mat &binaryImage, double \*nuMoment) {

Moments moment\_output;

moment\_output = moments(binaryImage, true); //white foreground region

nuMoment[0] = moment\_output.nu02;

nuMoment[1] = moment\_output.nu03;

nuMoment[2] = moment\_output.nu11;

nuMoment[3] = moment\_output.nu12;

nuMoment[4] = moment\_output.nu20;

nuMoment[5] = moment\_output.nu21;

nuMoment[6] = moment\_output.nu30;

}

double euclideanDistance\_colourRegion(Mat &image, Mat &trainImage, int \*colour\_query, int \*colour\_dataset, int totalColour\_query, int totalColour\_dataset) {

vector<Mat> singleColour\_query, singleColour\_dataset; //binary images of each colour

ostringstream oss;

string binaryWindowName;

for (int n = 0; n < totalColour\_query; n++) {

Mat imBw = Mat::zeros(image.size(), CV\_8UC1);

for (int r = 0; r < image.rows; r++) {

for (int c = 0; c < image.cols; c++) {

if (calcColourBin(image.at<Vec3b>(r, c)) == \*(colour\_query + n))

imBw.at<uchar>(r, c) = uchar(255);

}

}

singleColour\_query.push\_back(imBw);

//display binary images

//oss << "Query Binary Image " << n + 1;

//binaryWindowName = oss.str();

//namedWindow(binaryWindowName, WINDOW\_AUTOSIZE);

//imshow(binaryWindowName, imBw);

//imwrite(binaryWindowName + ".jpg", imBw);

//oss.str(""); //clear the string

//oss.clear(); //clear any error flags

}

for (int n = 0; n < totalColour\_dataset; n++) {

Mat imBw = Mat::zeros(trainImage.size(), CV\_8UC1);

for (int r = 0; r < trainImage.rows; r++) {

for (int c = 0; c < trainImage.cols; c++) {

if (calcColourBin(trainImage.at<Vec3b>(r, c)) == \*(colour\_dataset + n))

imBw.at<uchar>(r, c) = uchar(255);

}

}

singleColour\_dataset.push\_back(imBw);

//display binary images

//oss << "Dataset Binary Image " << n + 1;

//binaryWindowName = oss.str();

//namedWindow(binaryWindowName, WINDOW\_AUTOSIZE);

//imshow(binaryWindowName, imBw);

//imwrite(binaryWindowName + ".jpg", imBw);

//oss.str(""); //clear the string

//oss.clear(); //clear any error flags

}

double nuMoment\_query[7], nuMoment\_dataset[7];

double total = 0.0, ed;

for (int i = 0; i < totalColour\_query; i++) { //check query

int notMatched = 0;

for (int j = 0; j < totalColour\_dataset; j++) {

if (abs(convert2Bgr(\*(colour\_query + i))[0] / 32 - convert2Bgr(\*(colour\_dataset + j))[0] / 32) <= 2 &&

abs(convert2Bgr(\*(colour\_query + i))[1] / 32 - convert2Bgr(\*(colour\_dataset + j))[1] / 32) <= 2 &&

abs(convert2Bgr(\*(colour\_query + i))[2] / 32 - convert2Bgr(\*(colour\_dataset + j))[2] / 32) <= 2 &&

(abs(convert2Bgr(\*(colour\_query + i))[0] / 32 - convert2Bgr(\*(colour\_dataset + j))[0] / 32) +

abs(convert2Bgr(\*(colour\_query + i))[1] / 32 - convert2Bgr(\*(colour\_dataset + j))[1] / 32) +

abs(convert2Bgr(\*(colour\_query + i))[2] / 32 - convert2Bgr(\*(colour\_dataset + j))[2] / 32)) <= 3) {

centralNormalizedMoment(singleColour\_query[i], nuMoment\_query);

centralNormalizedMoment(singleColour\_dataset[j], nuMoment\_dataset);

for (int k = 0; k < 7; k++)

total += pow(\*(nuMoment\_query + k) - \*(nuMoment\_dataset + k), 2);

continue;

}

else

notMatched++;

if (notMatched == totalColour\_dataset) { //if no similar colour matched then minus 0

for (int k = 0; k < 7; k++)

total += pow(\*(nuMoment\_query + k) - 0, 2);

}

}

}

for (int i = 0; i < totalColour\_dataset; i++) { //check dataset

int notMatched = 0;

for (int j = 0; j < totalColour\_query; j++) {

if (abs(convert2Bgr(\*(colour\_dataset + i))[0] / 32 - convert2Bgr(\*(colour\_query + j))[0] / 32) > 2 ||

abs(convert2Bgr(\*(colour\_dataset + i))[1] / 32 - convert2Bgr(\*(colour\_query + j))[1] / 32) > 2 ||

abs(convert2Bgr(\*(colour\_dataset + i))[2] / 32 - convert2Bgr(\*(colour\_query + j))[2] / 32) > 2 ||

(abs(convert2Bgr(\*(colour\_dataset + i))[0] / 32 - convert2Bgr(\*(colour\_query + j))[0] / 32) +

abs(convert2Bgr(\*(colour\_dataset + i))[1] / 32 - convert2Bgr(\*(colour\_query + j))[1] / 32) +

abs(convert2Bgr(\*(colour\_dataset + i))[2] / 32 - convert2Bgr(\*(colour\_query + j))[2] / 32)) > 3)

notMatched++;

if (notMatched == totalColour\_query) { //if no similar colour matched then minus 0

for (int k = 0; k < 7; k++)

total += pow(\*(nuMoment\_dataset + k) - 0, 2);

}

}

}

ed = sqrt(total);

return ed;

}

void findContour(Mat &image, Mat &resultImage, double \*nuMoment) {

Mat imGray, imBw;

cvtColor(image, imGray, CV\_BGR2GRAY);

threshold(imGray, imBw, 220, 255, THRESH\_BINARY\_INV); //set values equal to or above 220 to 0, values below 220 to 255

// Floodfill from point (0, 0)

Mat im\_floodfill = imBw.clone();

floodFill(im\_floodfill, Point(0, 0), Scalar(255));

// Invert floodfilled image

Mat im\_floodfill\_inv;

bitwise\_not(im\_floodfill, im\_floodfill\_inv);

// Combine the two images to get the foreground.

Mat imOut = (imBw | im\_floodfill\_inv);

// Display images

/\*namedWindow("Step 1", WINDOW\_AUTOSIZE);

imshow("Step 1", imGray);

imwrite("Step 2.jpg", imGray);

namedWindow("Step 2", WINDOW\_AUTOSIZE);

imshow("Step 2", imBw);

imwrite("Step 2.jpg", imBw);

namedWindow("Step 3", WINDOW\_AUTOSIZE);

imshow("Step 3", im\_floodfill);

imwrite("Step 3.jpg", im\_floodfill);

namedWindow("Step 4", WINDOW\_AUTOSIZE);

imshow("Step 4", im\_floodfill\_inv);

imwrite("Step 4.jpg", im\_floodfill\_inv);

namedWindow("Foreground", WINDOW\_AUTOSIZE);

imshow("Foreground", imOut);

imwrite("Foreground.jpg", imOut);\*/

resultImage = imOut;

centralNormalizedMoment(imOut, nuMoment); //calculate the 7 central normalized moments

}

int eliminateBackground(Mat &image, Mat &processedImage, int \*topRankColour, int \*pixelPercentage, Mat &histogram) {

vector<int> colourVote(512);

for (int r = 0; r < image.rows; r++) {

for (int c = 0; c < image.cols; c++) {

if (processedImage.at<uchar>(r, c) == 255) {

int colourValue = calcColourBin(image.at<Vec3b>(r, c)); //quantized colour bin

colourVote[colourValue]++; //calculate vote of each colour

}

}

}

//sort quantized dominant colours

vector<pair<int, int>> bin\_sort; //a vector to store the quantized bin and sort it

int max\_fre = 0;

int totalColour; //to store the total number of contained colours

totalColour = sortDominantColour(colourVote, bin\_sort, max\_fre);

//draw the histogram in canvas

Mat canvas(620, 1000, CV\_8UC3, Scalar(255, 255, 255));

for (int i = 0; i < totalColour; i++)

rectangle(canvas, Point((30 + 10) \* i + 10, 620), Point((30 + 10) \* i + 40, 620 - bin\_sort[i].first \* 600 / max\_fre), convert2Bgr(bin\_sort[i].second), -1);

/\*namedWindow("Sorted Quantized Dominant Colours Histogram", WINDOW\_AUTOSIZE);

imshow("Sorted Quantized Dominant Colours Histogram", canvas);

imwrite("Sorted Quantized Dominant Colours Histogram.jpg", canvas);\*/

//merge similar colour pixel

int threshold = 32, colourIndex = 0;

//convert similar colour into same colour within bin\_sort vector according to colour rank

while (colourIndex < totalColour) {

for (int cnt = 0; cnt < totalColour; cnt++) {

if (abs(convert2Bgr(bin\_sort[colourIndex].second)[0] - convert2Bgr(bin\_sort[cnt].second)[0]) <= threshold \* 3 &&

abs(convert2Bgr(bin\_sort[colourIndex].second)[1] - convert2Bgr(bin\_sort[cnt].second)[1]) <= threshold \* 3 &&

abs(convert2Bgr(bin\_sort[colourIndex].second)[2] - convert2Bgr(bin\_sort[cnt].second)[2]) <= threshold \* 3 &&

(abs(convert2Bgr(bin\_sort[colourIndex].second)[0] - convert2Bgr(bin\_sort[cnt].second)[0]) +

abs(convert2Bgr(bin\_sort[colourIndex].second)[1] - convert2Bgr(bin\_sort[cnt].second)[1]) +

abs(convert2Bgr(bin\_sort[colourIndex].second)[2] - convert2Bgr(bin\_sort[cnt].second)[2])) <= threshold \* 6)

bin\_sort[cnt].second = bin\_sort[colourIndex].second;

}

colourIndex++;

}

vector<int> merge\_colourVote(512);

vector<pair<int, int>> merge\_bin\_sort;

int merge\_max\_fre = 0;

int merge\_totalColour;

for (int i = 0; i < totalColour; i++){

int colourValue = calcColourBin(convert2Bgr(bin\_sort[i].second));

merge\_colourVote[colourValue]++;

}

merge\_totalColour = sortDominantColour(merge\_colourVote, merge\_bin\_sort, merge\_max\_fre);

//draw the histogram in merge\_canvas

Mat merge\_canvas(620, 1000, CV\_8UC3, Scalar(255, 255, 255));

for (int i = 0; i < merge\_totalColour; i++)

rectangle(merge\_canvas, Point((30 + 10) \* i + 10, 620), Point((30 + 10) \* i + 40, 620 - merge\_bin\_sort[i].first \* 600 / merge\_max\_fre),

convert2Bgr(merge\_bin\_sort[i].second), -1);

/\*namedWindow("Dominant Colours Histogram After Merging", WINDOW\_AUTOSIZE);

imshow("Dominant Colours Histogram After Merging", merge\_canvas);

imwrite("Dominant Colours Histogram After Merging.jpg", merge\_canvas);\*/

histogram = merge\_canvas;

//store the dominant colours

int domColour\_no = 1; //number of dominant colour after neglecting noices

\*topRankColour = merge\_bin\_sort[0].second;

\*pixelPercentage = merge\_bin\_sort[0].first;

for (int i = 1; i < totalColour; i++) {

if (merge\_bin\_sort[i].first > merge\_bin\_sort[i - 1].first \* 0.1) {

\*(topRankColour + i) = merge\_bin\_sort[i].second;

\*(pixelPercentage + i) = merge\_bin\_sort[i].first;

domColour\_no++;

}

else

break;

}

return domColour\_no;

}

double euclideanDistance\_nuMoment(double \*param\_query, double \*param\_dataset, int arrSize) {

double total = 0.0, ed;

for (int i = 0; i < arrSize; i++)

total += pow(\*(param\_query + i) - \*(param\_dataset + i), 2);

ed = sqrt(total);

return ed;

}

double euclideanDistance\_pixelPercentage(int \*colour\_query, int \*colour\_dataset, int \*pp\_query, int \*pp\_dataset, int arrSize\_query, int arrSize\_dataset, int imgSize\_query,

int imgSize\_dataset) {

double total = 0.0, ed;

for (int i = 0; i < arrSize\_query; i++) { //check query

int notMatched = 0;

for (int j = 0; j < arrSize\_dataset; j++) {

if (abs(convert2Bgr(\*(colour\_query + i))[0] / 32 - convert2Bgr(\*(colour\_dataset + j))[0] / 32) <= 2 &&

abs(convert2Bgr(\*(colour\_query + i))[1] / 32 - convert2Bgr(\*(colour\_dataset + j))[1] / 32) <= 2 &&

abs(convert2Bgr(\*(colour\_query + i))[2] / 32 - convert2Bgr(\*(colour\_dataset + j))[2] / 32) <= 2 &&

(abs(convert2Bgr(\*(colour\_query + i))[0] / 32 - convert2Bgr(\*(colour\_dataset + j))[0] / 32) +

abs(convert2Bgr(\*(colour\_query + i))[1] / 32 - convert2Bgr(\*(colour\_dataset + j))[1] / 32) +

abs(convert2Bgr(\*(colour\_query + i))[2] / 32 - convert2Bgr(\*(colour\_dataset + j))[2] / 32)) <= 3) {

total += pow(\*(pp\_query + i) / double(imgSize\_query) - \*(pp\_dataset + j) / double(imgSize\_dataset), 2);

continue;

}

else

notMatched++;

if (notMatched == arrSize\_dataset) //if no similar colour matched then minus 0

total += pow(\*(pp\_query + i) / double(imgSize\_query) - 0, 2);

}

}

for (int i = 0; i < arrSize\_dataset; i++) { //check dataset

int notMatched = 0;

for (int j = 0; j < arrSize\_query; j++) {

if (abs(convert2Bgr(\*(colour\_dataset + i))[0] / 32 - convert2Bgr(\*(colour\_query + j))[0] / 32) > 2 ||

abs(convert2Bgr(\*(colour\_dataset + i))[1] / 32 - convert2Bgr(\*(colour\_query + j))[1] / 32) > 2 ||

abs(convert2Bgr(\*(colour\_dataset + i))[2] / 32 - convert2Bgr(\*(colour\_query + j))[2] / 32) > 2 ||

(abs(convert2Bgr(\*(colour\_dataset + i))[0] / 32 - convert2Bgr(\*(colour\_query + j))[0] / 32) +

abs(convert2Bgr(\*(colour\_dataset + i))[1] / 32 - convert2Bgr(\*(colour\_query + j))[1] / 32) +

abs(convert2Bgr(\*(colour\_dataset + i))[2] / 32 - convert2Bgr(\*(colour\_query + j))[2] / 32)) > 3)

notMatched++;

if (notMatched == arrSize\_query) //if no similar colour matched then minus 0

total += pow((\*(pp\_dataset + i) / double(imgSize\_dataset)) - 0, 2);

}

}

ed = sqrt(total);

return ed;

}

void sort2DArray(double arr[][2], int row) {

double tempValue;

for (int current = 0; current < row; current++) {

for (int back = current + 1; back < row; back++) {

if (arr[current][1] > arr[back][1]) {

for (int col = 0; col < 2; col++) {

//sort index and euclidean distance

tempValue = arr[current][col];

arr[current][col] = arr[back][col];

arr[back][col] = tempValue;

}

}

}

}

}

Mat attachResults2AWhiteBgWithHist(double arr[][2], vector<Mat> &trainImage, vector<Mat> &histogram, int number) {

int size = 150;

int x\_train, x\_hist, y\_train, y\_hist; //x is cols, y is rows

int w = 5; //Maximum number of images in a row

int h = 4; //Maximum number of images in a column

int max\_train, max\_hist;

double scale\_train, scale\_hist; //How much we have to resize the image

Mat displayImage(Size(100 + size \* w, 60 + size \* h), CV\_8UC3, Scalar(255, 255, 255));

Mat resizedImage;

for (int i = 0, m = 20, n = 20; i < 5; i++, m += (20 + size)){

//Find the width and height of the image

x\_train = trainImage[arr[i][0]].cols;

y\_train = trainImage[arr[i][0]].rows;

x\_hist = histogram[arr[i][0]].cols;

y\_hist = histogram[arr[i][0]].rows;

//Find whether height or width is greater in order to resize the image

max\_train = x\_train > y\_train ? x\_train : y\_train;

max\_hist = x\_hist > y\_hist ? x\_hist : y\_hist;

//Find the scaling factor to resize the image

scale\_train = double(max\_train) / size;

scale\_hist = double(max\_hist) / size;

//Align the images

if (i % w == 0 && m != 20) {

m = 20;

n += 20 + size;

}

Rect roi(m, n, (int)(x\_train / scale\_train), (int)(y\_train / scale\_train)); //Set the image roi to display the current image

resize(trainImage[arr[i][0]], resizedImage, Size(roi.width, roi.height)); //Resize the input image and copy the it to the large image

resizedImage.copyTo(displayImage(roi));

Rect roi2(m, n + size, (int)(x\_hist / scale\_hist), (int)(y\_hist / scale\_hist));

resize(histogram[arr[i][0]], resizedImage, Size(roi2.width, roi2.height));

resizedImage.copyTo(displayImage(roi2));

}

for (int i = 5, m = 20, n = 20; i < number; i++, m += (20 + size)){

//Find the width and height of the image

x\_train = trainImage[arr[i][0]].cols;

y\_train = trainImage[arr[i][0]].rows;

x\_hist = histogram[arr[i][0]].cols;

y\_hist = histogram[arr[i][0]].rows;

//Find whether height or width is greater in order to resize the image

max\_train = x\_train > y\_train ? x\_train : y\_train;

max\_hist = x\_hist > y\_hist ? x\_hist : y\_hist;

//Find the scaling factor to resize the image

scale\_train = double(max\_train) / size;

scale\_hist = double(max\_hist) / size;

//Align the images

if (i % w == 0 && m != 20) {

m = 20;

n += 20 + size;

}

Rect roi(m, n + size \* 2, (int)(x\_train / scale\_train), (int)(y\_train / scale\_train)); //Set the image roi to display the current image

resize(trainImage[arr[i][0]], resizedImage, Size(roi.width, roi.height)); //Resize the input image and copy the it to the large image

resizedImage.copyTo(displayImage(roi));

Rect roi2(m, n + size \* 3, (int)(x\_hist / scale\_hist), (int)(y\_hist / scale\_hist));

resize(histogram[arr[i][0]], resizedImage, Size(roi2.width, roi2.height));

resizedImage.copyTo(displayImage(roi2));

}

return displayImage;

}

Mat attachResults2AWhiteBg(double arr[][2], vector<Mat> &trainImage, int number) {

int size = 150;

int x, y; //x is cols, y is rows

int w = 5; //Maximum number of images in a row

int h = 2; //Maximum number of images in a column

int max;

double scale; //How much we have to resize the image

Mat displayImage(Size(100 + size \* w, 60 + size \* h), CV\_8UC3, Scalar(255, 255, 255));

Mat resizedImage;

for (int i = 0, m = 20, n = 20; i < number; i++, m += (20 + size)){

//Find the width and height of the image

x = trainImage[arr[i][0]].cols;

y = trainImage[arr[i][0]].rows;

//Find whether height or width is greater in order to resize the image

max = x > y ? x : y;

//Find the scaling factor to resize the image

scale = double(max) / size;

//Align the images

if (i % w == 0 && m != 20) {

m = 20;

n += 20 + size;

}

Rect roi(m, n, (int)(x / scale), (int)(y / scale)); //Set the image roi to display the current image

resize(trainImage[arr[i][0]], resizedImage, Size(roi.width, roi.height)); //Resize the input image and copy the it to the large image

resizedImage.copyTo(displayImage(roi));

}

return displayImage;

}

Source.cpp

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

#include "Header.h"

using namespace cv;

using namespace std;

int main(void) {

char queryName[20];

string option;

bool terminate = false;

int arrSize; //array size to be past into function

//loading, processing, and displaying images

Mat image, processedImage, roi\_left\_img, roi\_right\_img, roi\_left\_timg, roi\_right\_timg, displayResult;

vector<String> fn\_logo, fn\_flag; //image file name

vector<Mat> logo\_trainImage, logo\_processedTrainImage, flag\_trainImage, flag\_processedTrainImage; //training image

//dominant colours histogram

Mat tempImage;

Mat canvas(620, 1000, CV\_8UC3, Scalar(255, 255, 255)); //large image for display purpose

vector<Mat> histogram;

for (int histSize = 0; histSize < SIZE\_OF\_ARRAY; histSize++)

histogram.push\_back(canvas);

//dominant colour

int topRankColour\_img[SIZE\_OF\_ARRAY], topRankColour\_img\_left[SIZE\_OF\_ARRAY], topRankColour\_img\_right[SIZE\_OF\_ARRAY];

int topRankColour\_timg[SIZE\_OF\_ARRAY], topRankColour\_timg\_left[SIZE\_OF\_ARRAY], topRankColour\_timg\_right[SIZE\_OF\_ARRAY];

int totalDominantColour\_img, totalDominantColour\_timg;

int totalDominantColour\_roi\_img[2], totalDominantColour\_roi\_timg[2]; //0 be left, 1 be right

//pixel percentage

int pixelPercentage\_img[SIZE\_OF\_ARRAY], pixelPercentage\_img\_left[SIZE\_OF\_ARRAY], pixelPercentage\_img\_right[SIZE\_OF\_ARRAY]; //height of the colour histogram

int pixelPercentage\_timg[SIZE\_OF\_ARRAY], pixelPercentage\_timg\_left[SIZE\_OF\_ARRAY], pixelPercentage\_timg\_right[SIZE\_OF\_ARRAY];

//euclidean distance

double ed\_nuMoment[SIZE\_OF\_ARRAY][2], nuMoment\_img[7], nuMoment\_timg[7], index\_huMoment[SIZE\_OF\_ARRAY][2]; //arrays to store moment

double ed\_pixelPercentage[SIZE\_OF\_ARRAY][2], ed\_pixelPercentage\_left[SIZE\_OF\_ARRAY][2], ed\_pixelPercentage\_right[SIZE\_OF\_ARRAY][2]; //arrarys to store pixel percentage

double ed\_colourRegion[SIZE\_OF\_ARRAY][2];

double partitionFiltering[SIZE\_OF\_ARRAY][2]; //average ed between left ed and right ed

while (cin) {

//compute training images

logo\_processedTrainImage.clear(); //make sure the memory is empty

flag\_processedTrainImage.clear();

glob("Logo\_Datasets/\*.jpg", fn\_logo, false); //retrieve logo training image

glob("Flag\_Datasets/\*.jpg", fn\_flag, false); //retrieve flag training image

for (int i = 0; i < LOGO\_TRAINING\_IMAGES; i++)

logo\_trainImage.push\_back(imread(fn\_logo[i], IMREAD\_COLOR));

for (int j = 0; j < LOGO\_TRAINING\_IMAGES; j++)

logo\_processedTrainImage.push\_back(logo\_trainImage[j]);

for (int i = 0; i < FLAG\_TRAINING\_IMAGES; i++)

flag\_trainImage.push\_back(imread(fn\_flag[i], IMREAD\_COLOR));

for (int j = 0; j < FLAG\_TRAINING\_IMAGES; j++)

flag\_processedTrainImage.push\_back(flag\_trainImage[j]);

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\*\*\*\*\* Similar Image Retrieval System \*\*\*\*\*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

cout << "Please input query image: ";

cin.getline(queryName, 20);

image = imread(queryName, IMREAD\_COLOR);

processedImage = image;

if (image.empty()) {

cout << "No image found.\n";

system("pause");

system("cls");

}

else {

while (!terminate) {

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\*\*\*\*\* Similar Image Retrieval System \*\*\*\*\*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

cout << " (1)\tColour Descripor (Flag)\n";

cout << " (2)\tShape Descripor (Logo)\n";

cout << " (3)\tCombined Descriptor (Logo)\n";

cout << " (4)\tTry Another Query Image\n";

cout << " (5)\tExit\n\n";

cout << "Please enter your option: ";

getline(cin, option);

cout << endl;

if (option.length() != 1) {

cout << "Invalid input.\n";

system("pause");

}

else {

switch (option[0]) {

case '1':

displayQueryImage(image);

//first stage filtering

totalDominantColour\_img = wholeImage(image, processedImage, topRankColour\_img, pixelPercentage\_img, tempImage);

for (int imgIndex = 0; imgIndex < FLAG\_TRAINING\_IMAGES; imgIndex++) {

totalDominantColour\_timg = wholeImage(flag\_trainImage[imgIndex], flag\_processedTrainImage[imgIndex],

topRankColour\_timg, pixelPercentage\_timg, histogram[imgIndex]);

//similarity score computed by ed of dominant colour pixel percentage

ed\_pixelPercentage[imgIndex][0] = imgIndex;

ed\_pixelPercentage[imgIndex][1] = euclideanDistance\_pixelPercentage(topRankColour\_img, topRankColour\_timg, pixelPercentage\_img,

pixelPercentage\_timg, totalDominantColour\_img, totalDominantColour\_timg, image.rows \* image.cols,

flag\_trainImage[imgIndex].rows \* flag\_trainImage[imgIndex].cols);

}

sort2DArray(ed\_pixelPercentage, FLAG\_TRAINING\_IMAGES); //sort in ascending order

//display the top 10 rank images in term of similarity measurement on a white background

//for (int count = 0; count < 10; count++)

// cout << "ED of Pixel Percentage - " << fn\_flag[ed\_pixelPercentage[count][0]] << ": " <<ed\_pixelPercentage[count][1] << endl;

//displayResult = attachResults2AWhiteBgWithHist(ed\_pixelPercentage, flag\_trainImage, histogram, 10);

//namedWindow("Pixel Percentage", WINDOW\_AUTOSIZE);

//imshow("Pixel Percentage", displayResult);

//imwrite("Pixel Percentage.jpg", displayResult);

//second stage filtering

for (int imgIndex = 0; imgIndex < 10; imgIndex++) {

totalDominantColour\_timg = wholeImage(flag\_trainImage[ed\_pixelPercentage[imgIndex][0]],

flag\_processedTrainImage[ed\_pixelPercentage[imgIndex][0]], topRankColour\_timg, pixelPercentage\_timg, tempImage);

ed\_colourRegion[imgIndex][0] = ed\_pixelPercentage[imgIndex][0];

ed\_colourRegion[imgIndex][1] = euclideanDistance\_colourRegion(processedImage, flag\_processedTrainImage[ed\_pixelPercentage[imgIndex][0]],

topRankColour\_img, topRankColour\_timg, totalDominantColour\_img, totalDominantColour\_timg);

}

sort2DArray(ed\_colourRegion, 10);

//display the top 10 rank images in term of similarity measurement on a white background

for (int count = 0; count < 10; count++)

cout << "ED of Colour Region Moment - " << fn\_flag[ed\_colourRegion[count][0]] << ": " << ed\_colourRegion[count][1] << endl;

displayResult = attachResults2AWhiteBgWithHist(ed\_colourRegion, flag\_trainImage, histogram, 10);

namedWindow("Colour Region Moment", WINDOW\_AUTOSIZE);

imshow("Colour Region Moment", displayResult);

imwrite("Colour Region Moment.jpg", displayResult);

roi(image, roi\_left\_img, roi\_right\_img);

totalDominantColour\_roi\_img[0] = wholeImage(roi\_left\_img, processedImage, topRankColour\_img\_left, pixelPercentage\_img\_left, tempImage);

totalDominantColour\_roi\_img[1] = wholeImage(roi\_right\_img, processedImage, topRankColour\_img\_right, pixelPercentage\_img\_right, tempImage);

for (int imgIndex = 0; imgIndex < 20; imgIndex++) {

roi(flag\_trainImage[ed\_pixelPercentage[imgIndex][0]], roi\_left\_timg, roi\_right\_timg);

totalDominantColour\_roi\_timg[0] = wholeImage(roi\_left\_timg, flag\_processedTrainImage[ed\_pixelPercentage[imgIndex][0]],

topRankColour\_timg\_left, pixelPercentage\_timg\_left, tempImage);

totalDominantColour\_roi\_timg[1] = wholeImage(roi\_right\_timg, flag\_processedTrainImage[ed\_pixelPercentage[imgIndex][0]],

topRankColour\_timg\_right, pixelPercentage\_timg\_right, tempImage);

//pixel percentage

ed\_pixelPercentage\_left[imgIndex][0] = ed\_pixelPercentage[imgIndex][0];

ed\_pixelPercentage\_left[imgIndex][1] = euclideanDistance\_pixelPercentage(topRankColour\_img\_left, topRankColour\_timg\_left,

pixelPercentage\_img\_left, pixelPercentage\_timg\_left, totalDominantColour\_roi\_img[0], totalDominantColour\_roi\_timg[0],

roi\_left\_img.rows \* roi\_left\_img.cols, roi\_left\_timg.rows \* roi\_left\_timg.cols);

ed\_pixelPercentage\_right[imgIndex][0] = ed\_pixelPercentage[imgIndex][0];

ed\_pixelPercentage\_right[imgIndex][1] = euclideanDistance\_pixelPercentage(topRankColour\_img\_right, topRankColour\_timg\_right,

pixelPercentage\_img\_right, pixelPercentage\_timg\_right, totalDominantColour\_roi\_img[1], totalDominantColour\_roi\_timg[1],

roi\_right\_img.rows \* roi\_right\_img.cols, roi\_right\_timg.rows \* roi\_right\_timg.cols);

partitionFiltering[imgIndex][0] = ed\_pixelPercentage[imgIndex][0];

partitionFiltering[imgIndex][1] = (ed\_pixelPercentage\_left[imgIndex][1] + ed\_pixelPercentage\_right[imgIndex][1]) / 2;

}

sort2DArray(partitionFiltering, 10);

//display the top 10 rank images in term of similarity measurement on a white background

for (int count = 0; count < 10; count++)

cout << "ED of Average Pixel Percentage - " << fn\_flag[partitionFiltering[count][0]] << ": " << partitionFiltering[count][1] << endl;

displayResult = attachResults2AWhiteBgWithHist(partitionFiltering, flag\_trainImage, histogram, 10);

namedWindow("Average Pixel Percentage", WINDOW\_AUTOSIZE);

imshow("Average Pixel Percentage", displayResult);

imwrite("Average Pixel Percentage.jpg", displayResult);

waitKey();

break;

case '2':

displayQueryImage(image);

//central normalized moments filtering

findContour(image, processedImage, nuMoment\_img);

for (int imgIndex = 0; imgIndex < LOGO\_TRAINING\_IMAGES; imgIndex++) {

findContour(logo\_trainImage[imgIndex], logo\_processedTrainImage[imgIndex], nuMoment\_timg);

//assign image index

ed\_nuMoment[imgIndex][0] = imgIndex;

//assign euclidean distance of moment

ed\_nuMoment[imgIndex][1] = euclideanDistance\_nuMoment(nuMoment\_img, nuMoment\_timg, 7);

}

sort2DArray(ed\_nuMoment, LOGO\_TRAINING\_IMAGES); //sort in ascending order

//display the top 10 rank images in term of similarity measurement on a white background

for (int count = 0; count < 10; count++)

cout << "ED of Moment - " << fn\_logo[ed\_nuMoment[count][0]] << ": " << ed\_nuMoment[count][1] << endl;

displayResult = attachResults2AWhiteBg(ed\_nuMoment, logo\_trainImage, 10);

namedWindow("Moment", WINDOW\_AUTOSIZE);

imshow("Moment", displayResult);

imwrite("Moment.jpg", displayResult);

//hu moments filtering

for (int imgIndex = 0; imgIndex < 10; imgIndex++) {

//assign image index

index\_huMoment[imgIndex][0] = ed\_nuMoment[imgIndex][0];

//assign Hu Moment value

index\_huMoment[imgIndex][1] = matchShapes(processedImage, logo\_processedTrainImage[ed\_nuMoment[imgIndex][0]], CV\_CONTOURS\_MATCH\_I1, 0);

}

sort2DArray(index\_huMoment, 10);

//display the top 10 rank images in term of similarity measurement on a white background

for (int count = 0; count < 10; count++)

cout << "ED of Hu Moment - " << fn\_logo[index\_huMoment[count][0]] << ": " << index\_huMoment[count][1] << endl;

displayResult = attachResults2AWhiteBg(index\_huMoment, logo\_trainImage, 10);

namedWindow("Hu Moment", WINDOW\_AUTOSIZE);

imshow("Hu Moment", displayResult);

imwrite("Hu Moment.jpg", displayResult);

waitKey();

break;

case '3':

displayQueryImage(image);

//first stage, shape descriptor

findContour(image, processedImage, nuMoment\_img);

for (int imgIndex = 0; imgIndex < LOGO\_TRAINING\_IMAGES; imgIndex++) {

findContour(logo\_trainImage[imgIndex], logo\_processedTrainImage[imgIndex],nuMoment\_timg);

//assign image index

ed\_nuMoment[imgIndex][0] = imgIndex;

//assign euclidean distance of moment

ed\_nuMoment[imgIndex][1] = euclideanDistance\_nuMoment(nuMoment\_img, nuMoment\_timg, 7);

}

sort2DArray(ed\_nuMoment, LOGO\_TRAINING\_IMAGES); //sort in ascending order

//second stage, colour descriptor

totalDominantColour\_img = eliminateBackground(image, processedImage, topRankColour\_img, pixelPercentage\_img, tempImage);

for (int imgIndex = 0; imgIndex < 10; imgIndex++) {

totalDominantColour\_timg = eliminateBackground(logo\_trainImage[ed\_nuMoment[imgIndex][0]], logo\_processedTrainImage[ed\_nuMoment[imgIndex][0]],

topRankColour\_timg, pixelPercentage\_timg, histogram[ed\_nuMoment[imgIndex][0]]);

//similarity score computed by ed of dominant colour pixel percentage

ed\_pixelPercentage[imgIndex][0] = ed\_nuMoment[imgIndex][0];

ed\_pixelPercentage[imgIndex][1] = euclideanDistance\_pixelPercentage(topRankColour\_img, topRankColour\_timg, pixelPercentage\_img,

pixelPercentage\_timg, totalDominantColour\_img, totalDominantColour\_timg, image.rows \* image.cols,

logo\_trainImage[imgIndex].rows \* logo\_trainImage[imgIndex].cols);

}

sort2DArray(ed\_pixelPercentage, 10); //sort in ascending order

roi(image, roi\_left\_img, roi\_right\_img);

totalDominantColour\_roi\_img[0] = eliminateBackground(roi\_left\_img, processedImage, topRankColour\_img\_left, pixelPercentage\_img\_left, tempImage);

totalDominantColour\_roi\_img[1] = eliminateBackground(roi\_right\_img, processedImage, topRankColour\_img\_right, pixelPercentage\_img\_right,

tempImage);

for (int imgIndex = 0; imgIndex < 10; imgIndex++) {

roi(logo\_trainImage[ed\_pixelPercentage[imgIndex][0]], roi\_left\_timg, roi\_right\_timg);

totalDominantColour\_roi\_timg[0] = eliminateBackground(roi\_left\_timg, processedImage, topRankColour\_timg\_left, pixelPercentage\_timg\_left,

tempImage);

totalDominantColour\_roi\_timg[1] = eliminateBackground(roi\_right\_timg, processedImage, topRankColour\_timg\_right, pixelPercentage\_timg\_right,

tempImage);

//pixel percentage

ed\_pixelPercentage\_left[imgIndex][0] = ed\_pixelPercentage[imgIndex][0];

ed\_pixelPercentage\_left[imgIndex][1] = euclideanDistance\_pixelPercentage(topRankColour\_img\_left, topRankColour\_timg\_left,

pixelPercentage\_img\_left, pixelPercentage\_timg\_left, totalDominantColour\_roi\_img[0], totalDominantColour\_roi\_timg[0],

roi\_left\_img.rows \* roi\_left\_img.cols, roi\_left\_timg.rows \* roi\_left\_timg.cols);

ed\_pixelPercentage\_right[imgIndex][0] = ed\_pixelPercentage[imgIndex][0];

ed\_pixelPercentage\_right[imgIndex][1] = euclideanDistance\_pixelPercentage(topRankColour\_img\_right, topRankColour\_timg\_right,

pixelPercentage\_img\_right, pixelPercentage\_timg\_right, totalDominantColour\_roi\_img[1], totalDominantColour\_roi\_timg[1],

roi\_right\_img.rows \* roi\_right\_img.cols, roi\_right\_timg.rows \* roi\_right\_timg.cols);

partitionFiltering[imgIndex][0] = ed\_pixelPercentage[imgIndex][0];

partitionFiltering[imgIndex][1] = (ed\_pixelPercentage\_left[imgIndex][1] + ed\_pixelPercentage\_right[imgIndex][1]) / 2;

}

sort2DArray(partitionFiltering, 10);

//display the top 10 rank images in term of similarity measurement on a white background

cout << "\t\t\t\tAverage Pixel Percentage\tMoment\n";

for (int count = 0; count < 10; count++) {

cout << fn\_logo[partitionFiltering[count][0]] << ":\t" << partitionFiltering[count][1];

for (int count2 = 0; count2 < 10; count2++) {

if (partitionFiltering[count][0] == ed\_nuMoment[count2][0])

cout << "\t\t\t" << ed\_nuMoment[count2][1] << endl;

}

}

displayResult = attachResults2AWhiteBgWithHist(partitionFiltering, logo\_trainImage, histogram, 10);

namedWindow("Similarity Score of Combined Discriptor", WINDOW\_AUTOSIZE);

imshow("Similarity Score of Combined Discriptor", displayResult);

imwrite("Similarity Score of Combined Discriptor.jpg", displayResult);

waitKey();

break;

case '4':

system("cls");

break;

case '5':

return 0;

default:

cout << "Invalid input.\n";

system("pause");

}

if (option[0] == '4')

break;

else

destroyAllWindows();

}

}

}

}

}