Improvements on colour histogram-based CBIR

Ole Andreas Flaaten Jonsgård

Master's Thesis
Master of Technology in Media Technology
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2005

The MTech programme in Media Technology
is run in cooperation with the Royal Institute
of Technology (KTH) in Stockholm.

Institutt for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

This thesis tries out a number of new image retrieval techniques with the aim of improving the results of global colour histograms. It also looks at how to measure the effectiveness of such methods. Instead of relying solely on ground-truth databases this project suggests more use of questionnaires for this purpose. Although the CBIR methods suggested did not improve on traditional colour histograms, the questionnaire shows people are not necessarily in agreement with the results from the ground truth sets.

Sammendrag

Denne oppgaven prøver ut en del nye metoder for innholdsbasert bildesøk, med det for øye å forbedre resultatene fra rene fargehistogram-baserte metoder. Oppgaven ser også på hvordan man kan måle resultatene fra disse metodene. I stedet for å kun basere seg på sannhetssett-databaser foreslår denne oppgaven en mer utstrakt bruk av spørreundersøkelser. Selv om bildesøkmetodene som blir foreslått ikke gav noe bedre resultat enn rene fargehistogram, så viste resultatene fra spørreundersøkelsen at folk ikke nødvendigvis mener det samme som resultatene fra sannhetssett-databaser.

Acknowledgements

First and foremost I would like to thank Dr. Ali Alsam, my guide for this project for his help and suggestions. Secondly, Andrei Ouglov also deserves thanks for supplying the gamut intersection-results for the questionnaire and suggesting the pixel randomizing described in chapter 4.2. Last, but not by any means least, my fellow students and friends must be thanked. Without them, the months spent on this project would surely have been much more tedious.

Common abbreviations

CBIR	Content Based Image Retrieval (2.1)
AMP	Average Match Percentile (4.1.1)
MP	Match Percentile (4.1.1)
RGB	Red Green Blue-colour space (2.3.1)
HSV	Hue Saturation Value- colour space (2.3.2)
GCH	Global Colour Histogram (2.2.1)
LCH	Local Colour Histogram (2.2.3)
QBIC	Query By Image Content [14]
CS	Channel Splitting method (3.1.4)
AVG	Average squares method (3.1.5)
PHP	PHP Hypertext Preprocessor (3.2)
MPEG	Motion Picture Experts Group (3.3)

List of figures

Table of contents

1 Introduction

1.1 Background

With the advent of digital photography an ever increasing number of digital images is being produced. Businesses, the media, government agencies and even individuals all need to organize their images somehow. Today, the most common way of doing this is by textual descriptions and categorizing of images. This approach has some obvious shortcomings. Different people might categorize or describe the same image differently, leading to problems retrieving it again. It is also time consuming when dealing with very large databases. Content based image retrieval (CBIR) is a way to get around these problems.

Comparing two images and deciding if they are similar or not is a relatively easy thing to do for a human. Getting a computer to do the same thing effectively is however a different matter. Many different approaches to CBIR have been tried and many of these have one thing in common, the use of colour histograms. This project suggests some new, simple, methods to try and improve the results of standard colour histograms.

Another problem with CBIR research is how to measure a given method's effectiveness. One commonly used approach is by using specially compiled image databases with so called ground truth-sets. This approach was also used in this thesis, but in addition a number of people were asked to give their opinion about some of the methods in a questionnaire.

1.2 Research questions

A number of different methods for CBIR already exist and many of these use colour histograms in one form or another. The methods proposed in this thesis are no exceptions as they use histograms as a basis and try to improve their performance. The second question asked relates more to how one best can measure the performance of a given CBIR-method. The research questions for this project are therefore:

1. Will the methods proposed in this project perform better than pure colour histograms?
2. Will measuring the effectiveness of a given CBIR-method yield different results with average match percentile (AMP) calculations as opposed to asking people?

1.3 This document

This document opens with some general background information about CBIR and a description of the research questions the project has attempted to answer. The next chapter will contain some theory and a summary of related work. Chapter three will contain a description of the CBIR system, as well as a summary of the different CBIR-methods, used. A description of the experiments performed in the project follows in chapter four which in turn is followed by a conclusion and list of future work in the last chapter.

2 Theory and related work

2.1 Content based image retrieval

Figure 1: A CBIR-system

In a content based image retrieval system querying can be done with a sketch or, most commonly, by a query image (Fig. 1). A combination of the two and even textual descriptions can also be used. The goal in each case is to find the images most resembling the query. To achieve this, a number of different approaches have been suggested. Due to the vast amount of different methods in existence one could write an entire thesis describing these, as Schettini et al has done in their *A survey of methods for colour indexing and retrieval in image databases* [5]. This chapter will mainly concentrate on colour histogram-based such methods.

2.2 Colour histograms

2.2.1 Global colour histograms

For content based image retrieval to work, we have to find some features of the image that can be used when comparing it with another. One of the features most popular for image indexing and retrieval is colour. Comparing the colour distribution of two images will often say something about their similarity.

Figure 2: Two similar images and their similar colour histograms

Comparing all the colours in two images would however be very time consuming and complex, and so a method of reducing the amount of information must be used. One way of doing this is by quantizing the colour distribution into colour histograms. First introduced by Swain and Ballard [1], and used by many others, this is probably one of the more popular approaches to image retrieval today.

When computing a colour histogram for an image, the different colour axes are divided into a number of so-called bins. A three dimensional 8x8x8 RGB histogram (Fig. 3) would therefore contain a total of 512 such bins. When indexing the image, the colour of each pixel is found, and the corresponding bin's count is incremented by one. If the colour was 123, 231, 213, the bin at coordinates [4, 8, 7] would have its value incremented by one.

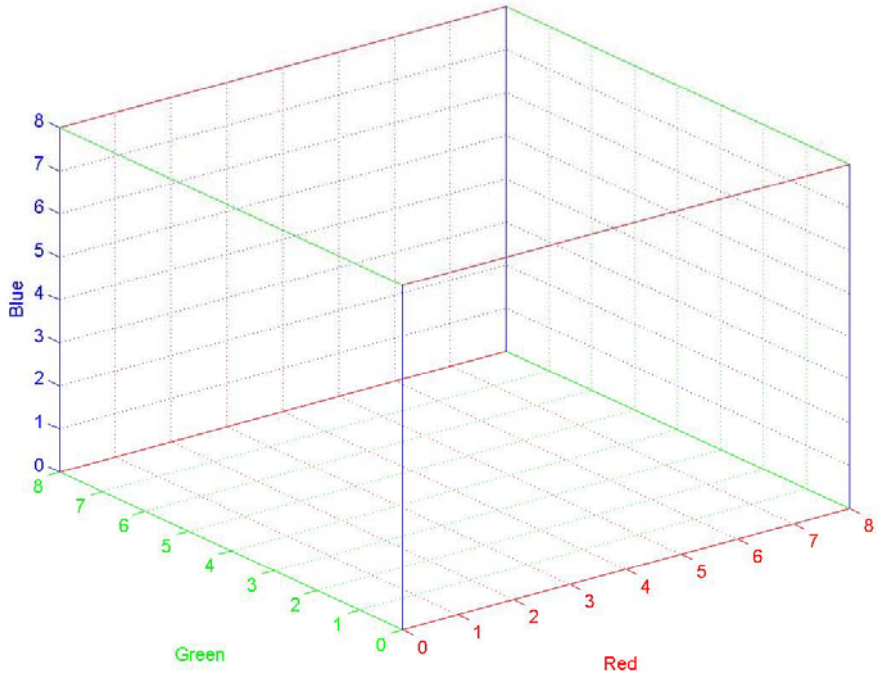


Figure 3: An empty 3-dimensional 8x8x8 RGB-histogram

2.2.2 Comparing histograms

When the images have been quantized into histograms, a method of comparing these is needed. Some of the most popular histogram comparison metrics are the L1 (1) and L2 (2) norms defined as:

$$L1 = \sum_{i=1}^{n} |Q_i - I_i| \qquad (1)$$

$$L2 = \sqrt{\sum_{i=1}^{n} (Q_i - I_i)^2} \qquad (2)$$

Where Q_i is the value of bin *i* in the query image and I_i is the corresponding bin in the database image. Based on experience from earlier projects as well as the findings of Richard Russel and Pawan Sinha in their article *Perceptually-based Comparison of Image Similarity Metrics* [6], the L1-norm is the metric of choice for this project.

2.2.3 Local colour histograms

One feature of colour histograms that can be both an advantage and a disadvantage is their lack of spatial information. This can be an advantage as a given image's global histogram will remain the same when rotated or flipped. This does however also mean that two perceptually very different images with similar colour distribution will be

deemed similar by a colour histogram-based retrieval system as illustrated in figure 4. To alleviate this problem several methods of introducing some spatial information have been suggested.

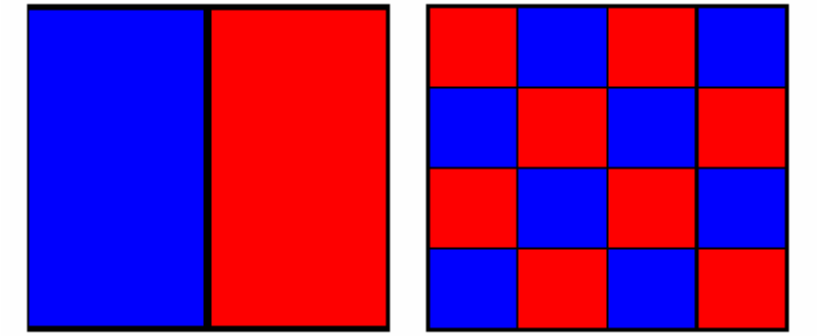


Figure 4: Two perceptually different images with equal colour distribution

The simplest and most obvious way of introducing such spatial information is the method used by Gong and others [11] . They divided the images into nine equal parts and calculated a histogram for each of these. This gives some spatial sensitivity, but increases the computing power and storage needed. One also looses the insensitivity to rotation we have in global colour histograms. The four part-method described in chapter 3.1.3 of this thesis is a variation of this method.

Shengjiu Wang [13] proposes a rotation-insensitive variant of LCH he calls the Harbin approach in his paper *A Robust CBIR Approach Using Local Color Histograms*. This method also divides the images into a number of equally sized regions and computes their LCH. The difference from Gongs approach lies in the method for comparing the images. His Harbin approach uses a system of weighted bipartite graphs to calculate the minimum cost distance between two images. With this method each part of one image is compared to all the parts of the comparison image. This makes the method less sensitive to rotation compared to Gong and others' method described above.

An approach to LCH not using equally sized regions of the image was proposed by Lu, Phillips and Harman [7]. They propose a system where two histograms are created, one for the foreground and another for the background. This was meant to alleviate the problems of background masking. Background masking occurs when an image for instance contains a small car on green lawn. The green lawn would take up more of the image area, and green would be weighted heavily in global colour histograms. This could be a disadvantage if what the user wanted to find was more images containing the small car in the foreground. Defined as the minimum rectangle containing the most important objects of an image, the foreground would be found automatically by determining vertical and horizontal pixel value transitions or set manually by the user.

2.2.4 Problems with colour histograms

The lack of spatial information is, as mentioned above, one problem of GCHs. This is however not the only problem associated with colour histograms. One other common problem is that of inter-bin similarity.

When quantizing images, every pixel will be placed in a given histogram bin, according to its colour. Inevitably, this will lead to colours that are perceptually very similar being placed in neighbouring bins. This can in turn lead to the difference between two histograms being very much larger than the perceptual difference between the actual images. To get around this problem a weighted version of the L2-metric described in section 2.2.2 of this document was suggested by Flickner, Sawhney et al in the documentation for IBM's QBIC project [14].

Another problem with colour histograms is deciding how many bins to use. A low number of bins decreases the storage space and time needed for indexing and retrieval, but also reduces the effectiveness. In his doctorate thesis Jeff Berens [2] found the ideal number of bins for, amongst others, the RGB colour space to be around 8x8x8 (512). More bins than this provided only a very small increase in effectiveness at the cost of a large decrease in speed. For this reason, the global histograms used in this thesis are 8x8x8 in size.

2.3 Colour models

A number of different ways of describing colours exists. In this thesis the RGB and HSV colour models were used, and therefore described here.

2.3.1 The RGB colour model

The RGB model uses three primary colours, red, green and blue, in an additive fashion to be able to reproduce other colours. As this is the basis of most computer displays today, this model has the advantage of being easy to extract. In a true-colour image each pixel will have has a red, green and blue value ranging from 0 to 255 giving a total of 16777216 different colours.

Figure 5: Additive mixing of red, green and blue [8]

One disadvantage with the RGB model is its behaviour when the illumination in an image changes. The distribution of rgb-values will change proportionally with the illumination, thus giving a very different histogram as shown in fig. 6.

To address some of the problems with the RGB colour space and region based image query, Syeda-Mahmood [19] proposed to partition it into 220 categories of perceptually similar colours.

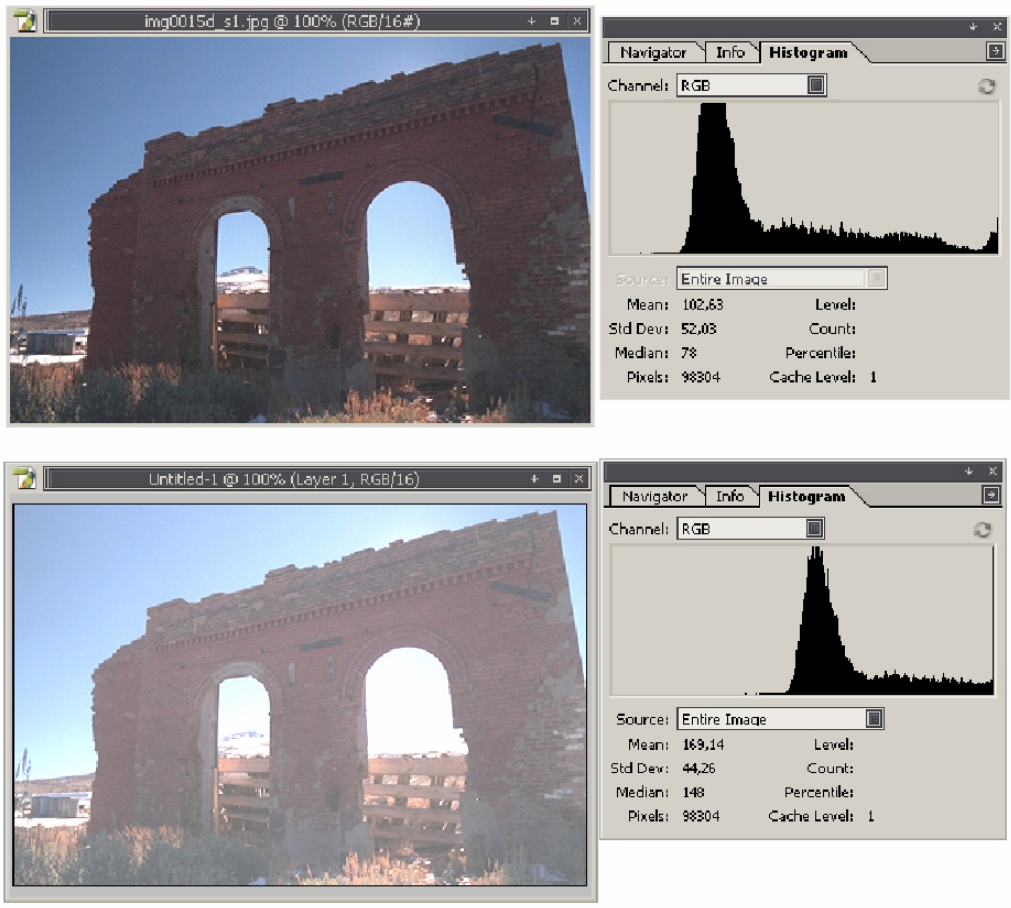


Figure 6: Changing the brightness in an RGB-image changes the histogram.

2.3.2 The HSV colour model

HSV defines a colour by its hue, saturation and value rather than by primary colours. The hue indicates a given colour's colour-type and ranges from 0 to 255. Saturation defines the vibrancy of the colour and ranges from 0 to 100% as does the hue. The latter describes the brightness of the colour.

Introduced in 1978 by Alvy Ray Smith, this model was based on how an artist mixes the colours on his palette. This is also a perceptually relevant colour model, meaning it describes colours similarly to how humans perceive them. This perceptual relevance might make it a better choice than RGB for image retrieval.

Calculating HSV is done by a non-linear transformation of RGB, making an HSV-based method slower than the same method using RGB. [9]

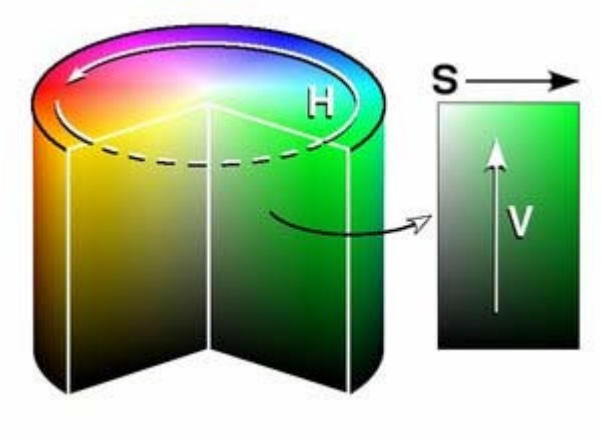


Figure 7: The HSV colour space [9]

2.4 Other image retrieval methods

Stricker and Orengo [14] proposed an alternative to colour histograms in 1995 with their colour moments. Instead of quantizing the images into histograms they use three features they call moments. These are the average colour of the image, the standard deviation of each colour channel and lastly the third root of each channel. This was shown to produce somewhat better results than simple GCH. Another plus with this method is the increase in speed as each image is described by only nine numbers (as opposed to the 512 numbers in an 8x8x8 RGB histogram). The average colour squares-method (3.1.5) in this thesis uses one of these moments, the average colour.

In 1996 Stricker and Dimai [10] introduced a method where they divided images into five partially overlapping regions (fig. 8) and calculated the colour moments of each. The centre region was given more weight than the others as this was where the most important information of a given image was thought to be. In addition, the pixels within each region are weighted according to how close they are to the region's centre.

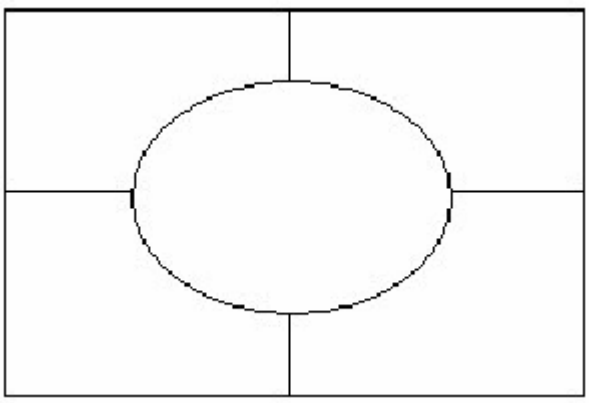


Figure 8: Image segmentation proposed by Stricker and Dimai [10]

Blobworld was the name given to a system developed by Carson and others in 1999. [16]. In this system each image is divided into regions, called blobs, which roughly correspond to an object. Colour and texture descriptors for each such blob is stored in a tree-structure and one or two of these are used for querying. Using this approach, they achieved better results than simple colour histograms.

Another approach that divides the image into regions based on its visual content was proposed by Omhover, Detyniecki and Bouchon-Meunier in 2004 [17]. Their system, called STRICT, uses a number of fuzzy similarity measures and allows the user to select an aggregate of these when running a query. Their methods have shown encouraging results compared to global colour histograms, but still require some work according to the authors themselves.

The method proposed by Smith and Chang [18] in their 1995 paper *Single colour extraction and image query* also identifies regions of colours in images. This system allows the user to query for all images containing for instance a yellow disc surrounded by blue (the sun). To achieve this, they quantize the images into a 166-bin HSV histogram and sub-sample them to approximately 196x196 pixels, keeping the correct aspect ratio. This quantized image is then colorized with a 5x5 median filter to reduce the small details and spot interference that complicates region identification. The image is then returned to an indexed RGB colour space and regions identified for all of the colours in the image. For a region to be deemed significant it has to pass several thresholds like size and contribution. Although the paper does not show any test results to show the effectiveness of the method, they conclude that it is a useful tool for image retrieval systems.

3 Resources

In this project a total of seven different methods of image indexing and retrieval was used, three existing, two new and two combination methods. To perform the experiments described in the next chapter a large heterogeneous image database was also needed.

3.1 Methods used

3.1.1 RGB Histogram

As mentioned earlier, colour histograms is one of the most common methods of image retrieval. The RGB histogram uses, as the name suggests, the RGB colour model described in chapter two.

In this project an 8x8x8-bin histogram was used to quantize the images giving a total of 512 different bins.

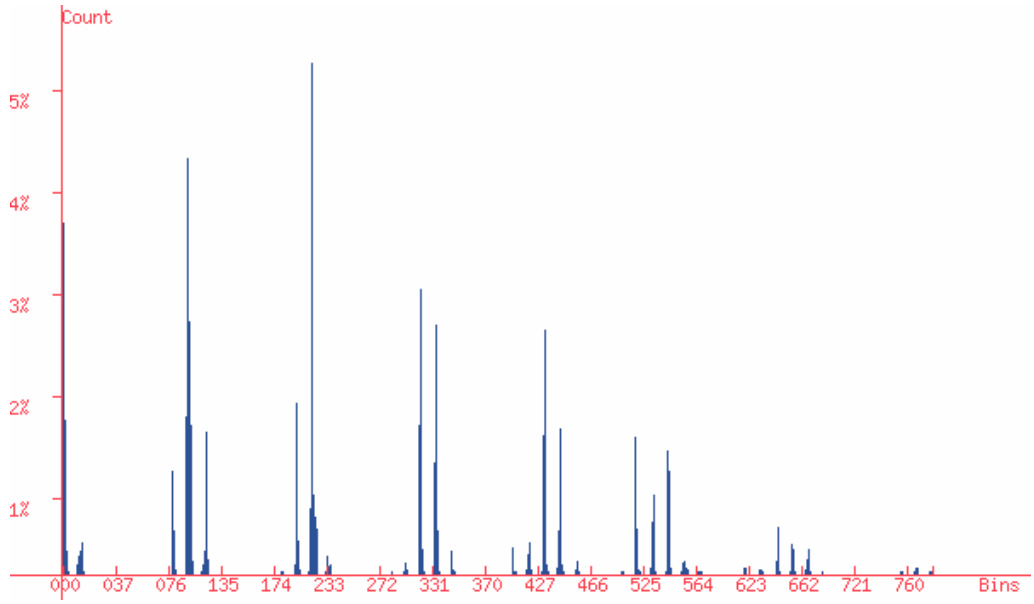


Figure 9: An 8x8x8 RGB-histogram represented two-dimensionally. Note that the numbers along the x-axis are bin-coordinates from [0,0,0] to [7,7,7].

3.1.2 HSV Histogram

This method is identical to the RGB histogram in that it quantizes the images into 8x8x8 bins. The difference lies only in the choice of colour model. For more information on HSV, see section 2.3.2 of this thesis.

3.1.3 Four part-division

Figure 10: An image divided by the four part division-method.

When using simple colour histograms no spatial information about the image is recorded. The histograms simply give a measure of how frequently a range of colours appear in the image, but not where. One simple method of adding some spatial information is dividing the image into four parts and then creating colour histograms from each of these. In this method each region of the query image is compared only to its corresponding region in the database images. The difference for each region is computed with the L1-metric described in 2.2.2 and the results are added together.

3.1.4 Channel splitting (R, G, B-histograms)

Figure 11: An image divided into its red, green and blue components

Shortened CS, the first of the two new methods proposed here is the channel splitting method. This method simply splits the image into its red, green and blue components, divides these into a number of columns and quantizes each column into colour histograms. This gives a total of 3x10x8 histogram-bins which are compared with the L1 metric to compute the similarity of the images.

In this project the images are divided into 10 columns and 8 colour bins. The method is tested both by itself and on the 100 best results from the RGB histogram-method.

3.1.5 Average colour-squares

Figure 12: An image and its 10x10 average colour squares.

This is the only method not directly using colour histograms. Instead each image is divided into a number of squares. The average colour of each square is then computed and used for comparison. Colour comparison is done by computing the vector distance [22] between corresponding RGB colour-vectors, and not by using the L1 metric.

For this project, this method was only used to try and improve the 100 best results from an RGB histogram query. This was done using both 5x5 and 10x10 average squares.

3.1.6 Gamut intersection

This method is proposed by Ali Alsam (the guide for this project) and Andrei Ouglov in an article submitted to the 13th colour imaging conference in Arizona in November 2005 [20]. Their method uses only the colour gamut distribution in an image. The colour gamut information is first projected onto two orthogonal planes. These projection images are then treated with a fillhole algorithm and median filter to aid in their comparing. The paper also suggests a comparison metric specific to this method. Put simply, the difference between two images is calculated as the difference between their projection images

When tested on the same MPEG7-database as used in this project, this method was shown to perform equal to or better than colour-histograms. For this reason, the method was chosen to be used in the questionnaire of this project.

3.2 The system

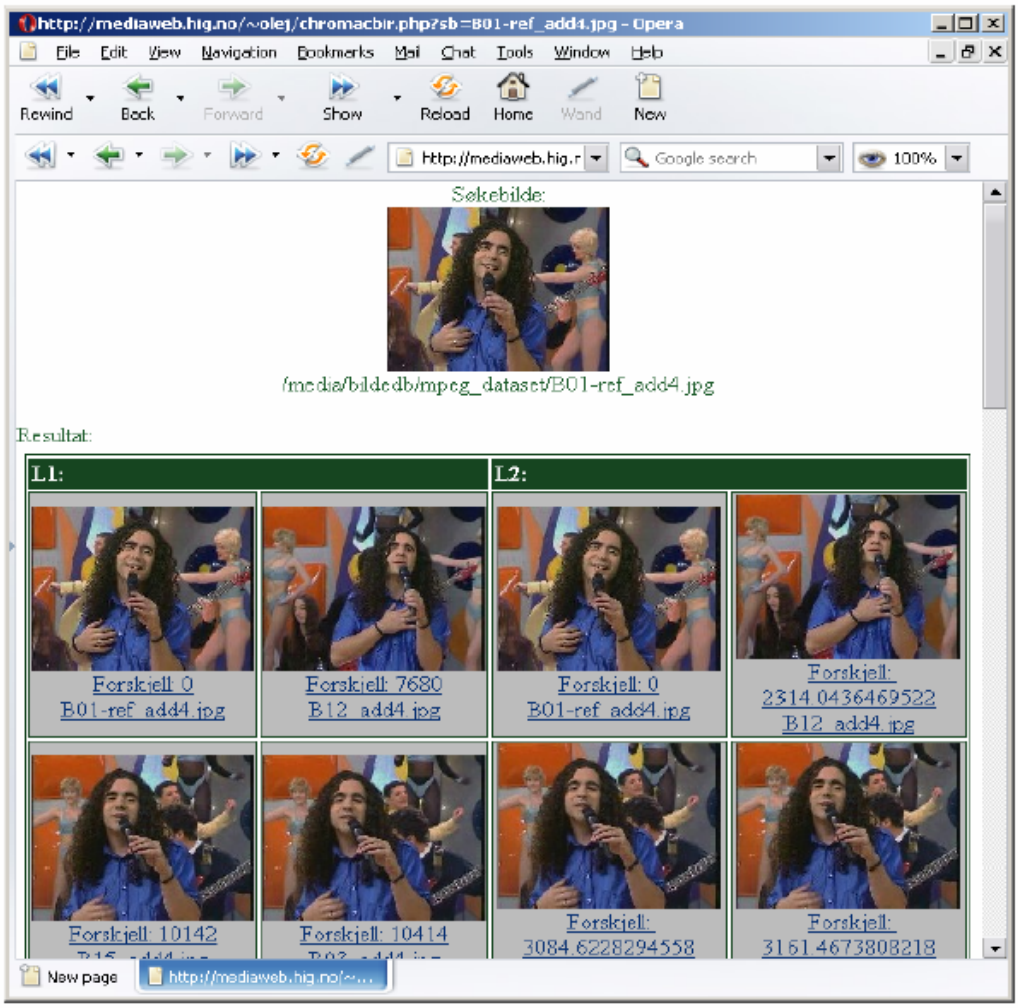


Figure 13: The web interface of the CBIR system.

In order to test the different CBIR-methods mentioned above it was necessary to have a CBIR-system. For this project, a system was implemented using PHP and MySql to give a web interface, and because the author of this thesis had some of the framework of the system already created for an earlier project. Despite this, this was easily the most time-consuming part of the project.

The system is built around one central PHP-class called *BildeTing*. This class contains all the functions for creating and comparing histograms etc. and can be found in appendix A. Functions in this class were called from a series of scripts to index the images using the various methods. The actual retrieval was also done by a series of scripts calling on functions in the *BildeTing* class. Examples of these scripts can be found in appendix B.

All histograms and other feature-vectors created for the project was stored in a MySql database. At the end of the project, this database contained a total of 15 tables and 66.2 megabytes of data. It is also important to note that all images were resized to 200x200 pixels before being quantized. Had this not been done, one would have had to normalize all the feature vectors during calculation for them to be comparable to one and other.

3.3 The image database

Testing and measuring the effectiveness of a CBIR-method obviously requires a large number of images. For this project, the MPEG-7 image database was chosen. This database has been used by a number of other projects, making it easier to compare their results to the results achieved here. It contains 5466 heterogeneous images from various sources ranging from video frame sequences of television shows to a large number of photographs. 386 of these images are arranged in so called ground-truth sets ranging in size from 2 to 32 images each. By using the query images of these ground-truth sets it is thus possible to calculate the effectiveness of a given CBIR-method. Exactly how this is done is described in more detail in section 4.1.1. This database has also been used by a number of other projects [2, 20], making it easier to compare their results to the results achieved here.

Figure 14: Nine of the 5466 images in the MPEG7-database.

4 Experiments

To test the effectiveness of the different CBIR-methods mentioned in the previous chapter two different approaches have been used. Firstly, all the methods have been tested on 15 ground-truth sets and had their average match percentile (AMP) calculated. Secondly a questionnaire was created and distributed to college students to see how they ranked the different methods.

4.1 AMP-measurements

4.1.1 Setup

The match percentile MP of a given image is defined as:

$$MP = (N - R)/(N - 1)$$ [2]

Where N is the number of images in the database and R is the rank of the returned image. This is calculated for each image and the results averaged. Calculating the rank of a given image is done as follows. All images in a 15 image ground-truth set returned among the top 15 results of a search is given the rank of 1. If an image is returned as number 16 it receives a rank of 2, 17 is ranked 3rd and so on. A score of 100% indicates a perfect match, meaning all images in a 15 image set were returned amongst the top 15 results.

In this project 15 of the ground-truth sets were used to calculate the AMP for each method. The complete ground-truth sets used can be found in appendix D. A search was performed, using all the methods described in chapter tree, with the first image of the ground-truth sets used as the query images. The 20 first retrieved images were then reviewed and the AMP calculated.

4.1.2 Results

GT-set:	A01-ref_add4	B01-ref_add4	C01-ref_add4	D01-ref_add4	E01-ref_add4	i0301_add5	i0312_add5	i0323_add5
Method:	i0334_add5	i0345_add5	i0356_add5	i0367_add5	i0378_add5	i0389_add5	i0400_add5	Total AMP
RGB 8x8x8	100,000 %	99,998 %	100,000 %	93,333 %	86,663 %	100,000 %	100,000 %	100,000 %
	100,000 %	100,000 %	100,000 %	100,000 %	99,998 %	100,000 %	99,998 %	98,666 %
HSV 8x8x8	100,000 %	99,998 %	100,000 %	99,999 %	99,998 %	100,000 %	100,000 %	100,000 %
	100,000 %	100,000 %	100,000 %	100,000 %	99,998 %	100,000 %	99,998 %	99,999 %
CS 10x8	86,664 %	20,000 %	99,999 %	46,661 %	93,330 %	100,000 %	100,000 %	100,000 %
	100,000 %	100,000 %	100,000 %	100,000 %	99,998 %	100,000 %	99,998 %	89,777 %
RGBCS	100,000 %	53,331 %	100,000 %	86,665 %	73,328 %	100,000 %	100,000 %	100,000 %
	100,000 %	100,000 %	100,000 %	100,000 %	99,998 %	100,000 %	99,998 %	94,221 %
4Part	100,000 %	86,663 %	100,000 %	100,000 %	80,000 %	100,000 %	100,000 %	100,000 %
	100,000 %	100,000 %	100,000 %	100,000 %	99,998 %	100,000 %	99,998 %	97,777 %
avgRGB 5x5	99,998 %	66,656 %	99,994 %	59,996 %	79,993 %	100,000 %	100,000 %	100,000 %
	100,000 %	100,000 %	100,000 %	100,000 %	99,998 %	100,000 %	99,998 %	93,776 %
avgRGB 10x10	99,996 %	59,996 %	93,332 %	46,656 %	59,991 %	100,000 %	100,000 %	100,000 %
	100,000 %	100,000 %	100,000 %	100,000 %	99,998 %	100,000 %	99,998 %	90,665 %

Table 1: The average match percentile results.

The best AMP results were achieved by the HSV histogram method with a score of 99.9%. RGB histograms came in a close second with 98.6%. A slightly surprising third was the 4-part method. Of the new methods proposed in this thesis, the RGB+CS came out best with 94.2%, followed by RGB+AVG 5x5 and 10x10 with 93.7% and 90.6% respectively. The CS-method alone came out worst with 89.7%, the only below 90%.

Looking at the results of individual GT-sets, the results are not always this clear. The CS method does for instance produce better results on one image than simple RGB histograms whilst performing poorer on others. HSV histograms do however perform consistently better than or equal to RGB. It is also interesting to note that all the methods produce the same AMP score on ten of the 15 images.

4.1.3 Discussion

From the results discussed above, it seems clear that the best method (of those tested) for image indexing and retrieval on this particular database is HSV histograms, with RGB histograms coming in a close second. The most unexpected results came from the 4-part method. This failed to do better than simple global histograms in total, and actually performed better than these on only one of the 15 query images. It is however worth noting that this method only performed worse on two sets, the only sets where none of the methods received a perfect score.

None of the new methods introduced by this thesis produce consistently better results than the RGB or HSV histograms. Although the CS method gives a better score than RGB on one of the images, it does quite a lot worse on several others. The combined methods which were meant to improve the performance of RGB histograms do quite the opposite. All three of these perform consistently worse than the simple global RGB and HSV histograms.

The fact that all the methods gave the same, perfect or near perfect, results on 10 of the 15 sets also shows some of the problems with these kinds of measurements. The images in these sets seem to be a sequence of video frames, and are very similar in appearance. As well as being very dissimilar in colour to all the other images in the database. As the different methods obviously do not perform equally well, it becomes necessary to find other methods of measuring their performance. For instance a questionnaire as described in chapter 4.2.

Because the database also contained single frames that appeared to be from the same sequences, but not part of the ground-truth sets, not all of these 10 sets gave a result of 100% as they would otherwise have done.

4.2 Questionnaire

4.2.1 Setup

The AMP gives a good indication of a given method's effectiveness. A weakness with such calculations however is its inability to say anything about the similarity of the first non-ground truth set images returned to the query image. There is only one way of measuring this, by asking people. In order to do this, a questionnaire was created. To show people the results from all the methods and ground-truth sets would require a lot of time and was unrealistic for a project of this size. Consequently three methods were chosen: RGB histogram, RGB+AVG and a colour gamut distribution method [20]. The results for the third method were provided by Andrei Ouglov**,** one of the authors of the article in which it was proposed.

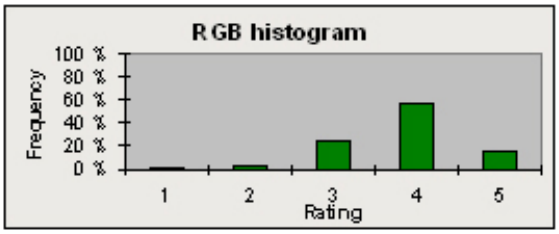
The participants were shown the results from a search on the same ground-truth set for each of the methods. The ground truth set was chosen at random from the 15 being used in the AMP-calculations discussed in 4.1. The subjects were then asked to first rate the results of each method from 1 to 5 (very bad, bad, mediocre, good, very good) and second to rank them from 1 to 3. In the second part of the questionnaire the same results were shown again, but this time the pixels of each image were randomized. These results were rated and ranked in the same manner as the first three.

The purpose of randomizing the pixels in the second part of the questionnaire was to see how the participants ranked the different methods based solely on colour distribution and not on actual image content and how these results differed from those of the first part.

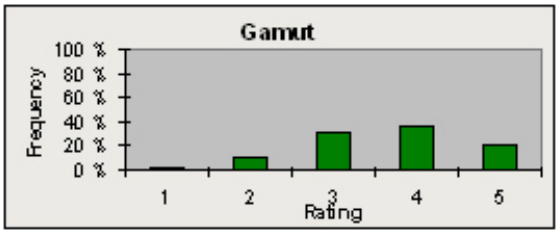
4.2.2 Results

As mentioned in the previous section, the questionnaire was web-based. The url was mailed to all students at Gjøvik College as well as friends of the author. A total of 180 people answered, but due to a large amount of incomplete answers the final number of results was 106. The participants ranged in age from 20 to 44 and consisted of 65 males and 41 females. No information was collected about the background of the participants, but judging by who the url was sent to it is probably safe to assume that they have some college-education and are in the process of aquiring more.

Bin	Frequency	%
1	1	1 %
2	3	3 %
3	25	24 %
4	61	58 %
5	16	15 %
Sum	106	
	Average:	3,8



Bin	Frequency	%
1	2	2 %
2	11	10 %
3	33	31 %
4	38	36 %
5	22	21 %
Sum	106	
		3,6



Bin	Frequency	%
1	2	2 %
2	11	10 %
3	51	48 %
4	36	34 %
5	6	6 %
Sum	106	
		3,3

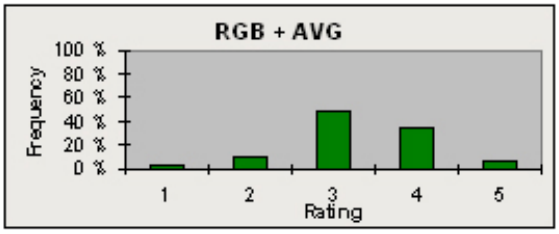
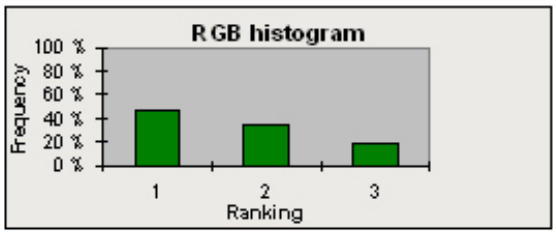


Figure 15: Ratings from part 1 of the questionnaire.

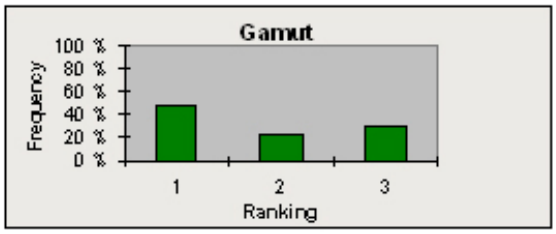
When looking at the first set of ratings (fig. 15), we see that the simple RGB histogram comes out top with 73% of the participants rating it as good or very good. 58% rate it as good and 15% very good (4 on a scale from 1 to 5). Similar numbers for the RGB+AVG and gamut methods are 40% and 56% respectively. Note also that the gamut method has the largest percentage of *very good* ratings with 21%. The RGB+AVG method stands out as being the most average method with a total of 48% giving it this rating.

The average ratings of this first set shows that RGB histograms come out best with an average 3.8, closely followed by Gamut and AVG+RGB with 3.6 and 3.3 respectively.

Bin	Frequency	%
1	49	46 %
2	37	35 %
3	20	19 %
Sum	106	100 %
		1,7



Bin	Frequency	%
1	51	48 %
2	24	23 %
3	31	29 %
Sum	106	100 %
		1,8



Bin	Frequency	%
1	5	5 %
2	48	45 %
3	53	50 %
Sum	106	100 %
		2,5

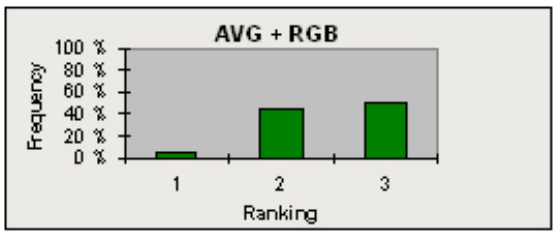
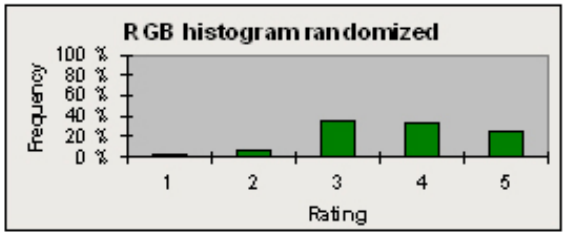


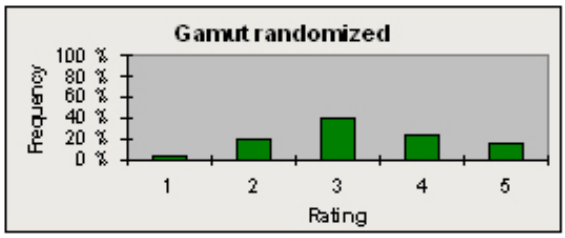
Figure 16: Rankings from part 1 of the questionnaire.

After rating each method individually, the test subjects were then asked to rank the three methods from one to three. Looking at these rankings we see that RGB histograms and the gamut method are ranked quite evenly. Although the gamut method gets a slightly higher number of 1st places, 48% vs. 46%, the RGB histogram gets a slightly better average ranking of 1.7 versus the 1.8 of gamut. When taking the number of 2nd places into account, the RGB histogram steals the lead with 81% 2nd or better versus gamut's 71%. The AVG+RGB method comes out as the clear looser with 50% ranking it third and only 5% giving it the first place.

Bin	Frequency	%
1	1	1 %
2	6	6 %
3	38	36 %
4	35	33 %
5	26	25 %
Sum	106	
		3,7



Bin	Frequency	%
1	4	4 %
2	20	19 %
3	41	39 %
4	24	23 %
5	17	16 %
Sum	106	
		3,3



Bin	Frequency	%
1	2	2 %
2	9	8 %
3	30	28 %
4	43	41 %
5	22	21 %
Sum	106	
		3,7

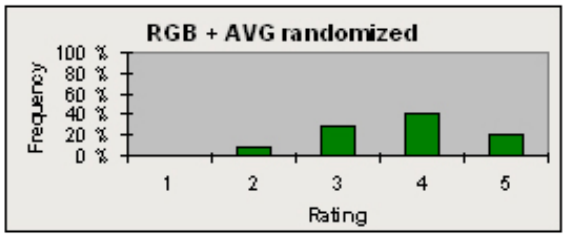
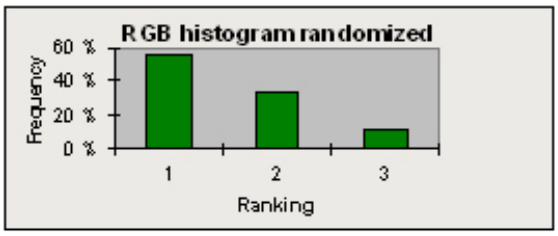


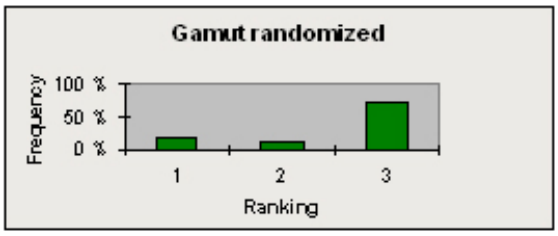
Figure 17: Ratings from part two of the questionnaire.

In the second set of results the participants were asked to rate, the images were exactly the same as in the previous set, but with all the pixels randomly repositioned. This does give some different results than the first set. Most notably RGB histograms and RGB+AVG are tied for first place, both having an average rating of 3.7. The gamut method falls down to a clear third with its 3.3 average. Looking closely at the numbers we see that the RGB+AVG method has the highest percentage of *good* and *very good* ratings with 62% total. Gamut is clearly worst with 62% average or lower.

Bin	Frequency	%
1	59	56 %
2	36	34 %
3	11	10 %
Sum	106	100 %
		1,5



Bin	Frequency	%
1	19	18 %
2	11	10 %
3	76	72 %
Sum	106	100 %
		2,5



Bin	Frequency	%
1	34	32 %
2	61	58 %
3	11	10 %
Sum	106	100 %
		1,8

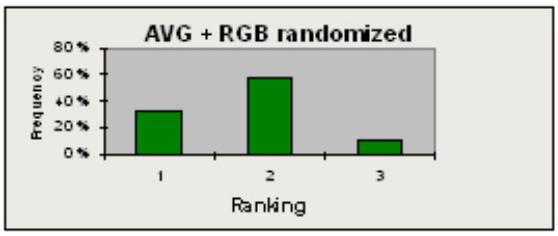


Figure 18: Rankings from part two of the questionnaire.

The rankings of the randomized results are quite clear. The RGB histogram comes out 1st with 59% giving it this ranking. AVG+RGB comes in a clear second with 61% ranking it as such and the gamut method gets a quite convincing 72% of the third places.

In addition to the rating and rankings, the people answering the questionnaire were invited to give any comments they might have. A total of 25 participants chose to leave their comments. These varied in length from one to 167 words and ranged from expressing excitement about CBIR to misgivings about the layout of the questionnaire.

4.2.3 Discussion

When looking at the first part of the questionnaire results we see that people clearly rated the AVG+RGB method worse than the two other methods. This mirrors the results of the AMP-calculations in 4.1.2. When it comes to deciding a winner things get slightly more unclear. The gamut-method, which has been shown to perform favourably to or equally well as colour histograms, gets a slightly higher percentage of *very good*-ratings, but the RGB histograms comes out top with its higher combined percentage of *good* and *very good* and consequently higher average rating. The same can be seen in the rankings of this first portion. The gamut-method gets the highest

percentage of 1st places as well as 10% more 3rd placed than the RGB histogram. AVG+RGB again comes out a clear third.

From this first part of the results then, we can conclude that AVG+RGB does not perform better than traditional colour histograms in the opinion of the test subjects. This is not at all unexpected. What was more unexpected however was the rating of the gamut intersection method. Although it got the most *very good*-ratings and 1st place rankings, it also received more *average* and below rankings as well as 3rd places than the RGB histogram-method. This may be explained to some extent by the relative darkness of some of the non-ground truth images returned. These dark images were from the same basketball game as the ground truth set (the ground truth set used can be foun on page 13 of appendix D), but showed the entire stadium as opposed to close-ups of the game. If the subject only glanced at the images they may not have noticed this. It is also important to note that the subjects of this test were only shown the results from one search. With a different query image the results may have been different, although both the gamut and colour histogram methods had a 100% match percentile on this set.

The second part of the questionnaire was intended to see whether or not people's opinion of the search results would change when the images had their pixels randomized. As can be seen from the results described in 4.2.2, they did. The histogram and RGB+AVG-methods are now tied for first place when looking at the average rating. RGB+AVG actually gets more *good* and *very good* ratings than the pure histogram method. Gamut intersection now comes in a clear number three.

These results are not unexpected and can be probably be explained by looking at how the different methods work. As explained in chapter 3, the RGB+AVG method first uses RGB histograms to retrieve images. The first 100 of these are then sorted again using the average squares method. Because of this, chances are good the randomized images will look similar to those returned by just the RGB histogram method. The last images returned by the gamut intersection method appear much darker than the rest, and this is probably the reason why it was ranked last.

One potential source of errors with the results of this experiment is how the questionnaire itself was constructed. 4 of the 25 people who left comments said it was difficult to compare the different methods without viewing all the results at the same time. Two people also commented that the last three methods, the randomized results, were very difficult to compare, something that could help explain the difference in the results from the first part. The fact that people were only shown the result of one search could also be a problem. The reason they were not shown more was a simple question of attention span. The fact that 74 people started but failed to complete the questionnaire might indicate it was more than long enough as it was. In addition, one person commented (using 167 words) that he felt the questionnaire was to time consuming.

5 Conclusions and future work

5.1 Conclusions

One of the goals of this project, as described in section 1.2, were to see if any of the new methods introduced here could improve on the effectiveness of simple global colour histograms. Although the new methods sadly did not perform better, this goal was nonetheless achieved. Through both AMP measurements and a questionnaire it was shown that neither the average squares- or the channel splitting-methods produced any better results than global colour histograms.

As mentioned in chapter 4.1.3, the ground-truth sets in this database all seem to be made up of sequences of video frames. This means that the images in each set are very similar in colour distribution and could be one of the reasons why the pure histogram methods showed such good results. In a real world system like a newspaper's image database, the results might be very different. This is a problem with all ground-truth databases. The ground-truth sets need to be put together by a person, and as such are a result of their subjective feelings on what constitutes image similarity. Whichever method one chooses to measure the performance of ones CBIR-methods it is important to note that this is a field where it is difficult to find absolute truths. Some methods may be very good at retrieving certain types of images, but perform poorly on others.

The second goal of this thesis was also answered. The results of the questionnaire yielded slightly different results than those of the AMP-measurements. The standard RGB-histograms came out on top in all of the tests, despite the gamut intersection method's better results on ground truth-testing. It is too early to say anything conclusive on the basis of the results achieved here, but it could indicate that ground truth testing is not the best way of measuring a given CBIR-method's effectiveness. The results of the randomized images-part of the questionnaire did differ quite a bit from the first part. Whether or not this is actually a useful method to compare cbir-results needs more work to be determined. After all, people are usually most interested in the actual content of the image, rather than its colour distribution.

5.2 Further work

On the methods proposed in this thesis, further work and testing is needed for them to be effective, but they have shown some promise.

An area that certainly warrants further research is that of measuring CBIR performance. This is an area where there has not been a lot of previous work. The focus of most papers found by the author of this thesis seems to be on suggesting new and clever methods for image retrieval, not on how best to judge their effectiveness. Using ground truth databases can be a good indicator, but it seems worthwhile to also ask people their opinion. One interesting project could be devising a standardized questionnaire or method for this. This could be used in conjunction with ground-truth testing and would allow for a more balanced view of a method's effectiveness.

Another area in the field of CBIR where more work is needed is creating consumer applications that use the technology already available. None of the large search engines use content based retrieval when this thesis is written, and only a handful of applications exists. One of these is the open source-project imgSeek [21], but this is still early in development.

6 References

[1] M. J. Swain and D.H. Ballard. *Color indexing*. In International Journal of Computer Vision, Vol. 7(1), pp 11-32, 1991

[2] Jeff Berens. Image *Indexing using Compressed Colour Histograms*. Thesis submitted for the Degree of Doctor of Philosophy in the School of information Systems, University of East Anglia, Norwich

[3] Greg Pass and Ramin Zabih. *Comparing Images Using Joint Histograms*. ACM Journal of Multimedia Systems, Vol. 7(3), pp. 234-240, May 1999

[4] Marcus Stricker and Alexander Dimai. *Spectral Covariance and Fuzzy Regions for Image Indexing*. In Machine Vision and Applications, vol. 10, pp 66-73, 1997

[5] R. Schettini, G. Ciocca, S Zuffi. *A survey of methods for colour image indexing and retrieval in image databases*. Color Imaging Science: Exploiting Digital Media, (R. Luo, L. MacDonald eds.), J. Wiley, 2001.

[6] R. Russel, P Sinha. *Perceptually based Comparison of Image Similarity Metrics.* MIT AI Memo 2001-014. Massachusetts Institute of Technology, 2001

[7] Guojun Lu. *Multimedia database management systems*, chapter 6, pp 131-177, Artech House, 1999

[8] Wikipedia article on the RGB colour model. http://en.wikipedia.org/wiki/RGB, , last visited june 29th 2005.

[9] Wikipedia article on HSV. http://en.wikipedia.org/wiki/HSV_color_space, last visited June 29th 2005.

[10] Markus Stricker and Alexander Dimai. *Color indexing with weak spatial constraints*. SPIE conference, Feb. 96, San Jose.

[11] Gong Y, Chuan C.H, Xiaoyi G. *Image indexing and and retrieval using color histograms*, Multimedia Tools and Applications, vol.2 pp. 133-156, 1996

[12] Lu, G. J. Phillips and S. Rahman. *Techinques to Improve Color Based Image Retrieval Performance.* Proceedings of International Symposium on Audio, Video, Image Processing and Intelligent Applications, Baden Baden, Germany. pp 57-61. August 17-21 1998

[13] Shengjiu Wang, *A Robust CBIR Approach Using Local Color Histograms*, Technical Report TR 01-03, Departement of computing science, University of Alberta, Canada. October 2001.

[14] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. *Query by image and video content the qbic system*. *IEEE Computer*, 28(9):23–32, 1995.

[15] Markus Stricker and Markus Orengo, *Similarity of color images*, SPIE 95, San Jose. 1995

[16] Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein and Jitendra Malik, *Blobworld: A System For Region-Based Image Indexing and Retrieval*, International Conference on Visual Information Systems '99. 1999.

[17] J.F. Omhover, M. Detyniecki and B. Bouchon-Meunier, *A Region Similarity Based Image Retrieval System*, The 10th International conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems Perugia, Italy 2004.

[18] John R. Smith and Shih-Fu Chang, *Single Color Extraction and Image Query*, Proceedings of the I.E.E.E. International Conference on Image Processing (ICIP-95), Washington DC, October 1995

[19] Tanveer Fathima Syeda-Mahmood, *Data and Model-Driven Selection Using Colour Regions,* MIT AI memo no. 1270, Massachusetts Institute of Technology, 1992.

[20] Ali Alsam and Andrei Ouglov, *Gamut Intersection For Image Indexing*, submitted to the 13th color imaging conference, Scotsdale Arizona, November 7th - 11th 2005.

[21] Homepage of the imgSeek project, http://imgseek.python-hosting.com/, last visited june 30th 2005.

[22] Wikipedia article on vectors, http://en.wikipedia.org/wiki/Vector_(spatial), last visited june 30th 2005.

7 Appendixes

Appendix A: PHP-code for the BildeTing class.

Appendix B: PHP-code for one of the retrieval and indexing scripts.

Appendix C: All questionnaire results.

Appendix D: Ground-truth sets.

Appendix A

PHP-code for the BildeTing class.

```php
<?php
    class bildeTing
    {
        //klassevaribler
        var $s_dbhost = 'localhost';    //databasehost
        var $s_dbuser = 'olej';         //brukernavn
        var $s_dbpw   = 'Gunnar';       //passord
        var $s_dbname = 'olej';         //databasenavn


        /*********************************************************************/
        /*                  histogram(bildeurl, antall bøtter)               */
        /*                  Returns a $binCount^3 RGB histogram              */
        /*********************************************************************/
        function histogram($bildeUrl, $binCount)
        {
            mysql_connect($this->s_dbhost,$this->s_dbuser,$this->s_dbpw)
                || die(mysql_error());
            @mysql_select_db($this->s_dbname)
                || die("Unable to select database");

            $im = imagecreatefromjpeg($bildeUrl)
                or die("oisann, klarte ikke å lage bilde nei");
            $orgSize = getImageSize($bildeUrl);

            $smallIm = $this->reSize($bildeUrl, 200, 200);
            //$smallIm = $im;
            $binDiv = 256 / $binCount;
            $histogram = array(array(array()));
            for ($x = 0; $x < imagesx($smallIm); $x++)
            {
                for ($y = 0; $y < imagesy($smallIm); $y++)
                {
                    //Finner fargen til den aktuelle pixelen
                    $rgb = imageColorAt($smallIm, $x, $y);
                    //Deler fargen i R, G og B-komponenter
                    $r = ($rgb >> 16) & 0xFF;
                    $g = ($rgb >> 8) & 0xFF;
                    $b = $rgb & 0xFF;
                    //Finner bøttekoordinatene ved å heltallsdividere på $binDiv (som er 256
                    delt
                    // på antall bøtter)
                    $rBotte = bcdiv($r, $binDiv, 0);
                    $gBotte = bcdiv($g, $binDiv, 0);
                    $bBotte = bcdiv($b, $binDiv, 0);
                    //Legger til en i det aktuelle bøttekoordinatet.
                    $histogram[$rBotte][$gBotte][$bBotte]++;
                }//for
            }//for
            return $histogram;
        }//function histogram


        /*********************************************************************/
        /*                       statVerdier(bildeurl)                       */
        /*          Calculates the colour modes of the R-channel             */
        /*********************************************************************/
        function statVerdier($bildeURL)
        {
            $sum = 0;
            $sum2 = 0;
            $sum3 = 0;
            $statArray = array();

            mysql_connect($this->s_dbhost,$this->s_dbuser,$this->s_dbpw)
                || die(mysql_error());
            @mysql_select_db($this->s_dbname)
                || die("Unable to select database");

            $smallIm = $this->reSize($bildeURL);

```

```
71              //Middelverdi.
72              for ($x = 0; $x < imagesx($smallIm); $x++)
73              {
74                  for ($y = 0; $y < imagesy($smallIm); $y++)
75                  {
76                      $rgb = imageColorAt($smallIm, $x, $y);
77                      $farge = ($rgb >> 16) & 0xFF;
78                      $sum += $farge;
79                  }//for
80              }//for
81
82              $n =  160*120;
83              $middel = (1/$n) * $sum;
84
85
86              //Varians og skjevhet
87              for ($x = 0; $x < imagesx($smallIm); $x++)
88              {
89                  for ($y = 0; $y < imagesy($smallIm); $y++)
90                  {
91                      $rgb = imageColorAt($smallIm, $x, $y);
92                      $farge = ($rgb >> 16) & 0xFF;
93                      $sum2 += pow(($farge - $middel), 2);
94                      $sum3 += pow(($farge - $middel), 3);
95                  }//for
96              }//for
97
98              $varians = pow(((1/$n) * $sum2), 0.5);
99              if ($sum3 <0)
100             {
101                 $skjevhet = -pow(((1/$n) * abs($sum3)), 1/3);
102             }
103             else
104             {
105                 $skjevhet = pow(((1/$n) * $sum3), 1/3);
106             }
107
108             // Legger inn i databasen
109             $insertQuery =  "INSERT INTO oleaStat (filnavn, middelverdi, varians, skjevhet)
                VALUES ('$bildeURL', '$middel', '$varians', '$skjevhet')";
110             mysql_query($insertQuery);
111
112             $statArray[0] = $middel;
113             $statArray[1] = $varians;
114             $statArray[2] = $skjevhet;
115
116             return $statArray;
117
118         }//function statVerdier
119
120
121         /*********************************************************************/
122         /*              gatekryssAvstandsBeregning(h1, h2)                   */
123         /*          Returns the L1-difference between the histograms         */
124         /*                          h1 and h2                                */
125         /*********************************************************************/
126         function gatekryssAvstandsBeregning($h1, $h2)
127         {
128             $forskjell = 0;
129             for($i = 0; $i < count($h1); $i++)
130             {
131                 $forskjell += abs($h1[$i] - $h2[$i]);
132             }
133             return $forskjell;
134         }
135
136
137         /*********************************************************************/
138         /*                          L2(h1, h2)                               */
139         /*          Returns the L2-difference between the histograms         */
140         /*                          h1 and h2                                */
```

```php
/*********************************************************************/
function L2($h1, $h2)
{
    $forskjell = 0;
    for($i = 0; $i < count($h1); $i++)
    {
        //print "<br /> HER ER JEG!!!!! <br />";
        $forskjell += pow(($h1[$i] - $h2[$i]), 2);
    }
    return sqrt($forskjell);
}


/*********************************************************************/
/*                  reSize(bildeurl, høyde, bredde)                  */
/*      Returns a copy of $bildeUrl with $x height and $y width      */
/*********************************************************************/
function reSize($bildeUrl, $x, $y)
{
    $imr = imagecreatefromjpeg($bildeUrl)
    or die("oisann, klarte ikke å lage bilde i resize nei");
    $orgSize = getImageSize($bildeUrl);
    $smallIm = imagecreatetruecolor($x, $y);
    imagecopyresized($smallIm, $imr, 0, 0, 0, 0, $x, $y, $orgSize[0], $orgSize[1]);
    return $smallIm;
}


/*********************************************************************/
/*                      rgb_to_hsv(array(r, g, b)                    */
/*                        Converts RGB to HSV                        */
/*********************************************************************/
function rgb_to_hsv($rgb)
{
    for ($c=0; $c<3; $c++)
    {
        $rgb[$c] = $rgb[$c] / 255;
    }

    $hsv = array(0, 0, 0);
    $max = max($rgb);
    $min = min($rgb);

    $hsv[2] = $max;
    $hsv[1] = ($max) ? (($max - $min) / $max) : 0;

    if (!$hsv[1])
    {
        $hsv[0] = null;
    }
    else
    {
        $delta = $max - $min;
        if ($rgb[0] == $max)
        {
            $hsv[0] = ($rgb[1] - $rgb[2]) / $delta;
        }
        else if ($rgb[1] == $max)
        {
            $hsv[0] = 2 + ($rgb[2] - $rgb[0]) / $delta;
        }
        else
        {
            $hsv[0] = 4 + ($rgb[0] - $rgb[1]) / $delta;
        }

        $hsv[0] *= 60;
        if ($hsv[0] < 0)
        {
            $hsv[0] += 360;
        }
```

```
212             }
213             ksort($hsv);
214             $hsv[1] = $hsv[1] * 100;
215             $hsv[2] = $hsv[2] * 100;
216             for($i = 0; $i < count($hsv); $i++)
217             {
218                 $hsv[$i] = round($hsv[$i]);
219             }
220             return $hsv;
221         }// function rgb_to_hsv
222 
223         /**********************************************************************/
224         /*                  rgb_to_chroma(array(r, g, b)                      */
225         /*              Converts RGB to 2d opponent chromaticity space        */
226         /**********************************************************************/
227         function rgb_to_chroma($rgb)
228         {
229             $crhoma = array();
230             $R = $rgb[0];
231             $G = $rgb[1];
232             $B = $rgb[2];
233 
234             if ($R==$G && $G==$B && $B==0)
235             {
236                 $litenr = 0;
237                 $liteng = 0;
238                 $litenb = 0;
239             }
240             else
241             {
242                 $litenr = round(($R / ($R+$G+$B)) * 100);
243                 $liteng = round(($G / ($R+$G+$B)) * 100);
244                 $litenb = round(($B / ($R+$G+$B)) * 100);
245             }
246                 $chroma[0] = $litenr-$liteng;
247                 $chroma[1] = $litenr+$liteng-(2*$litenb);
248 
249             return $chroma;
250         }// function rgb_to_chroma()
251 
252 
253         /**********************************************************************/
254         /*                  histogram(bildeurl, antall bøtter)                */
255         /*          Returns a $hBinCount*$svBinCount^2 HSV histogram          */
256         /**********************************************************************/
257         function hsvHist($bildeUrl, $hBinCount, $svBinCount)
258         {
259             $im = imagecreatefromjpeg($bildeUrl)
260                 or die("oisann, klarte ikke å lage bilde nei");
261             $orgSize = getImageSize($bildeUrl);
262 
263             $smallIm = $this->reSize($bildeUrl, 200, 200);
264             $hBinDiv = 360 / $hBinCount; //5, 10 eller potens av 2
265             $svBinDiv = 100 / $svBinCount; // potens av 5
266 
267             $histogram = array(array(array()));
268             $rgbArr = array();
269 
270             for ($x = 0; $x < imagesx($smallIm); $x++)
271             {
272                 for ($y = 0; $y < imagesy($smallIm); $y++)
273                 {
274                     //Finner fargen til den aktuelle pixelen
275                     $rgb = imageColorAt($smallIm, $x, $y);
276                     //Deler fargen i R, G og B-komponenter
277                     $rgbArr[0] = ($rgb >> 16) & 0xFF;
278                     $rgbArr[1] = ($rgb >> 8) & 0xFF;
279                     $rgbArr[2] = $rgb & 0xFF;
280                     //gjør om til HSV
281                     $hsv = $this->rgb_to_hsv($rgbArr);
282                     //Finner bøttekoordinatene ved å heltallsdividere på $hBinDiv og
```

```
                    $svBinDiv
283                 $hBotte = bcdiv($hsv[0], $hBinDiv, 0);
284                 $sBotte = bcdiv($hsv[1], $svBinDiv, 0);
285                 $vBotte = bcdiv($hsv[2], $svBinDiv, 0);
286                 //Legger til en i det aktuelle bøttekoordinatet.
287                 $histogram[$hBotte][$sBotte][$vBotte]++;
288             }//for
289         }//for
290         return $histogram;
291     }//function hsvHist
292
293
294     /*********************************************************************/
295     /*                  chromaHist(bildeurl, antall bøtter)              */
296     /*          Returns a $hBinCount^3 2D-oppoenent chroma histogram     */
297     /*********************************************************************/
298     function chromaHist($bildeUrl, $BinCount)
299     {
300         $im = imagecreatefromjpeg($bildeUrl)
301             or die("oisann, klarte ikke å lage bilde nei");
302         $orgSize = getImageSize($bildeUrl);
303
304         $smallIm = $this->reSize($bildeUrl, 200, 200);
305         $BinDiv = 100 / $BinCount; // potens av 2
306
307         $histogram = array(array());
308         $rgbArr = array();
309
310         for ($x = 0; $x < imagesx($smallIm); $x++)
311         {
312             for ($y = 0; $y < imagesy($smallIm); $y++)
313             {
314                 //Finner fargen til den aktuelle pixelen
315                 $rgb = imageColorAt($smallIm, $x, $y);
316                 //Deler fargen i R, G og B-komponenter
317                 $rgbArr[0] = ($rgb >> 16) & 0xFF;
318                 $rgbArr[1] = ($rgb >> 8) & 0xFF;
319                 $rgbArr[2] = $rgb & 0xFF;
320                 //gjør om til chroma
321                 $chroma = $this->rgb_to_chroma($rgbArr);
322                 //Finner bøttekoordinatene ved å heltallsdividere på $hBinDiv og
                    $svBinDiv
323                 $rgBotte = bcdiv($chroma[0], $BinDiv, 0);
324                 $byBotte = bcdiv($chroma[1], $BinDiv, 0);
325                 //Legger til en i det aktuelle bøttekoordinatet.
326                 $histogram[$rgBotte][$byBotte]++;
327             }//for
328         }//for
329         return $histogram;
330     }//function chromaHist
331
332
333     /*********************************************************************/
334     /*                       fourPartDiv(bildeurl)                       */
335     /*      Divides $bildeUrl in to 4 equal squares and returns these    */
336     /*********************************************************************/
337     function fourPartDiv($bildeUrl)
338     {
339         //print $bildeUrl;
340         /*$srcIm = imagecreatefromjpeg($bildeUrl)
341             or die("oisann, klarte ikke å lage bilde nei");
342         $orgSize = getImageSize($bildeUrl);*/
343         $srcIm = $this->reSize($bildeUrl, 200, 200);
344         $trgtIm = Array();
345         $width = 100;
346         $height = 100;
347
348         for ($i=0; $i<4;$i++)
349         {
350             $trgtIm[$i] = imagecreatetruecolor($width, $height);
351         }
```

```
352             imagecopy($trgtIm[0], $srcIm, 0, 0, 0, 0, $width, $height);
353             imagecopy($trgtIm[1], $srcIm, 0, 0, $width, 0, $width, $height);
354             imagecopy($trgtIm[2], $srcIm, 0, 0, 0, $height, $width, $height);
355             imagecopy($trgtIm[3], $srcIm, 0, 0, $width, $height, $width, $height);
356
357             return $trgtIm;
358         }//function fourPartDiv
359
360
361         /*********************************************************************/
362         /*              fourPartHist(imagearray, bincount)                   */
363         /*  returns $binCount^3 RGB-histograms for all images in $imArr      */
364         /*********************************************************************/
365         function fourPartHist($imArr, $binCount)
366         {
367             $binDiv = 256 / $binCount;
368             $histArr = array(array(array(array())));
369             $pixCount = imagesx($imArr[0]) * imagesy($imArr[0]);
370
371             for ($i=0;$i<count($imArr);$i++)
372             {
373                 for ($x = 0; $x < imagesx($imArr[$i]); $x++)
374                 {
375                     for ($y = 0; $y < imagesy($imArr[$i]); $y++)
376                     {
377                         //Finner fargen til den aktuelle pixelen
378                         $rgb = imageColorAt($imArr[$i], $x, $y);
379                         //Deler fargen i R, G og B-komponenter
380                         $r = ($rgb >> 16) & 0xFF;
381                         $g = ($rgb >> 8) & 0xFF;
382                         $b = $rgb & 0xFF;
383                         //Finner bøttekoordinatene ved å heltallsdividere på $binDiv (som er
                            256 delt
384                         // på antall bøtter)
385                         $rBotte = bcdiv($r, $binDiv, 0);
386                         $gBotte = bcdiv($g, $binDiv, 0);
387                         $bBotte = bcdiv($b, $binDiv, 0);
388                         //Legger til en i det aktuelle bøttekoordinatet.
389                         $histArr[$i][$rBotte][$gBotte][$bBotte]++;
390                     }//for
391                 }//for
392             }//for
393             return $histArr;
394         }//function fourPartHist
395
396
397         /*********************************************************************/
398         /*              sepChanHist(bildeurl, Xbincount, cbincount)          */
399         /*          returns an $xBins*$cBins channelsplit-histogram          */
400         /*********************************************************************/
401         function sepChanHist($bildeUrl, $xBins, $cBins)
402         {
403             $smallIm = $this->reSize($bildeUrl, 200, 200);
404
405             $pixArr =  array(array(array()));
406             $xbinDiv = imagesx($smallIm) / $xBins;
407             $cbinDiv = 256 / $cBins;
408
409             for ($x = 0; $x < imagesx($smallIm); $x++)
410             {
411                 for ($y = 0; $y < imagesy($smallIm); $y++)
412                 {
413                     //Finner fargen til den aktuelle pixelen
414                     $rgb = imageColorAt($smallIm, $x, $y);
415                     //Deler fargen i R, G og B-komponenter
416                     $r = ($rgb >> 16) & 0xFF;
417                     $g = ($rgb >> 8) & 0xFF;
418                     $b = $rgb & 0xFF;
419                     $pixArr[0][bcdiv($x, $xbinDiv, 0)][bcdiv($r, $cbinDiv, 0)]++;
420                     $pixArr[1][bcdiv($x, $xbinDiv, 0)][bcdiv($g, $cbinDiv, 0)]++;
421                     $pixArr[2][bcdiv($x, $xbinDiv, 0)][bcdiv($b, $cbinDiv, 0)]++;
```

```
422                 }//for
423             }//for
424 
425             return $pixArr;
426 
427         }//function sepChanHist
428 
429 
430         /*********************************************************************/
431         /*              tenPartDiv(bildeurl, division count)                 */
432         /*          Divides $bildeUrl in to $divCount equal squares          */
433         /*********************************************************************/
434         function tenPartDiv($bildeUrl, $divCount)
435         {
436             $srcIm = $this->reSize($bildeUrl, 200, 200);
437             $trgtIm = Array();
438             $width = 200/$divCount;
439             $height = 200/$divCount;
440 
441             for ($i=0; $i<$divCount* $divCount;$i++)
442             {
443                 $trgtIm[$i] = imagecreatetruecolor($width, $height);
444             }
445             $j=0;
446             for ($x=0; $x<200; $x+=$width)
447             {
448                 for ($y=0; $y<200; $y+=$height)
449                 {
450                     imagecopy($trgtIm[$j], $srcIm, 0,0, $x, $y, $width, $height);
451                     $j++;
452                 }
453             }
454             return $trgtIm;
455         }//function tenPartDiv
456 
457         /*********************************************************************/
458         /*                        avgColor(image array)                      */
459         /*      returns the average RGB-colour of all the images in $imArr   */
460         /*********************************************************************/
461         function avgColor($imArr)
462         {
463             $avgArr = array(array());
464 
465 
466             for ($i=0; $i <count($imArr);$i++)
467             {
468                 $tempArr = array(array());
469                 $j = 0;
470                 for ($x=0;$x<imagesx($imArr[$i]);$x++)
471                 {
472                     for ($y=0;$y<imagesy($imArr[$i]);$y++)
473                     {
474                         $rgb = imageColorAt($imArr[$i], $x, $y);
475                         $r = ($rgb >> 16) & 0xFF;
476                         $g = ($rgb >> 8) & 0xFF;
477                         $b = $rgb & 0xFF;
478                         $tempArr[0][$j] = $r;
479                         $tempArr[1][$j] = $g;
480                         $tempArr[2][$j] = $b;
481                         $j++;
482                     }
483                 }
484                 $avgArr[$i][0] = round(array_sum($tempArr[0]) / (imagesx($imArr[$i]) *
                    imagesy($imArr[$i])), 0);
485                 $avgArr[$i][1] = round(array_sum($tempArr[1]) / (imagesx($imArr[$i]) *
                    imagesy($imArr[$i])), 0);
486                 $avgArr[$i][2] = round(array_sum($tempArr[2]) / (imagesx($imArr[$i]) *
                    imagesy($imArr[$i])), 0);
487             }
488             return $avgArr;
489         }// function avgColor
```

```
490
491         function avgHist($bildeUrl, $divCount)
492         {
493             $imArr = $this->tenPartDiv($bildeUrl, $divCount);
494             $avgArr = $this->avgColor($imArr);
495             return $avgArr;
496         }// function avgHist
497
498
499         /*********************************************************************/
500         /*              arrMerge(array1, array2, weight1, weight2)           */
501         /*                  merges two arrays with weighting                 */
502         /*********************************************************************/
503         function arrMerge ($arr1, $arr2, $weight1, $weight2)
504         {
505             $tmpArr1 = array();
506             $tmpArr2 = array();
507
508             foreach ($arr1 as $value1)
509                 array_push($tmpArr1, $value1*$weight1);
510             foreach ($arr2 as $value2)
511                 array_push($tmpArr2, $value2*$weight2);
512
513             $retArr = array_merge($tmpArr1, $tmpArr2);
514             return $retArr;
515         }
516
517
518         /*********************************************************************/
519         /*                       imRandomize(bildeurl)                       */
520         /*          Randomly rearranges all the pixels in $bildeUrl          */
521         /*********************************************************************/
522         function imRandomize($bildeUrl)
523         {
524             $smallIm = $this->reSize($bildeUrl, 200, 200);
525             $returnIm = imagecreatetruecolor(200,200);
526             $imArr = array();
527             $i = 0;
528             for ($x = 0; $x < imagesx($smallIm); $x++)
529             {
530                 for ($y = 0; $y < imagesy($smallIm); $y++)
531                 {
532                     //Finner fargen til den aktuelle pixelen
533                     $imArr[$i] = imageColorAt($smallIm, $x, $y);
534                     $i++;
535                 }
536             }
537             shuffle($imArr);
538             $j=0;
539             for ($x = 0; $x < imagesx($smallIm); $x++)
540             {
541                 for ($y = 0; $y < imagesy($smallIm); $y++)
542                 {
543                     imagesetpixel($returnIm, $x, $y, $imArr[$j]);
544                     $j++;
545                 }
546             }
547             return $returnIm;
548         }// function imRandomize
549
550
551         /*********************************************************************/
552         /*                       avgCompare(array1, array2)                  */
553         /*          Computes the vector distance between two colour arrays   */
554         /*********************************************************************/
555         function avgCompare($arr1, $arr2)
556         {
557             $totDiff = 0;
558             $threshold = 10;
559             $simCount = 0;
560             $totCount = count($arr1)/3;
```

```
561             $k = 0;
562 
563             for($i = 0; $i < count($arr1); $i+=3)
564             {
565                 //print "<br /> HER ER JEG!!!!! <br />";
566                 $rdiff = abs($arr1[$i] - $arr2[$i]);
567                 $gdiff = abs($arr1[$i+1] - $arr2[$i+1]);
568                 $bdiff = abs($arr1[$i+2] - $arr2[$i+2]);
569                 $vLength1 = sqrt(pow($arr1[$i], 2) + pow($arr1[$i+1], 2) + pow($arr1[$i+2],
                    2));
570                 $vLength2 = sqrt(pow($arr2[$i], 2) + pow($arr2[$i+1], 2) + pow($arr2[$i+2],
                    2));
571                 $vMax = max($vLength1, $vLength2);
572                 $vDiff[$k] = sqrt(pow($rdiff, 2) + pow($gdiff, 2) + pow($bdiff, 2));
573                 $totDiff += $vDiff[$k];
574                 $k++;
575             }
576             return $totDiff;
577         }//function avgcompare
578     }//class bildeTing
579 
580 ?>
```

Appendix B

PHP-code for one indexing- and one retrieval-script.

```php
<html>
<head>
    <link rel="stylesheet" type="text/css" href="gr1.css" />
</head>
<body>

<?php
    /*************************************************************/
    /*              CBIR-script for RGB-histograms               */
    /*************************************************************/

    require '../classes/dbClass.php';
    require '../classes/bildeTing.php';

    $dbObj = new db;
    $dbObj->koble_til();
    $btObj = new bildeTing;
    $sb = $_GET['sb'];

    $sokDir = '/media/bildedb/mpeg_dataset/';
    $sokBilde = $sokDir.$sb;

    $query = "SELECT * FROM Histogram";
    $query2 = "SELECT Hist FROM Histogram WHERE Filnavn = '".$sokBilde."'";
    $result = array();


    $qRes2 = $dbObj->query($query2);
    $qObj2 = mysql_fetch_object($qRes2);
    $qArr2 = explode('|', $qObj2->Hist);

    $qRes = $dbObj->query($query);

    //This is where the search itself is performed.
    while ($qObj = mysql_fetch_object($qRes))
    {
        $qArr = explode('|', $qObj->Hist);

        $svarL1 = $btObj->gatekryssAvstandsBeregning($qArr, $qArr2);
        $svarL2 = $btObj->L2($qArr, $qArr2);

        $resultL1[$qObj->Filnavn] = $svarL1;
        $resultL2[$qObj->Filnavn] = $svarL2;
    }

    print ('<center>');
    print ('Søkebilde: <br />');
    print ('<IMG SRC="bilde.php?pic='.$sokBilde.'&width=150" ALT="noko er gale"/><br />');
    print ($sokBilde.'<br />');
    print ('</center>');

    asort($resultL1);
    asort($resultL2);

    print ('<br />Resultat: <br />');
    $keysL1 = array_keys($resultL1);
    $keysL2 = array_keys($resultL2);

    print '<table class="search"><tr><th class="meny" colspan="2"> L1: </th><th
    class="meny" colspan="2"> L2: </th></tr>';

    for($i = 0; $i < 20; $i++)
    {
        $tmp = explode($sokDir, $keysL1[$i]);
        $tmp11 = explode($sokDir, $keysL1[$i+1]);
        $tmp2 = explode($sokDir, $keysL2[$i]);
        $tmp21 = explode($sokDir, $keysL2[$i+1]);
        print ('<tr><td class="search">');
        print ('<A href=cbir.php?sb='.$tmp[1].'><IMG
        SRC="bilde.php?pic='.$keysL1[$i].'&width=150" ALT="noko er gale"/></A><br />');
        print ('<A href=histvis.php?sb='.$tmp[1].'>Forskjell:
        '.$resultL1[$keysL1[$i]].'<br />');
        print ('<A href=excel.php?sb='.$tmp[1].'>'.$tmp[1].'</A></td>');
        print ('<td class="search">');
        print ('<A href=cbir.php?sb='.$tmp11[1].'><IMG
        SRC="bilde.php?pic='.$keysL1[$i+1].'&width=150" ALT="noko er gale"/></A><br />');
```

```
73          print ('<A href=histvis.php?sb='.$tmp11[1].'>Forskjell:
            '.$resultL1[$keysL1[$i+1]].'<br />');
74          print ('<A href=excel.php?sb='.$tmp11[1].'>'.$tmp11[1].'</A></td>');
75          print ('<td class="search">');
76          print ('<A href=cbir.php?sb='.$tmp2[1].'><IMG
            SRC="bilde.php?pic='.$keysL2[$i].'&width=150" ALT="noko er gale"/></A><br />');
77          print ('<A href=histvis.php?sb='.$tmp2[1].'>Forskjell:
            '.$resultL2[$keysL2[$i]].'<br />');
78          print ('<A href=excel.php?sb='.$tmp2[1].'>'.$tmp2[1].'</A></td>');
79          print ('<td class="search">');
80          print ('<A href=cbir.php?sb='.$tmp21[1].'><IMG
            SRC="bilde.php?pic='.$keysL2[$i+1].'&width=150" ALT="noko er gale"/></A><br />');
81          print ('<A href=histvis.php?sb='.$tmp21[1].'>Forskjell:
            '.$resultL2[$keysL2[$i+1]].'<br />');
82          print ('<A href=excel.php?sb='.$tmp21[1].'>'.$tmp21[1].'</A></td></tr>');
83          $i++;
84      }
85  ?>
86  </body>
87  </html>
88
89  <?php
90      /*********************************************************************/
91      /*          This is the script for creating RGB-histograms           */
92      /*********************************************************************/
93      require '../classes/dbClass.php';
94      require '../classes/bildeTing.php';
95
96      $dbObj = new db;
97      $dbObj->koble_til();
98      $btObj = new bildeTing;
99
100     $startTime = time();
101     $sokBotter = 8;
102     $bildeDir = '/media/bildedb/mpeg_dataset/';
103
104
105     if ($handle = opendir($bildeDir))
106     {
107        while (false !== ($file = readdir($handle)))
108        {
109             if ($file != "." && $file != "..")
110             {
111                 //echo "$file\n";
112                 $histStr = '';
113                 $fullFile = $bildeDir.$file;
114                 $hist = $btObj->histogram($fullFile, $sokBotter);
115                 for ($r = 0; $r < $sokBotter; $r++)
116                 {
117                     for ($g = 0; $g < $sokBotter; $g++)
118                     {
119                         for ($b = 0; $b < $sokBotter; $b++)
120                         {
121                             //legger verdien av bøtten til histStr
122                             $histStr.=$hist[$r][$g][$b];
123                             //Hvis det ikke er siste bøtten, legg til en |
124                             if (!($r == $b && $b == $g && $g == ($sokBotter - 1)))
125                             {
126                                 $histStr.='|';
127                             }
128
129                         }
130                     }
131                 }
132                 $dbObj->query("INSERT INTO Histogram (Filnavn, Hist) VALUES ('$fullFile',
                    '$histStr')");
133             }
134             echo "-";
135         }
136         closedir($handle);
137
138         $endTime = time();
139         $elapsedTime = $endTime - $startTime;
140         print ("Tid brukt: ".$elapsedTime." sekunder\n");
141     }
142 ?>
```

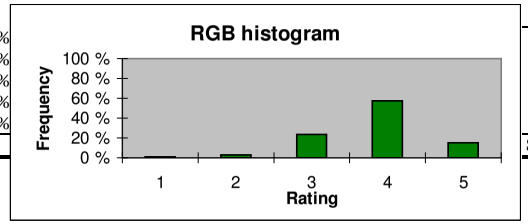
Appendix C

All results of the questionnaire described in chapter 4.2.

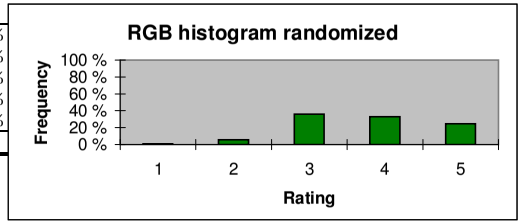
id	bra1	bra2	bra3	bra5	bra6	bra7	rank1	rank2	rank3	rank4	rank5	rank6	sex	age	comments
1	4	5	3	5	3	3	2	1	3	1	2	3	k	26	
2	3	3	3	4	4	4	2	3	1	1	2	3	k	25	
4	5	4	5	5	5	5	2	3	1	1	1	1	m	29	synes alle søkene ga imponerende resultater jeg. God jobb :)"
5	4	3	5	3	3	3	2	3	1	2	2	2	m	25	
6	5	4	4	3	3	3	2	3	1	1	2	3	k	27	
8	4	3	2	4	5	4	1	2	3	2	1	3	k	27	
9	4	3	4	3	4	3	2	3	1	2	1	3	m	35	
10	4	3	3	4	4	2	1	3	2	2	1	3	m	40	
11	4	3	4	4	3	4	3	2	1	2	1	3	m	26	Eventuelle kommentarer her.
14	4	3	4	2	2	2	1	3	2	1	2	3	M	25	
15	5	4	5	5	5	5	2	3	1	1	2	3	m	22	
20	4	3	4	3	4	2	1	2	3	2	1	3	m	25	Halla Ole A
17	4	2	4	3	1	2	3	2	1	2	1	3	m	24	
19	3	3	4	3	4	4	2	3	1	3	2	1	k	23	
23	4	4	4	3	3	3	1	2	3	1	2	3	m	24	Min rangering av metodene kan du egentlig se fullstendig bort fra. De første treffene er etter hva jeg kan se temmelig like, uavhengig av metode, de store forskjellene kommer først på de siste 10 treffene...og et urelevant bilde er urelevant uansett om det er bilde av en elefant eller en mus. Siden metodene resulterer i så like resultater burde dette vært presentert på en annen måte for av vi som svarer på en slik undersøkelse skal ha mulighet for å vurdere forskjellene. Det blir for mye jobb og gå frem og tilbake mellom metode 1, 2 og 3 og scrolle opp og ned. Dersom du kunne presentere tre metoder på hver side...med bildene i tre rader...så ville det vært mye enklere å sammenligne. Du har uansett valgt en spennende oppgave, og jeg ser frem til presentasjonen - men slik denne spørreundersøkelsen er lagt opp er mitt svar verdiløst, da jeg ikke har mulighet til å skille resultatene fra hverandre når de presenteres på denne måten.
24	5	3	3	3	3	2	1	2	3	1	2	3	k	23	
26	4	4	5	5	5	4	3	2	1	1	2	3	m	25	Det kan jo aldri bli perfekt da, hvis du skal få en del treff... Hva med å utvide søkebasen din med et søk i google-bilder?
27	3	4	5	3	4	3	3	2	1	2	1	3	m	25	
29	4	4	4	4	4	3	3	2	1	2	1	3	m	27	
33	4	4	5	5	4	4	2	3	1	1	2	3	m	24	
34	3	3	3	4	4	4	2	3	1	2	3	1	k	28	
35	4	3	5	5	4	5	2	3	1	1	2	1	m	26	
36	3	2	2	4	3	2	1	3	2	1	2	3	K	23	
38	5	4	5	5	5	4	2	3	1	1	2	3	m	21	På den siste så søkte den etter søy, og fant støy. Jeg synes den gjorde det veldig bra, den fant jo bare støybilder... ;) Likte at de støybildene med størst forskjell i luminans og farge fra søkebildet havnet til sist...
39	3	4	5	3	4	5	3	2	1	3	2	1	k	23	Dette var et spennede prosjekt. Gleder meg til å høre om resultatet. Kanskje du kan revosjonere bildesøk:) Rangeringen min ser du jo over, jeg mener at nr 3 og 6 gav likest visuelle resultater.
44	4	3	3	3	4	2	1	2	3	1	2	3	m	26	
48	4	5	4	5	5	4	2	1	3	1	2	3	m	21	
50	4	3	4	3	4	2	1	3	2	2	1	3	K	24	Dette er en meget god idé, og det er et fremskritt hvis dette systemet «foredles»! Særlig nyttig ved søk etter et spesielt utrykk i et bilde man leter etter."
52	5	4	3	4	4	4	2	1	3	1	2	3	m	25	Er litt usikker på om du får noe godt svar ut av denne testen.
53	3	3	2	3	3	3	2	2	2	2	1	2	k	23	
57	3	2	3	2	2	1	1	3	2	1	2	3	k	20	
59	4	4	4	5	5	5	3	2	1	1	2	3	k	24	
60	4	4	4	3	3	4	1	3	2	2	1	3	k	24	
61	3	3	3	3	4	4	2	3	1	3	2	1	m	21	
62	4	4	5	5	5	3	2	3	1	1	2	3	m	20	
66	4	3	4	3	3	3	2	3	1	1	2	3	m	29	
67	4	3	4	3	3	2	1	3	2	1	2	3	m	24	spennende
69	4	3	4	4	4	3	2	3	1	2	1	3	m	24	
70	5	5	5	4	4	3	2	3	1	1	2	3	m	21	spennende prosjekt!
71	4	3	4	3	4	3	2	2	1	2	1	3	K	29	
72	4	4	5	4	3	3	2	3	1	1	2	3	k	25	
75	3	2	2	4	3	2	1	3	2	1	2	3	m	24	
76	4	3	4	5	5	5	2	3	1	1	2	3	k	22	
77	4	3	4	5	4	3	1	3	2	1	2	3	m	22	Dette er en teknologi jeg har ventet lenge på. Bra jobba.
78	3	3	4	4	3	5	2	3	1	2	3	1	k	25	
80	3	2	4	2	3	4	2	3	1	3	2	1	k	21	
81	4	4	5	5	5	5	2	3	1	1	2	3	K	41	
82	5	4	5	4	3	3	2	3	1	1	2	3	k	20	litt vanskelig å sammenligne hvilke søk som var best når man ikke så de samtidig
83	3	2	2	2	2	1	1	2	3	1	2	3	m	21	
84	4	3	3	5	5	4	1	3	2	2	1	3	m	23	
86	3	3	4	4	4	3	1	3	2	2	1	3	m	23	
91	4	4	4	3	4	4	2	3	1	3	2	1	m	25	interessant!
92	4	3	3	4	4	3	1	3	2	1	2	3	k	35	

93	4	4	5	5	5	5	2	3	1	3	2	1	k	22	Ikke alltid lett å se så mye forskjell på søkene. Spesielt bildene av sand ble veldig like for meg.
94	4	5	5	4	4	3	3	2	1	2	1	3	k	23	
96	4	3	4	1	1	1	1	2	3	1	2	3	k	25	
97	4	4	5	4	5	4	3	2	1	2	1	3	K	23	
98	5	3	3	3	4	2	1	2	3	2	1	3	m	23	
100	4	4	3	4	4	3	1	2	3	1	2	3	m	23	
101	4	4	2	3	3	2	1	2	3	1	2	3	m	23	
102	4	3	3	3	4	2	1	2	3	2	1	3	m	25	
104	5	3	3	3	4	3	1	3	2	1	3	2	m	22	
108	4	3	3	3	4	2	1	3	2	1	2	3	k	25	
109	2	3	4	3	3	3	3	2	1	1	2	3	m	22	
110	4	4	3	5	4	3	2	1	3	1	2	3	k	23	dette var vanskelig og uoversiktlig å svare på. muligens fordi jg aldri har gjort noe sånt før...
111	3	2	3	4	4	5	1	2	3	3	1	2	k	22	
112	5	4	5	5	5	3	2	3	1	2	1	3	m	24	
113	5	4	4	3	2	1	1	2	3	1	2	3	K	20	
114	5	5	5	4	5	4	2	3	1	2	1	3	m	24	
115	4	3	3	4	4	3	1	3	2	1	2	3	m	44	
117	3	3	4	5	5	4	3	2	1	1	1	2	k	21	
122	4	3	3	4	4	3	1	2	3	2	1	3	k	22	
124	5	5	5	4	3	3	3	2	1	1	2	3	m	22	VELDIG vanskelig å skille bildene i metode 4,5,6.
125	4	4	5	4	3	4	1	3	2	1	3	2	m	35	Litt vanskelig å bestemme det beste resultatet ut fra bildene i metode 4, 5 eller 6, men gjorde ihvertfall et forsøk.
128	4	3	3	5	5	5	1	3	2	1	2	3	m	36	Spennende saker
130	3	3	3	4	2	4	1	2	3	1	3	2	k	25	
131	3	2	3	3	2	2	2	3	1	1	3	2	m	24	
132	4	4	3	5	4	4	1	2	3	1	3	2	m	26	Jeg synes den med oksen var finest!
133	4	3	4	5	4	5	1	3	2	1	2	1	m	22	
135	4	4	3	3	3	3	1	2	3	2	3	1	k	26	
137	4	4	4	4	4	4	3	2	1	2	3	1	m	20	
138	3	3	3	4	4	3	1	2	3	1	2	3	m	20	
140	5	4	3	4	5	2	1	2	3	2	1	3	m	43	
141	3	3	2	3	3	3	3	1	2	3	2	1	k	29	det siste aner jeg ikke hva du er ute etter
144	4	3	3	4	3	3	1	2	3	1	2	3	m	23	
147	4	4	4	5	5	5	3	2	1	2	1	3	m	27	
149	4	4	4	3	3	3	1	3	2	2	1	3	m	30	
150	4	3	3	3	3	2	1	2	3	2	1	3	m	23	
151	4	4	4	5	4	4	2	3	1	1	2	3	M	22	
153	4	2	2	5	5	5	1	2	3	1	2	3	m	23	Skikkelig tøft å se at den velger bildene som er like. Aldri sett noe lignende.
154	3	2	1	2	2	2	1	2	3	2	1	2	m	23	
160	1	1	2	3	3	3	3	2	1	3	2	1	K	22	
163	4	4	4	3	4	3	2	3	1	2	1	3	k	23	
168	4	3	3	4	3	3	1	2	3	1	2	3	K	42	Hei! Har gjennomført testen din. Flott tiltak selv om det kanskje ikke er det jeg kommer til å bruke mest tid på. Lykke til videre! :)"
169	2	2	1	4	4	3	1	2	3	2	1	3	m	25	
170	4	4	4	3	3	3	1	3	2	1	2	3	k	24	
171	4	3	4	4	4	3	2	3	1	1	2	3	m	20	Smart måte å søke etter bilder. Blir noen merkelige bilder ut i fra søkebildet, skjønner ikke hvordan de kan dukke opp, har jo ikke noe tilfelles med søkebildet :) Tror du er på god vei til å lage noe bra, lykke til :)
175	3	3	3	3	2	4	3	2	1	3	2	1	m	25	
176	2	1	2	2	2	2	1	3	2	1	2	3	m	24	Fungerer relativt brukbart men har noen ekstremverdier som ødelegger helhetsinntrykket. Bør kanskje fokusere mer på mønstergjenkjenning i tillegg til det som nå ser ut som fargegjenkjenning.
177	3	3	3	3	3	3	3	2	1	1	2	3	m	25	
178	4	3	5	3	4	5	2	3	1	2	3	1	m	24	
180	5	3	3	5	5	5	1	2	3	1	1	1	k	23	
183	4	3	2	5	5	5	1	2	3	3	1	2	k	32	
184	3	4	4	3	3	3	3	2	1	1	2	3	m	24	
185	3	3	4	4	4	2	3	2	1	1	2	3	m	22	
186	4	3	4	4	4	3	1	3	2	2	3	1	m	30	

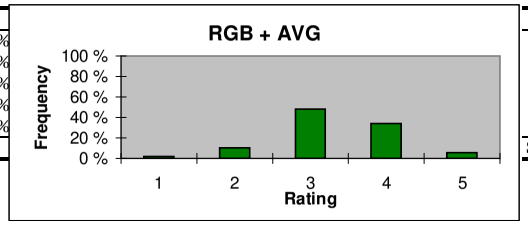
Bin	*Frequency*	%	Cum %
1	1	1 %	1 %
2	3	3 %	4 %
3	25	24 %	27 %
4	61	58 %	85 %
5	16	15 %	100 %
Sum	106		
Average:		3,8	



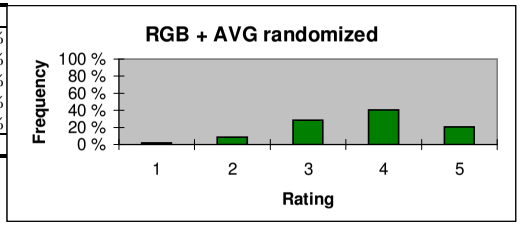
Bin	*Frequency*	%	Cum %
1	1	1 %	1 %
2	6	6 %	7 %
3	38	36 %	42 %
4	35	33 %	75 %
5	26	25 %	100 %
Sum	106		
		3,7	



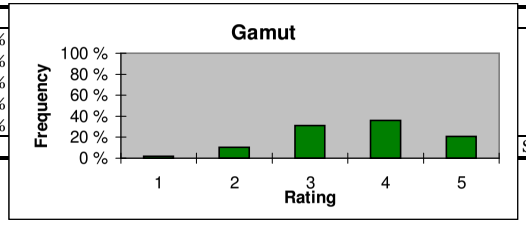
Bin	*Frequency*	%	Cum %
1	2	2 %	2 %
2	11	10 %	12 %
3	51	48 %	60 %
4	36	34 %	94 %
5	6	6 %	100 %
Sum	106		
		3,3	



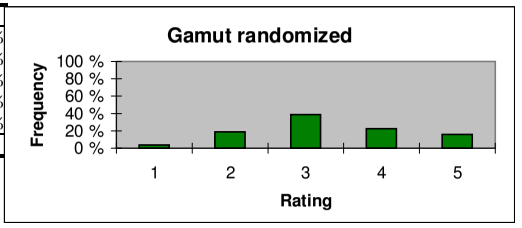
Bin	*Frequency*	%	Cum %
1	2	2 %	2 %
2	9	8 %	10 %
3	30	28 %	39 %
4	43	41 %	79 %
5	22	21 %	100 %
Sum	106		
		3,7	



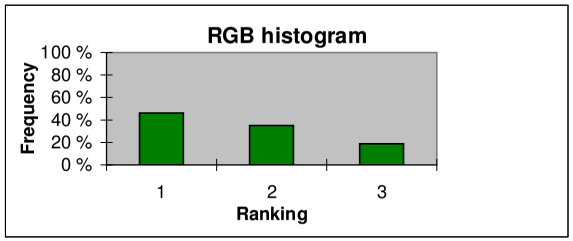
Bin	*Frequency*	%	Cum %
1	2	2 %	2 %
2	11	10 %	12 %
3	33	31 %	43 %
4	38	36 %	79 %
5	22	21 %	100 %
Sum	106		
		3,6	



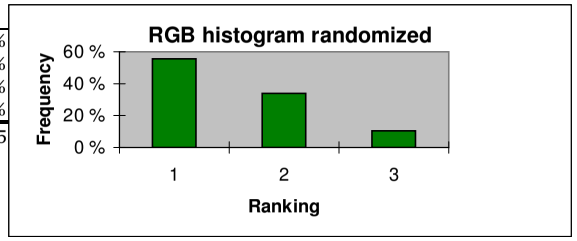
Bin	*Frequency*	%	Cum %
1	4	4 %	4 %
2	20	19 %	23 %
3	41	39 %	61 %
4	24	23 %	84 %
5	17	16 %	100 %
Sum	106		
		3,3	



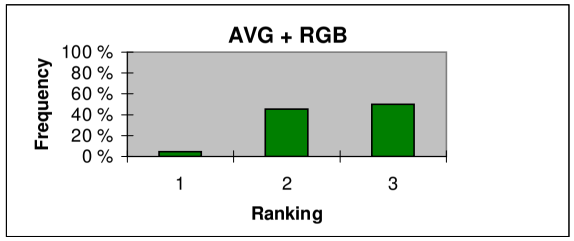
Bin	Frequency	%
1	49	46 %
2	37	35 %
3	20	19 %
Sum	106	100 %
		1,7



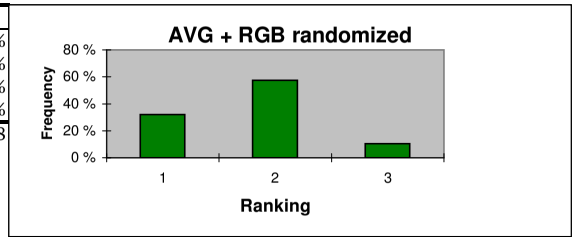
Bin	Frequency	%
1	59	56 %
2	36	34 %
3	11	10 %
Sum	106	100 %
		1,5



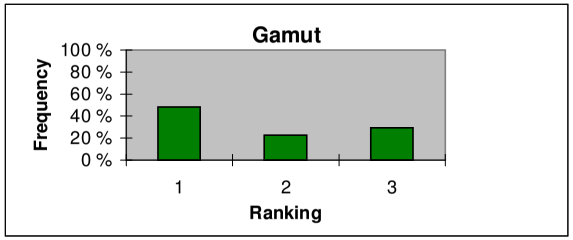
Bin	Frequency	%
1	5	5 %
2	48	45 %
3	53	50 %
Sum	106	100 %
		2,5



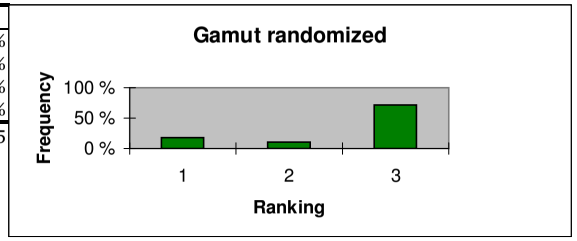
Bin	Frequency	%
1	34	32 %
2	61	58 %
3	11	10 %
Sum	106	100 %
		1,8



Bin	Frequency	%
1	51	48 %
2	24	23 %
3	31	29 %
Sum	106	100 %
		1,8



Bin	Frequency	%
1	19	18 %
2	11	10 %
3	76	72 %
Sum	106	100 %
		2,5



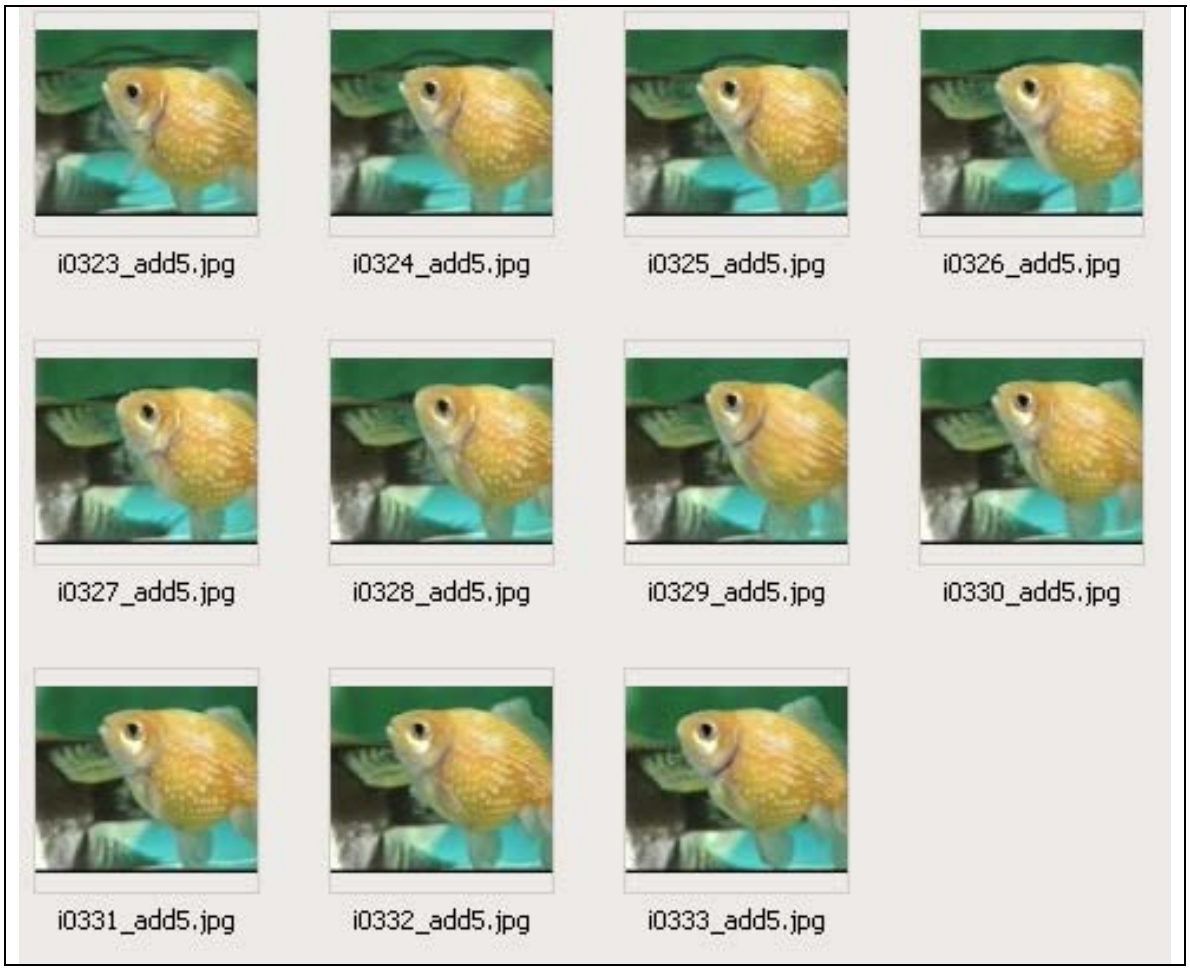
Appendix D

All ground-truth sets used in the project.

Appendix D: Ground truth sets

Appendix D: Ground truth sets

Appendix D: Ground truth sets



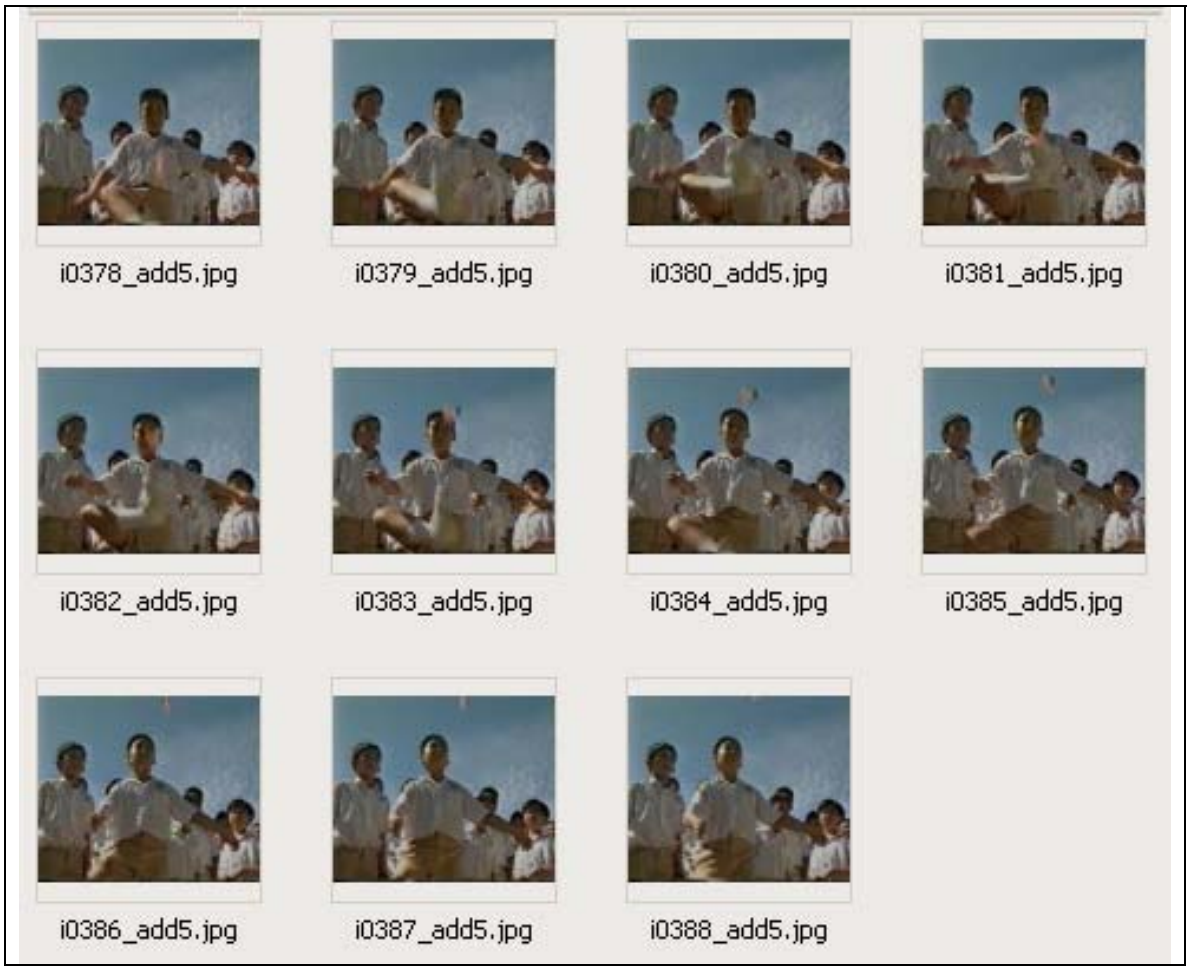
Appendix D: Ground truth sets

Appendix D: Ground truth sets

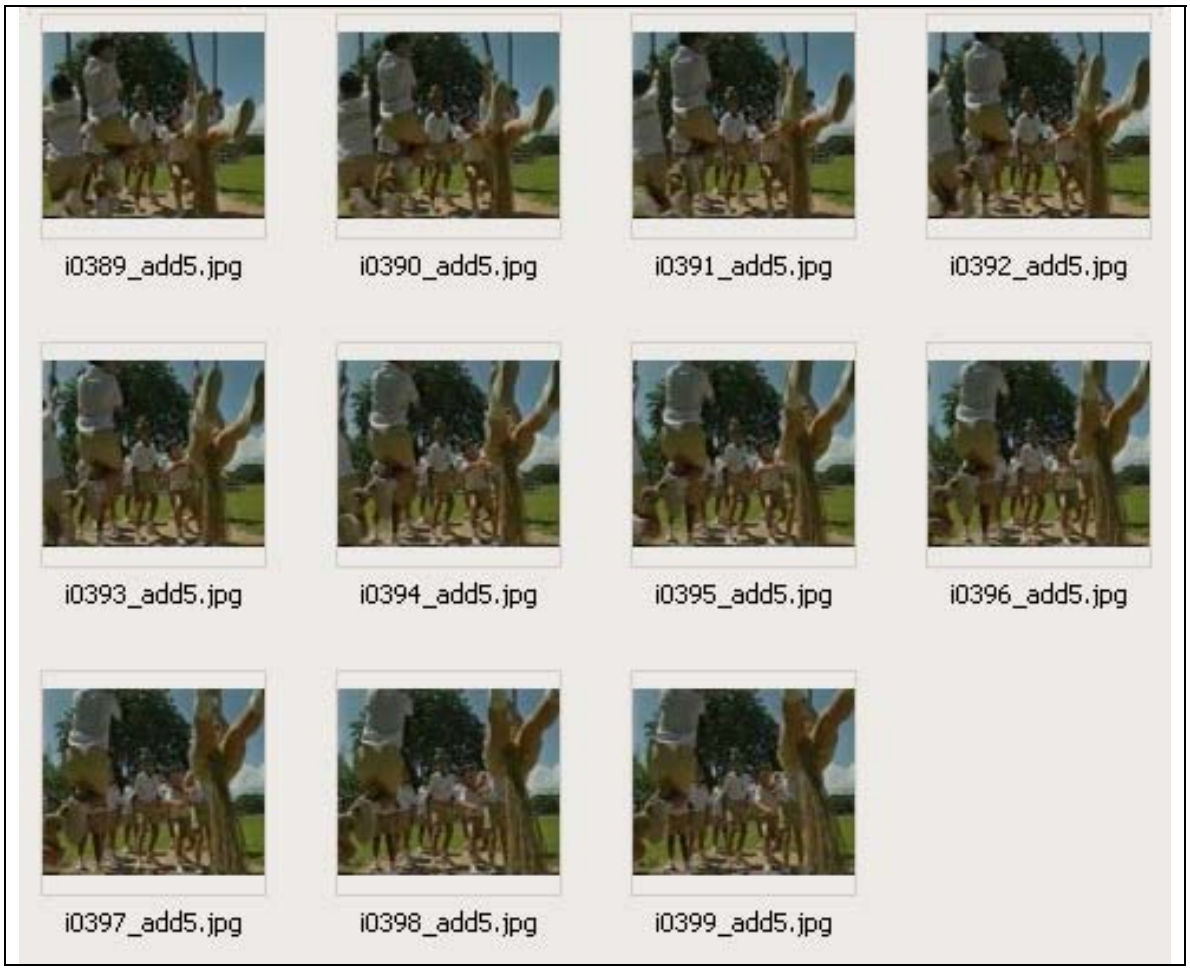
Appendix D: Ground truth sets

Appendix D: Ground truth sets

Appendix D: Ground truth sets

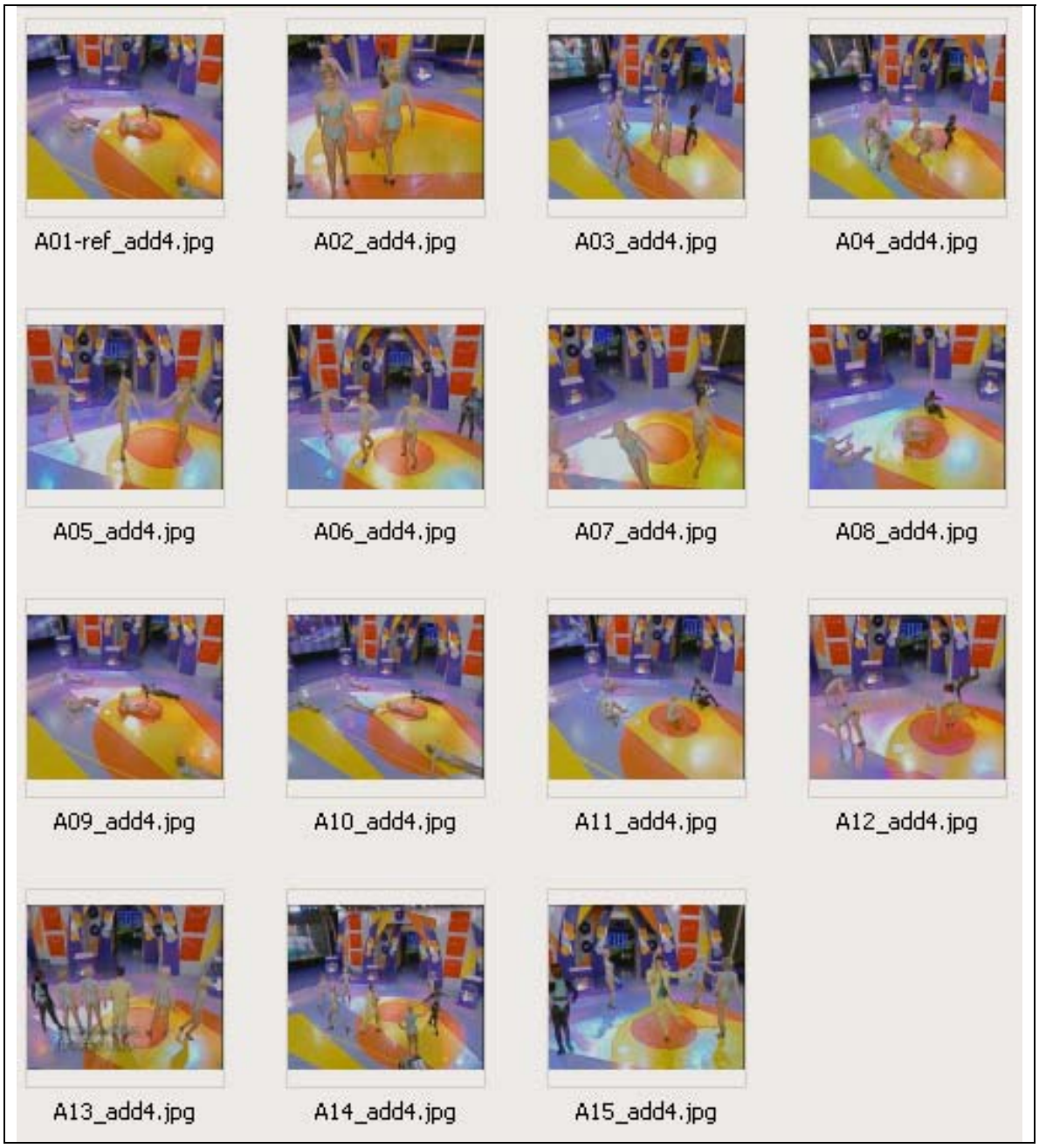


Appendix D: Ground truth sets

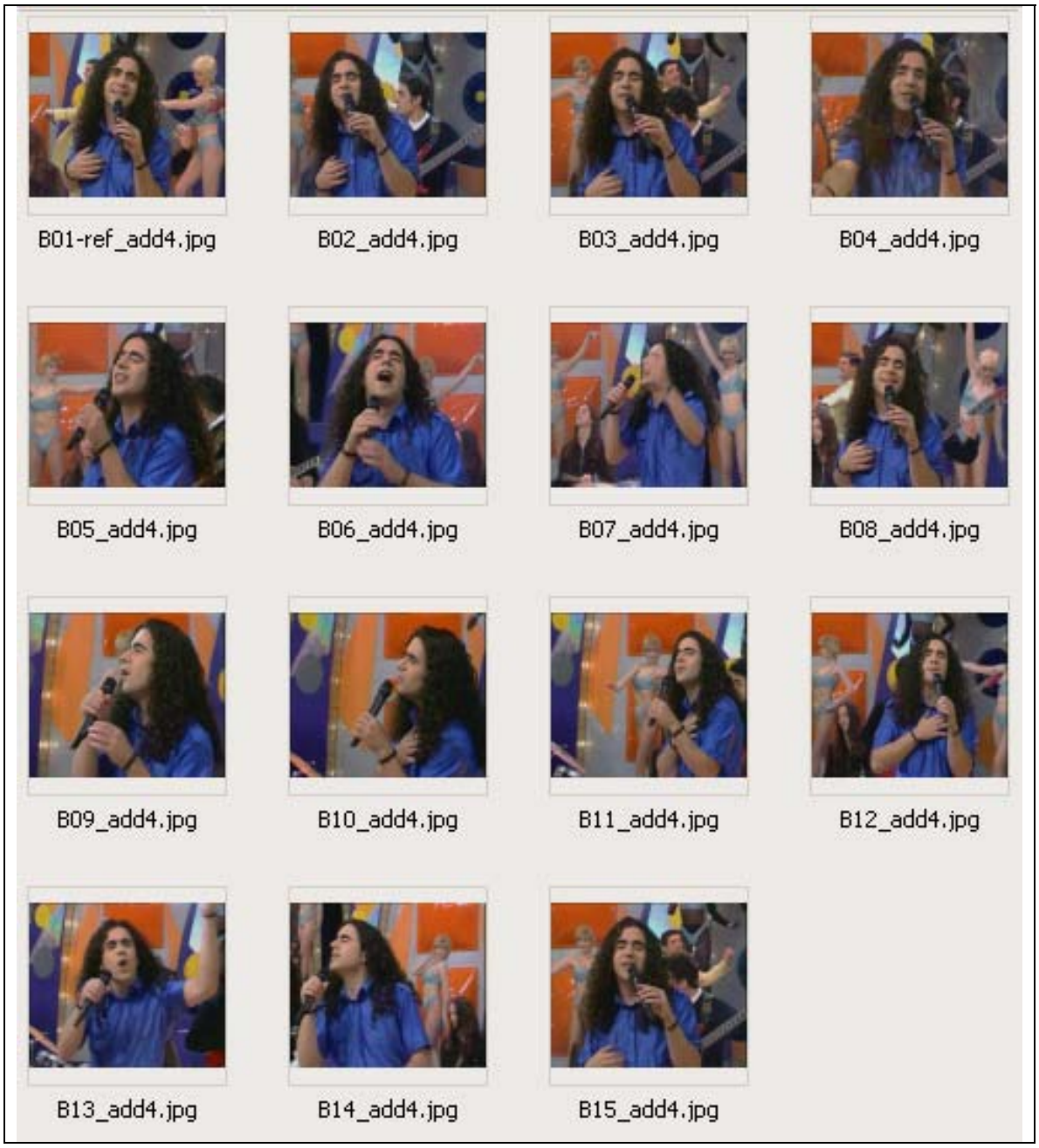


Appendix D: Ground truth sets

Appendix D: Ground truth sets



Appendix D: Ground truth sets



Appendix D: Ground truth sets

Appendix D: Ground truth sets

Appendix D: Ground truth sets