UiB | Dept. of Information Science and Media Studies
UiTø | Dept. of Computer Science
NTNU | Dept. of Computer and Information Science
Telenor | Telenor R&I

CAIM
CONTEXT-AWARE IMAGE MANAGEMENT

VISI3

Context Aware Image Retrieval

Christoph Carlson

Sept. 2009

CAIM-UiB

TR-8

Table of Contents

Abstract

The project's main objective this summer was to translate the entire web application VISI2 (Vortex Image Search Interface 2) from ColdFusion, a proprietary tag-based language developed by Adobe, into Java, using JSPs and Servlets. The back-end of the application contains an Oracle 10g database. In VISI2, TBIR (Text-Based Image Retrieval) makes use of a full-text search engine called Verity used by ColdFusion. Replacing ColdFusion code with Java code led to another sub-goal, which was to replace Verity with Oracle's internal full-text search. Another goal of the project was to improve the CBIR/TBIR (Content-Based Image Retrieval / Text-Based Image Retrieval) search by using the CAIRANK algorithm (Hartvedt, 2007). Testing was also a major part of this summer's project. Automated precision tests for CBIR and CBIR/GPS searches were developed that tested all the images in the database and wrote the results to csv-files. This also helped in the process of finding the weights (shape, color, texture, spatial structure) which gave the best results for our domain.

In addition to the primary project goals, some new functionality was developed; a thumbnail generator, tile generator, more information about the image and a random start-up seed image.

1. Project goals

1.1 Translate from Coldfusion to Java

VISI (Næss, 2007) and VISI2 (Rørvik, 2008) were written in the proprietary scripting language ColdFusion. The primary reasons for this code translation was that ColdFusion is a proprietary language and that the mobile application MMIR2 (Hellevang, 2008) is already written in Java. Another reason was that it will hopefully be easier to continue development since most future project assistants are expected to be familiar with Java..

1.2 Full-text search from Verity to Oracle

Replacement of the ColdFusion code with Java code meant that the Verity Search for TBIR needed to be replaced. Since Oracle has built-in text search technology in all Oracle Database editions, it was decided to use this instead of Verity. Oracle Text has support for the Boolean operators AND, OR, NOT, NEAR etc. It also contains support for advanced features like

fuzzy search.

Each object in the database has a description field stored as a CLOB (Character Long Object). Since this column contains the most information about the object, it was chosen for the text index for the object. Implementation details follow in a later chapter.

1.3 CBIR/TBIR to use CAIRANK algorithm

The CBIR/TBIR search in VISI2 returned result-sets consisting of the results from the TBIR search followed by the results from the CBIR Search. This led to a request for use of a better CBIR/TBIR algorithm, known as the CAIRANK algorithm, developed by Christian Hartvedt. (Hartvedt, 2007)

1.4 Precision testing

A major part of this summer's task was to develop precision tests to see how accurate the system was. Results and details follow in chapter 3.

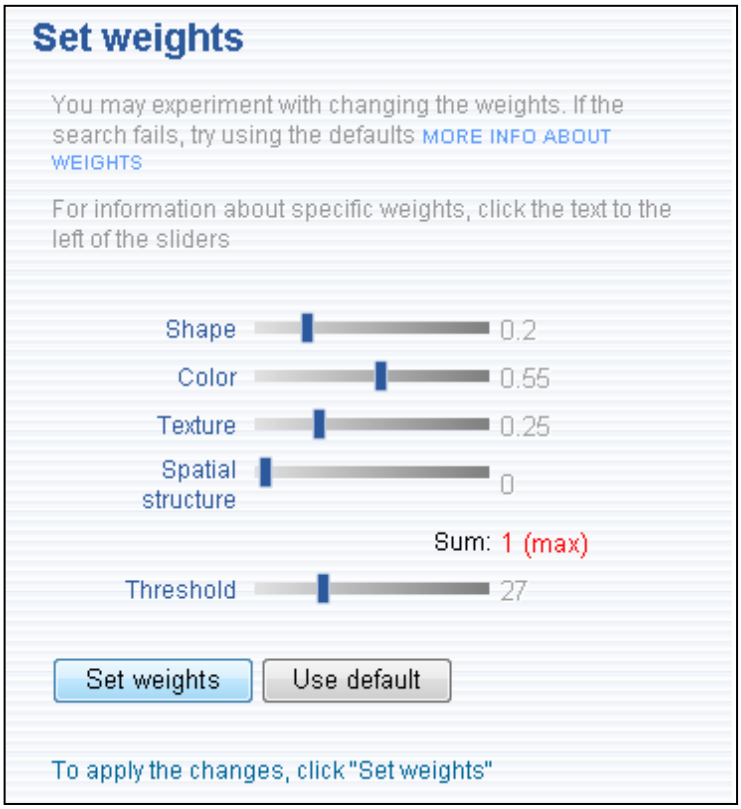
2. Prototype documentation

2.1 Technologies used

2.1.1 Servlets

A servlet is an instance of a Java class that implements the Servlet interface from the javax.servlet package. It receives a request and dynamically generates a response.

An example is shown in Screenshot 1 and Figure 1 below for the SetWeightsServlet which is called from search.jsp when the Set weights-button is selected.



Screenshot 1 - Set weights interface

```
public void doPost(HttpServletRequest request, HttpServletResponse
response) {
	// Set weights
	request.getSession().setAttribute("shape",
		request.getParameter("shapeSliderDiv_value"));
	request.getSession().setAttribute("spatial",
		request.getParameter("spatialStructureSliderDiv_value"));
	request.getSession().setAttribute("texture",
		request.getParameter("textureSliderDiv_value"));
	request.getSession().setAttribute("color",
		request.getParameter("colorSliderDiv_value"));
	request.getSession().setAttribute("threshold",
		request.getParameter("thresholdSliderDiv_value"));
	// Redirects user back to search.jsp
	response.sendRedirect("../jsp/search.jsp?a=visible");
```

Figure 1 - The set weights servlet

Servlet mapping is taken care of by web.xml in the WebContent folder:

```
<servlet-mapping>
	<servlet-name>SetWeightsServlet</servlet-name>
	<url-pattern>/handlers/SetWeightsServlet</url-pattern>
</servlet-mapping>
```

2.1.2 JavaServerPages

JSP is a server side technology based on Java, which allows software developers to create dynamic web pages. JSPs can be seen of as a high-level abstraction of a servlet, and are compiled into servlets by a JSP compiler. JSPs are probably the simplest way of presenting dynamic content mixed with HTML and better to use for presenting content than e.g. a servlet. Pure java code can be embedded in JSP by putting it between <% and %>

Example of setting the request's character encoding to UTF-8:

```
<% request.setCharacterEncoding("UTF-8"); %>
<html>
	<head>
		<title>VISI3 - The VORTEX Image Search Interface</title>
	</head>
```

You can also use expressions ${} , e.g. to print a parameter or a variable:

```
<p>Your name is ${param.name}</p>
```

2.1.3 JSP Standard Tag Libraries/Custom tag libraries

In a JSP, other tags than HTML can be included by combining the JSP Standard Tag Libraries and custom-made tag libraries. The JSP Standard Tag Libraries (JSTL) contain a tag collection which offers some commonly used functionality like xml-parsing, loops and conditions.

First, a reference to the taglib must be given:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

Then a tag, for example a conditional test, can be used:

```
<c:if test="${fn:length(error) > 0}">
	<c:redirect url="../error.jsp?type=${error}"/>
```

```
</c:if>
```

This code checks if the length of the variable error is larger than 0. If it is, the user is redirected to the error page.

If the standard taglibs do not have the required functionality a supplementary tag library can be created. The way it has been done in VISI3, is to make classes that extend SimpleTagSupport, and the doTag() method is called every time the new tag is used. When a tag has been made, one needs to make a .tld-file with a reference to the tag.

```
<%@ taglib prefix="caimdb" uri="/WEB-INF/tld/caimdb.tld" %>
```

2.2 Design

The code for VISI3 has been separated into three main parts: front-end (view), business logic (controller) and data model (model).

2.2.1 Data model

The data model consists of data access objects or wrapper objects. In VISI3, the data access objects are used to provide an abstract interface to the Image table, Object table and the Tile table in the database. These can later be used to extract information and present it in the front-end layer, so that the client does not have to have specific knowledge about the database.

An example of the usage of ImageMetadataDAO from image.jsp:

```
<caimdb:getImageMetadata var="metadata" imageID="${param.imageID}"/>
<table>
      <tr><td
class="key">Caption</td><td>${metadata.caption}</td></tr>
</table>
```

getImageMetadata from the caimdb taglib will fetch the ImageMetadataDAO with the imageID specified in the attribute and set it to the variable name specified. Then calling ${metadata.caption} will call the getCaption() method in ImageMetadataDAO

2.2.2 Business logic

The business logic layer consists of customized tags, servlets and other java classes. The source code is found under the VISI3\WebContent\WEB-INF\src folder[1].

One example of a business logic class is the DBHandler class in the VISI3's handlers.db package. Creating an object of this class, enables the user to connect to the database. This is also where you have to specify the username, password and host (These can be obtained from the CAIM administrator). It would have been better to do this in a config file, but because of time constraints and since this is a prototype this was not done.

```
public Connection createConnection() throws InstantiationException,
IllegalAccessException, ClassNotFoundException, SQLException {
      Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
      Connection conn =
      DriverManager.getConnection("jdbc:oracle:thin:@"+hostname,
      username, password);
      return conn;
```

2.2.3 Front-end

The front-end layer consists of HTML, JSP, CSS and JavaScript code. This part of the code will be found under the WebContent folder. This is what the user will see in his/her web browser.

2.3 Search functionality

Each of the different searches (CBIR, TBIR etc) has it's own java class, which contains the method performSearch. This method returns the set of imageIDs, represented as a java array. The tag performSearch in caimdb.tld separates between the different searches, depending on the attribute value «searchtype».

2.3.1 Content-Based Image Retrieval search

The CBIR search uses a seed image, compares the image signature to the images in the database and returns a list of the images that are most alike based on their signature. The parameters it should prioritize can be altered by adjusting the weights texture, color, shape and spatial structure. The threshold will determine how many images are returned in the result-set. If a threshold of 25 has been chosen, every image with a better score than 25 will be be returned. The CBIR search used to have its own Oracle PL/SQL procedure. This seemed to be a problem, since it gave different results than the mobile application MMIR3. VISI3

[1] Access to the VISI3 source code is through the CAIM/UiB administrator, Joan Nordbotten

now uses the same procedure as MMIR3 (Hellevang, 2009). The main difference is that the gps parameter is set to -1 , when called from VISI3. This is because the mobile application needs to rescale images, and this time-consuming process is not necessary from VISI3.

An example of a CBIR search could be:

```
<caimdb:performSearch searchtype="cbir" filename="holberg.jpeg"
var="imageIDs" captiontable="captions" color="0.4" texture="0.3"
shape="0.2" spatial="0.1" threshold="25" />
```

This tag is used in the results_images.jspf and sets the images to the variable imageIDs. A loop through the imageIDs variable (which is a String[]) will print out the imageIDs.

In JSTL this can be done like this:

```
<c:forEach var="imageID" items="${imageIDs}" varStatus="status">
      <p>${imageID}</p>
</c:forEach>
```

Below is a data flow chart which shows how the data flows when a search is performed.

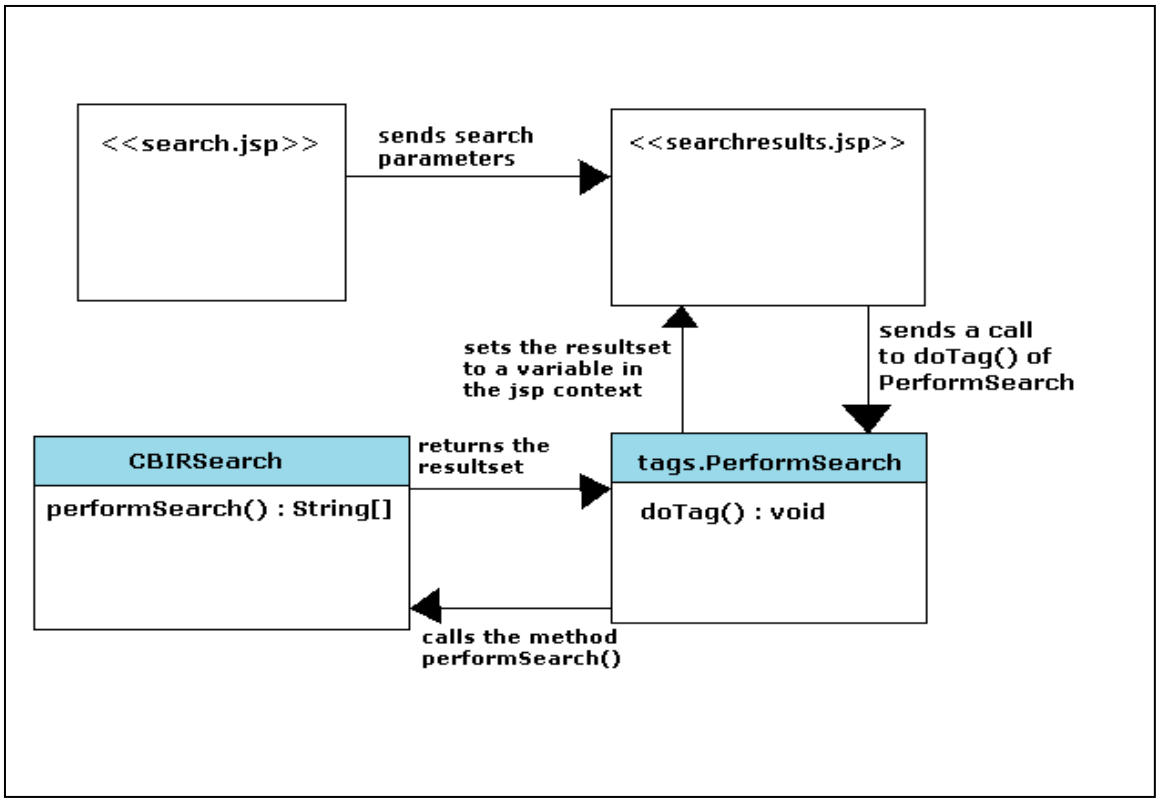


Figure 2 - CBIR data flow

2.3.2 Text-Based Image Retrieval Search

The TBIR search uses pure text as search parameter, and searches through the indexes for the large text documents (each object has one) for words that are equal. At the moment, there is no support for fuzzy, but this should be possible with Oracle full-text search and might be a task for future development. First, it was needed to index the column we were supposed to search in. This was done like this:

```
CREATE INDEX text_index ON text(TEXT)
INDEXTYPE IS ctxsys.context
PARAMETERS ('
DATASTORE CTXSYS.DEFAULT_DATASTORE
STOPLIST ctxsys.default_stoplist
filter ctxsys.inso_filter
');
```

This index should be updated every time the text table is updated. This is done using:

```
EXECUTE ctx_ddl.sync_index('text_index', '2M');
```

Searching in the indexed table is done by the CONTAINS operator. An example:

```
SELECT id from text WHERE contains(TEXT, 'church AND stave', 1) > 0;
```

The returned results are the text ids, which are linked to an object, which again are linked to one or more images. The query to find which images are relevant to each text id looks like this:

```
SELECT DISTINCT DEREF(c.image).id as imageid
FROM object x, table(x.texts) y, contains c
WHERE x.id = DEREF(c.object).id
AND DEREF(y.text).id=?
```

This query might be run several times when more than one text id is returned. Therefore, the imageIDs are added to a temporary HashSet. If the imageID already contains in the HashSet, it's not added to the result-set.

2.3.3 CBIR/TBIR Search

The CBIR/TBIR function consists of two separate searches, a CBIR search and a TBIR search. It then iterates through the result-sets to find the images that are in both result-sets. These images get a merged score (100 – CBIR Score) + TBIR Score. The reason for

subtraction of the CBIR Score from 100 is that the CBIR Score goes from 100 to 0, where 0 is the best score. The TBIR score ranges from 0-100 where 100 is the best score. In order to merge the CBIR score with the TBIR score, the CBIR score needs to be normalized. When all the images from both result-sets are given a new score and sorted, they will be placed in the first list. Then the results from TBIR (which are not in the intersection of CBIR & TBIR) are listed, sorted by the TBIR score. These are listed before the CBIR results, because text search usually has a higher precision than image search. Finally, the results from CBIR that are not in the intersection of CBIR & TBIR are listed.

In other words, the results are listed like this:

1. **CBIR ∩ TBIR**
2. **TBIR ∩ !(CBIR ∩ TBIR)**
3. **CBIR ∩ !(CBIR ∩ TBIR)**

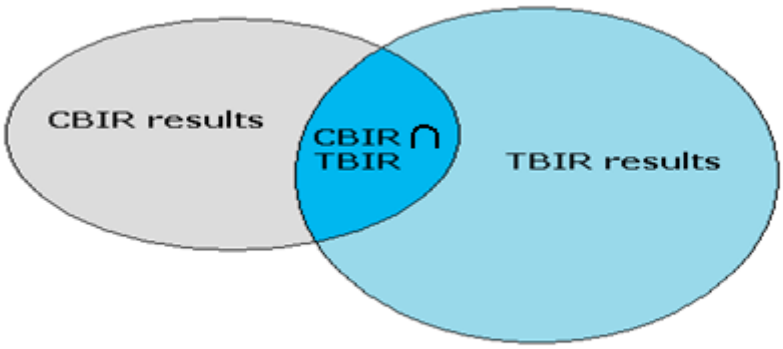
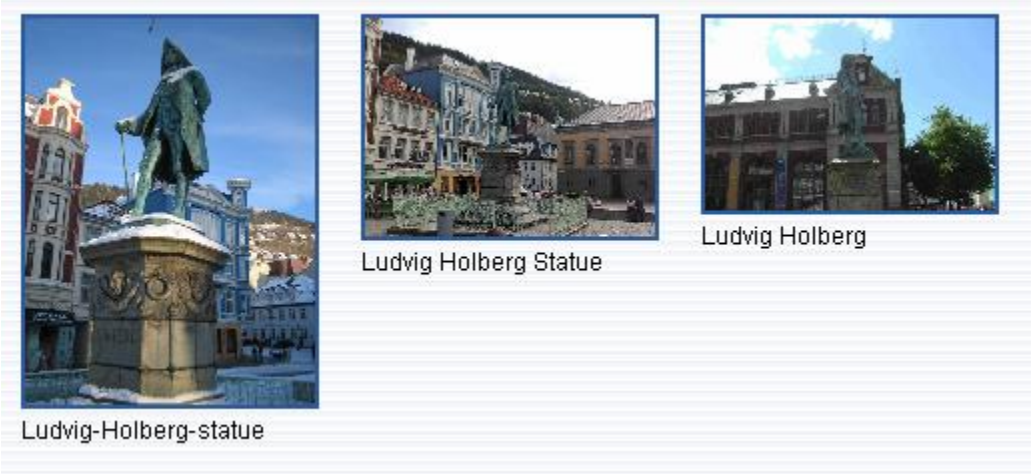


Figure 3 - CBIR+TBIR result calculation

2.3.3.1 Comparison of CBIR/TBIR from VISI2 and VISI3

There seems to be a noticeable difference in the results from CBIR/TBIR in VISI3 and VISI2. When the classic holberg.jpeg is used as the seed image, supplemented with the text «holberg», the different versions of VISI returned:



Screenshot 2 - The 3 first results from CBIR/TBIR using VISI2

Screenshot 3 - The 3 first results from CBIR/TBIR at VISI3

This is of course not enough data to conclude anything, but it seems like VISI3 returns images that look more like the seed image.

2.3.4 GPS Search

The GPS Search uses the same Oracle procedure as VISI2. The source code is located under the handlers.search.GPSSearch class.

2.3.5 CBIR & GPS Search

The CBIR & GPS Search uses the exact same procedure as MMIR3 and MMIR2, and most of the code is the same. See MMIR2 report (Rørvik, 2008) for more information.

2.4 New functionality

2.4.1 Thumbnails

VISI2's searches are very slow. One of the reasons is that it loads all the images in their full size and resizes them with CSS. VISI3 uses a thumbnail generator that creates a resized version of the original image. Images with a larger width than height set the width to 150 and the height proportional to the width. If the image has a larger height than width, the height is set to 150 and the width proportional. This made the searches much faster in VISI3 as shown in Table 1 below.

ImageID	VISI2	VISI3
66	42,1s	2,5s
39	34,0s	2,8s
29	34,6s	2,1s
Average	**36,9s**	**2,5s**

Table 1 - Comparison of VISI2 and VISI3

As you can see, VISI2 uses about 1500% more time to load a result page after a search than VISI3.

Notice! The catch is that the thumbnail generator must be run every time new images are added to the database, since the images are inserted via manual inserts and not through a Content Management System. This is done through an administration panel:

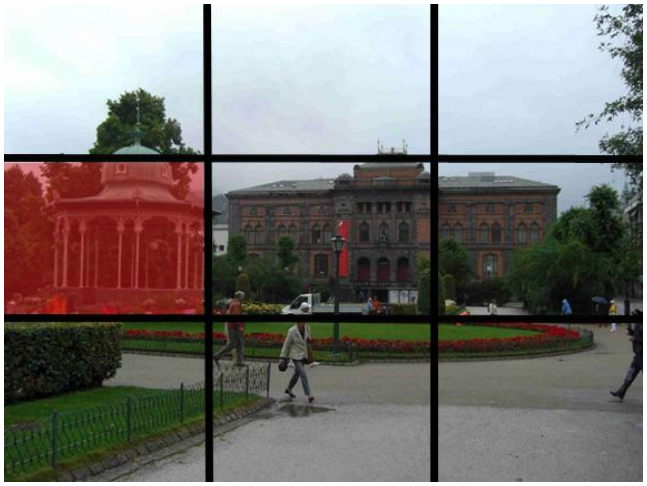
http://bulmeurt.uib.no:8080/VISI3/jsp/thumbbuilder/

2.4.2 Tile generator

An image tile generator was created to test its effect on retrieval of images of details of a building and to support the object selector function developed for MMIR3 (Hellevang, 2009). The idea was to fragment a picture into a grid of, for example 9 tiles. These tiles were to be added into the database and searched on them with the full images. The aim was to increase precision of the search results.

Screenshot 4 - Close-up image

An example: If the image in Screenshot 4, which is a close-up of *Musikkpaviljongen*, is used as a seed image to search against all the tiles in the database, hopefully a match will be made against the tile marked in red in Screenshot 5.



Screenshot 5 - The 3x3 grid with Musikkpaviljongen marked in red

The VISI3 tile generator, developed in java, can be used through the admin panel at http://bulmeurt.uib.no:8080/VISI3/jsp/thumbbuilder/. It is possible to choose the number of columns and rows the images are to be split into. This program will create tiles of all the images in the Image table and put them in bulmeurt.uib.no:8080/tempimages/. These tiles will also be inserted automatically into the database with a reference to the original image. For database details, see BergenByDB technical report (Møller, 2009).

2.4.3 Extended information about an image

Presumably, when a user selects an image from a result set, s/he wants to know more about the objects in the image. VISI2 returns structural information about the image, such as date taken, size, format, gps. VISI3 has been extended to include information about the objects in the image. If the image contains several objects, a list of them is returned to the right of the image with the name and a little description, as shown in Screenshot 6.

Screenshot 6 - Object listing

Selecting an object description will return an extended description as well as a list of all the other images in the database that contain this object.

Screenshot 7 - Other images

This is made in JavaScript, so that the entire page is not reloaded on selection of the «Neste» and «Forrige» buttons.

2.4.4 Random seed image

VISI2 used a fixed, default seed image of the statue of Ludvig Holberg. In VISI3, the initial seed image will be a random image from the database. Pushing the «Use random»-button on the main page will change the image to another random image.

3. Testing

3.1 Precision score

Precision scores are widely used in information retrieval and define the exactness of a document search. Precision is measured using the following formula:

Precision = # of relevant documents retrieved / # documents retreived

A perfect precision score of 100% means that all of the returned images were relevant.

Automatic precision testing was done using each image in the BergenBy DB as a seed image. Since object identifiers were contained in each DB image, relevance in the result sets was defined as an image containing at least one of the objects in the seed image. Precision scores were calculated for the 4 first, 8 first, 12 first, 16 first images and then for the total result-set. The reason for choosing every 4th image is that the mobile application MMIR3 receives image sets of 4, and MMIR3 is the main focus in the CAIM project. It was also assumed that a mobile phone user was very unlikely to request more than 4 result screens.

3.2 Weight testing

The default weights used in VISI2 were Texture: 0,3 – Color: 0,4 – Shape: 0,2 – Spatial structure: 0,1. These were used as a starting point to identify an improved set of search weights for the current content of the BergenBy DB. About 50 tests were run with small adjustments to the weighting for each test. Each test consisted of a CBIR search using each of the images in the database, and automatically calculating the average precision score for all 500+ searches.

Table 2 shows the 3 different weight combinations that seemed to provide the best precision scores:

Weights				Precision scores			
Texture	**Color**	**Shape**	**Spatial**	**Precision4**	**Precision8**	**Precision16**	**PrecisionTotal**
0,2	0,55	0,15	0,1	0,37	0,23	0,15	0,1
0,11	0,57	0,24	0,08	0,36	0,23	0,14	0,1
0,1	0,56	0,24	0,1	0,36	0,23	0,14	0,1

Table 2 - Precision scores using different search criteria weights

Notice! The reason why these precision scores are much higher than the ones later on is that the seed image was included in the result-sets here.

3.3 CBIR Precision

3.3.1 The algorithm for automation

The precision algorithm for the CBIR search test iterates through all images in the database that contain objects. For each image, it does a CBIR search with this image as seed image. The CBIR search returns a list of images ordered by their score. It iterates through the list of returned images and compares the object identifiers in the seed image to the object identifiers in the returned image. If one of the objects is in both images, the returned image is noted as relevant. After all the returned images are gone through, each seed image will be given a precision score at the 4^{th}, 8^{th}, 12^{th}, 16^{th} and all images in the result-set. The exception is when

a seed image search only returns 7 images. These will not be given a precision score for the 8th, 12th and 16th image. After all the images in the database have been tested, the program will calculate an average precision score for the 4th, 8th etc image. Seed images with less than 12 images will be excluded from the average calculation of Precision12 and Precision16.

3.3.2 Results

Table 3 gives the average scores for the 500 image queries in the CBIR precision test. The "Relevant hits" column shows the average number of relevant images in the result-sets. As expected, precision is higher at the 4th image than at the 8th image. Note that the average of the total precision score includes relevant images that have occurred after the fixed markers, i.e. when 7 images were returned with relevant images in slots 5, 6, and/or 7 or when 11 images were returned with relevant images in slots 9, 10 and/or 11.

Precision4	Precision8	Precision12	Precision16	PrecisionTotal	Relevant hits
13,49%	11,00%	9,54%	8,67%	8,98%	2,09

Table 3 - Precision score for CBIR Search

3.4 CBIR/GPS Precision

3.4.1 The algorithm for automation

The algorithm is similar to the pure CBIR tests. The difference is that it extracts the GPS location from the image (not the object) and also uses this as a search parameter. The test includes search using three distances; 100m, 300m and 500m.

3.4.2 Results

Table 4 shows the average precision scores for the CBIR/GPS search.

Distance	Precision4	Precision8	Precision12	Precision16	Precision Total	Relevant hits
100	77,31%	70,73%	75,60%	75,00%	54,78%	0,55
300	45,63%	38,30%	39,31%	38,13%	46,39%	2,16
500	32,83%	28,70%	24,76%	23,28%	39,72%	1,85

Table 4 - Precision score for CBIR&GPS search

As can be expected, precision decreases as the distance increases. However, the total of relevant images returned increase from 0,55 per search to 2,16 per search while increasing distance from 100 to 300. This is because a lot of the images are taken from a distance larger than 100 meters. Surprisingly, the amount of relevant images returns decreases when the distance is increased to 500. This might be due to a bug in the Oracle procedure, but it was

discovered too late to look more into.

4. Tools used

- Acer Aspire 5530 w/ Windows Vista
- Eclipse IDE Ganymede v 3.4.1 with Tomcat plugin installed
- Oracle SQL Developer

Acer Aspire 5530 was the computer that I used this summer, and Windows Vista was the operating system that I ran.

I used Eclipse for J2EE programming and the Tomcat plugin was necessary so that I didn't have to deploy the project manually for every change I did.

Oracle SQL developer was used for getting information about the database, testing queries, changing procedures etc.

5. Future development

For future development, I think it would be interesting to test the tile search properly, where one searches in both the tiles and the regular image table. If this gives a positive result, this can be implemented in the regular CBIR search.

Another thing which could be done to improve the TBIR search is to extend the index from only the text column to include the name column. Now a search for «Mariakirken», will not return relevant images because the church is called «St. Mary's Church» in the text.

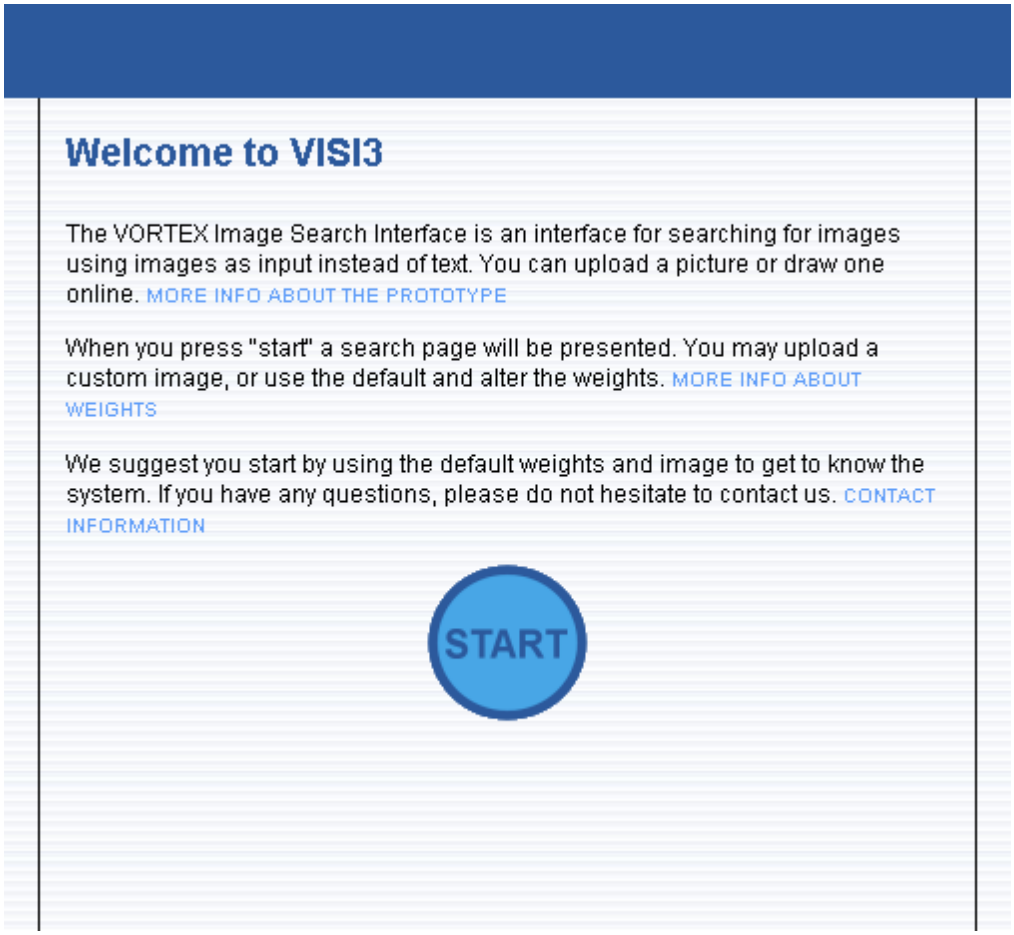
Additionally, one could add support for user-generated tagging on the pictures, like the functionality in Facebook. The tagging would be of objects, and not of people, though.

The bug that was found during CBIR & GPS precision testing should also be looked into. It's not logical to get more relevant images with a distance of 300, than with a distance of 500, since the expected result would include the images that at 300 plus the images with 300+ distance.

Last, but not least, I think the code base needs some refactoring. It could be useful to use a framework like Spring. Since this was my first time developing an entire web project in Java, the architecture probably isn't optimal.

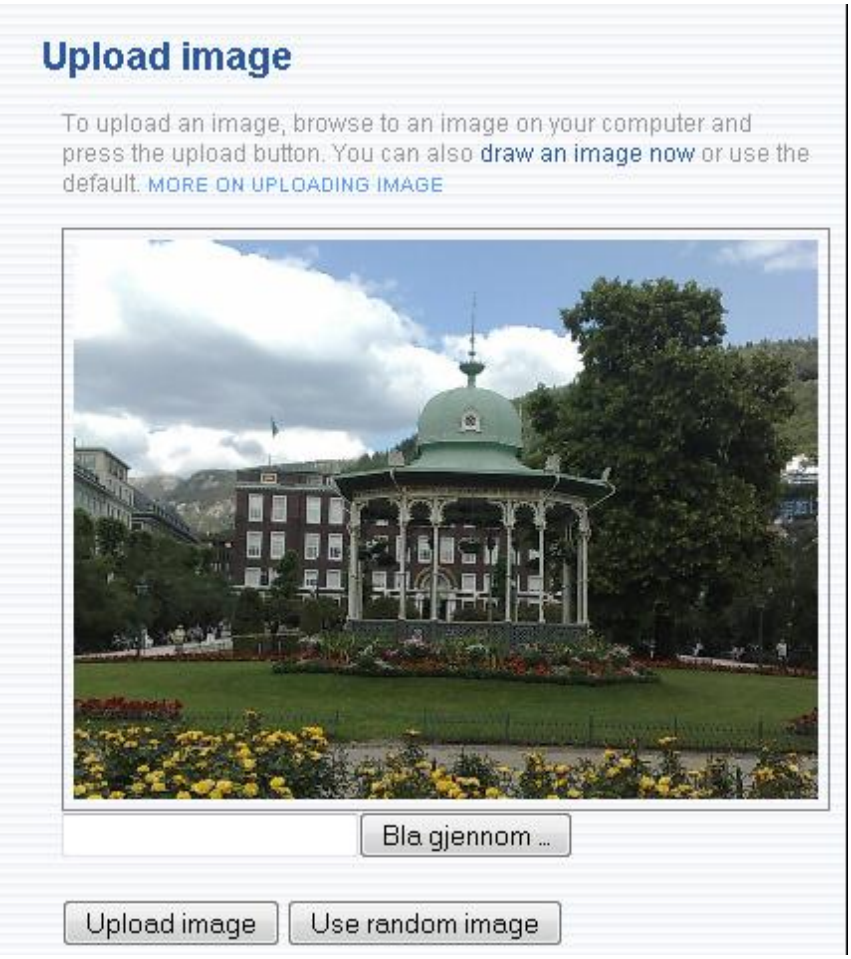
6. User manual

The start screen for first time entry into VISI3 at http://bulmeurt.uib.no:8080/VISI3/ is shown below. The Start-button provides entry to the search page.



Screenshot 8 - The front page

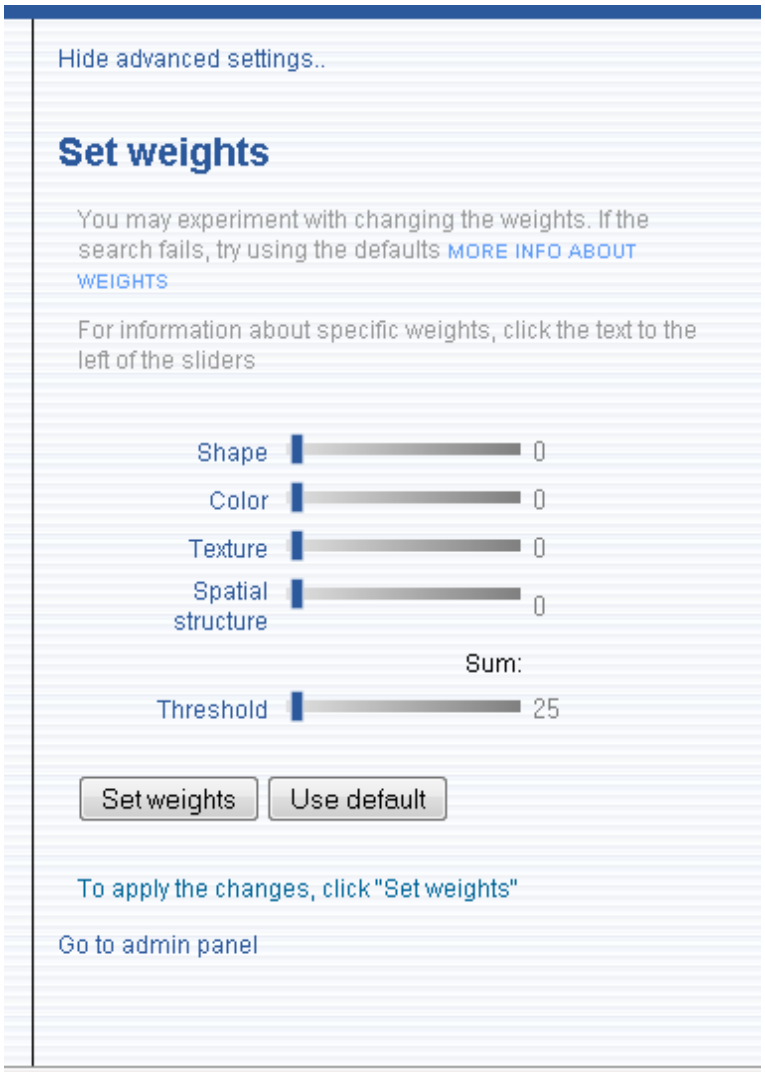
The search page has 3 panels for specification of an image query, shown in screenshots 9, 10 and 11 below.



Screenshot 9: The left box on search page

This is the left part of the search page. This part of the site contains all information about the seed image. The user can choose to upload from his/her own computer by pushing «Bla gjennom ...», find the image and push «Upload image». Or he/she can push «draw an image now» (in the introductory text) and draw his own image with the Java Applet available. Or else he/she can push «Use random image» and another random image from the database will appear.

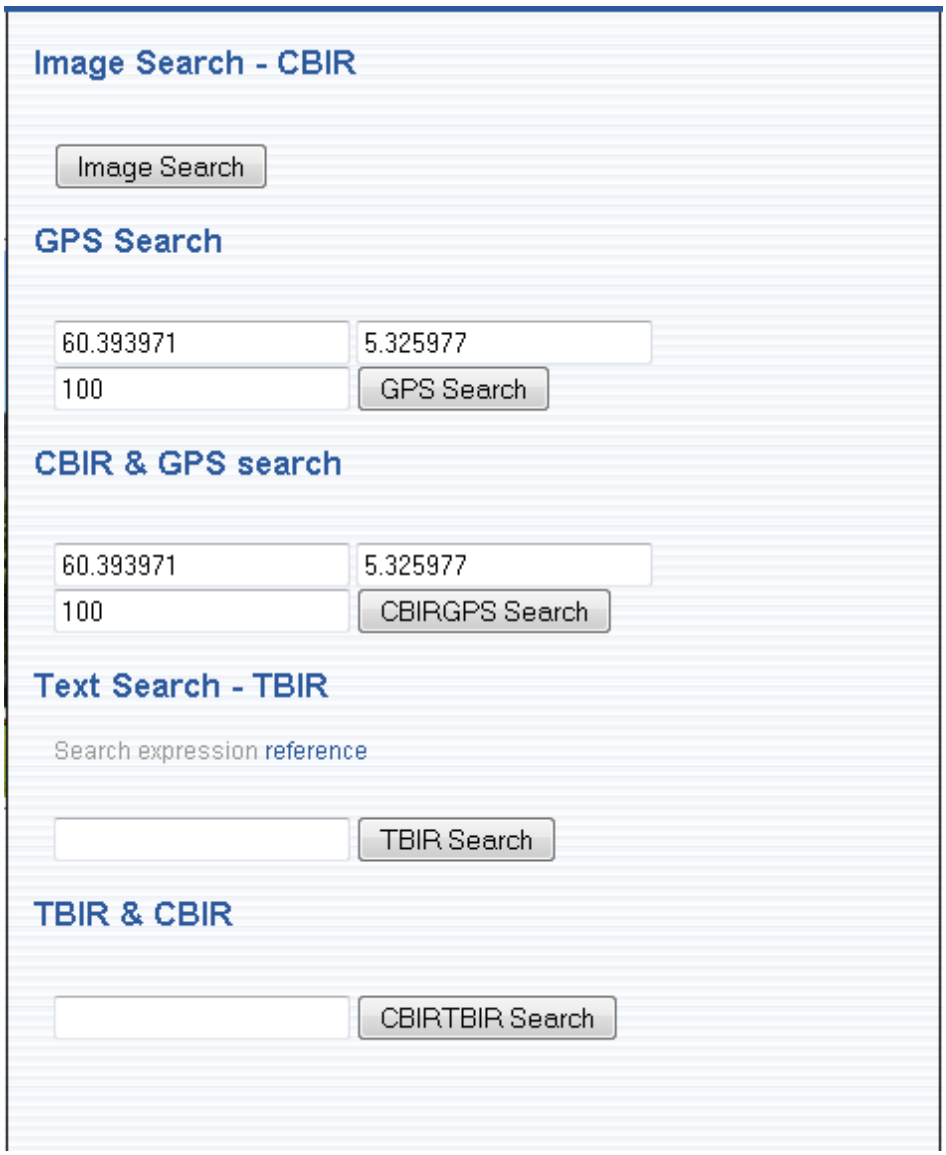
Weights are specified using the following panel:



Screenshot 10: The right box on search page

This is the advanced settings. To see this, push the link «Show advanced settings» in the top right corner of the search page. Here the weights for the search can be changed. To find out more about the specific weights, push the links «Shape», «Color» etc. The «Go to admin panel» is a password-protected admin-panel, which will be gone through later.

The various search options are shown below and in the central panel of the VISI3 search page.



Screenshot 11: The center box on search page

Image search uses the seed image specified to the left, and does a normal CBIR search.

GPS Search uses the specified GPS positions for the search and does not use the seed image. A distance measure can also be given in the 3rd text-box. The distance is measured in meters that the object can be away from the image (photographer) gps location and still be included in the result-set. The default is 100 meters.

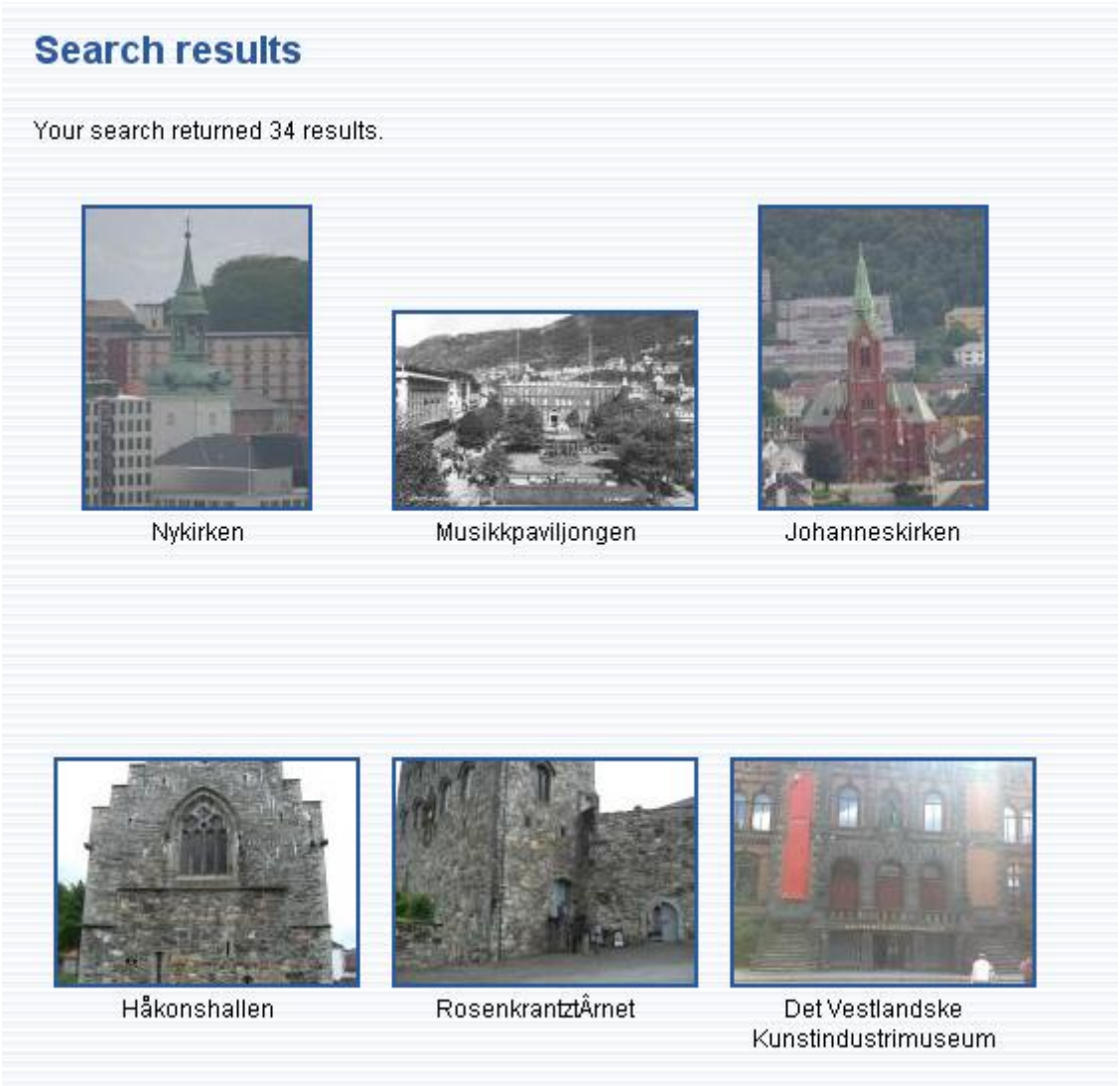
CBIR&GPS Search does a combined CBIR and GPS search and uses both the seed image

and the GPS specified coordinates.

TBIR Search does a pure text search with the parameters specified in the text box. Boolean operators NOT, AND, LIKE and NEAR can be included.

TBIR&CBIR Search combines the parameters from the text box and the seed image and does a TBIR&CBIR search as described above.

A typical result page is shown below in Screenshot 13. It is possible to select any image which will return more information about that image and the objects in the image.



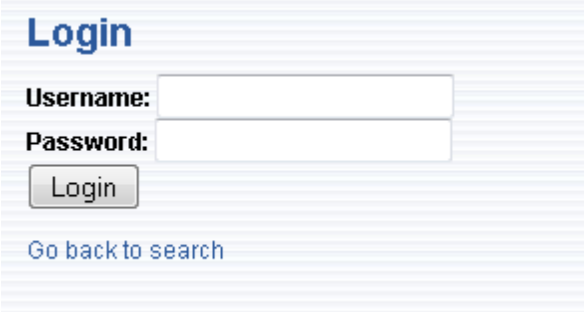
Screenshot 12 - Search results

This is the layout of the page you'll see when you have performed a search. You can click each of these images to look at a larger version and find out more about the objects that are visible in the image.

Screenshot 13: Image-specific page

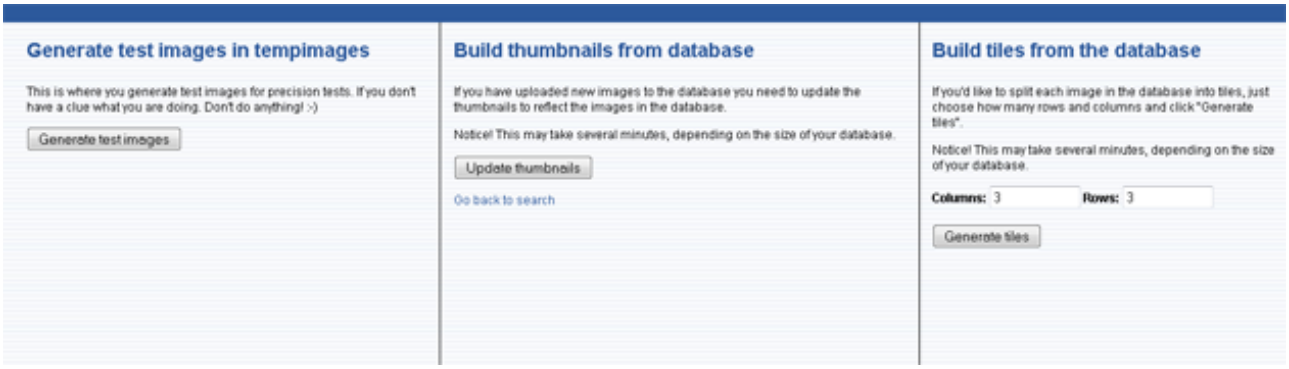
At the left, there is some image-specific information, like width, height, gps location of the picture etc. There is also a magnifier function. The square box on the image can be dragged around and it will magnify the content under it. To the right, there is information about the object in the image. Below the picture is a list of other images that contain this object. If the image contains more than one object, there will be a list of the objects in the image, as you see in screenshot 1 on page 13.

Admin panel



Screenshot 14: Login box

First, you need to login with the username and password. To get the username and password, contact the CAIM administrator.



Screenshot 15: Admin panel

The left panel allows generation of test images for the precision testing. These will be placed in the http://bulmeurt.uib.no:8080/tempimages with names like testimage_imageID.jpg

In the center box, is used to update the thumbnails. This must be done every time images are added to the database. The thumbnails will be placed in http://bulmeurt.uib.no:8080/tempimages with filenames like thumb_imageID.jpg.

The right panel is to be used to generate tiles from all the images in the database. Remember to delete the tile sequence in the oracle database before doing this, unless this is the first time creating tiles. The tiles will be placed in http://bulmeurt.uib.no:8080/tempimages with filenames like tile_imageID_fragmentNumber.jpg. The fragmentNumber is a number from 1-9 if a 3*3 grid is defined and 3 rows, where 1 is the tile in the top left corner, and 2 is the tile in the top center, etc.

7. References

Hartvedt, C. (March 2007) Utilizing context in ranking results from distributed image retrieval – the CAIRANK Prototyp, Dept. of Information and Media Sciences, Univ. of Bergen.

Hellevang, M. (Sept. 2008) MMIR2 - Mobile Multimedia Image Retrieval (ver.2). CAIM-TR-4, Dept. of Information and Media Sciences, Univ. of Bergen.

Hellevang, M. (Sept. 2009). MMIR3 – Mobile Multimedia Information Retrieval. CAIM-TR-7, Dept. of Information and Media Sciences, Univ. of Bergen.

Møller, T. (Sept. 2009). Bergen By – a Multi-Modal Image Database. CAIM-TR-9, Dept. of Information and Media Sciences, Univ. of Bergen.

Næss, B. (Sept. 2007). The VISI Prototype for Image Retrieval. CAIM-TR-1, Dept. of Information and Media Sciences, Univ. of Bergen.

Rørvik, A. (Sept. 2008) VISI2 - Combining CBIR Search with Text and GPS Location Criteria. CAIM-TR-6, Dept. of Information and Media Sciences, Univ. of Bergen.