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

MMIR3

Mobile Multimedia Information Retrieval

The third edition of the CAIM prototype for mobile image retrieval using the Nokia S60 Platform, Java Mobile Edition and Oracle InterMedia

Mathias Hellevang

Sept. 2009

CAIM-UiB

TR-7

Table of Contents

1 Introduction

This report is written to document the MMIR3 (Mobile Multimedia Information Retrieval 3) prototype developed for the CAIM-project (Context-Aware Image Retrieval) during the summer of 2009 by Mathias Hellevang. MMIR3 is the third mobile prototype developed for the CAIM project, and is based on the MMIR2 prototype developed by Hellevang the summer of 2008.

1.1 Objectives

The project objectives for the MMI3-prototype were to expand the MMIR2-prototype to include support for CBIR (Content Based Image Retrieval) by drawing / sketch, i.e. using a drawing to search for images rather than using images from the mobile camera. The drawing functionality would be designed to be compatible with the touch screen available on the Nokia 5800 XpressMusic mobile phone.

In addition to this the prototype was to be extended to be able to download relevant information about a selected building / image, e.g. audio or text with the building's history.

Last but not least, the prototype was to utilize the Nokia 5800 XpressMusic touch screen to perform object specification on a given image, i.e. select relevant parts from an image (e.g. selecting the building, but not the bushes around). CBIR by object specification was to be tested in real life situations using a precision score to determine performance.

A detailed list of objectives follows:

	Requirement	Status summary
1	Query-by-sketch functionality to enable queries by free hand sketches	Implemented to be compatible with the Nokia 5800 XpressMusic mobile phone and its touch screen (with stylus). Includes a selection of eight colours and a single pencil-shape (1px thick).
2	Object-specification for images on the MMIR3 prototype	Implemented to be compatible with the Nokia 5800 XpressMusic mobile phone and its touch screen (with stylus). It is possible to select a rectangle from photo taken by the camera, making it possible for the user to select the relevant parts of an image, while discarding the irrelevant parts.
3	Develop feature to retrieve text about a given image / object	All searches will now return relevant text for an image automatically. The feature can be disabled / enabled in MMIR3's settings panel.
4	Develop feature to retrieve audio about a given image / object	All searches will now return relevant audio for an image automatically. The feature can be disabled / enabled in MMIR3's settings panel.
5	Correct any bugs present in MMIR2	A bug in the browse-feature which caused the application to crash is fixed. Another bug was present in the display of total images, but should now be corrected. Some cosmetic work done to improve the application's consistency.

2 Tools used

Eclipse (version 3.5.0): Eclipse is a graphical integrated development environment (IDE) for Java. Eclipse supports a wide array of plug-ins, making it expandable and very customizable. Eclipse was used for MMIR3 because it has been used with great success for earlier versions of MMIR.

Mobile Tools for Java (version 1.0): Mobile Tools for Java, previously known as JavaME, is a plug in for Eclipse which includes support for J2ME for S60-mobile phones.

Nokia N97 SDK for Symbian OS (version 1.0): Nokia's software development kit. This includes required J2ME-libraries not included with a default Java-setup, an emulator for testing J2ME-application, as well as different utilities to make communication with mobile phone easier.

Oracle SQL Developer (version 1.5.4): SQL Developer is a free, graphical tool for database development. This is a very handy tool for development, maintenance and testing of object-relational databases and procedures.

3 Design

3.1 Overview of design

MMIR3 is the third mobile prototype developed at the University of Bergen for the CAIM-project. MMIR3 is based on MMIR2 (Hellevang, 2008), and uses the same architecture as the previous version. The overall design of the prototype is composed of three parts:

1. The MMIR3 client, running on a Nokia 5800 XpressMusic mobile phone.
2. The MMIR3 server application. The server is running as a servlet on bulmeurt.uib.no, inside an Apache Tomcat container.
3. The BergenBy database server, also located at bulmeurt.uib.no (developed using Oracle 10g).

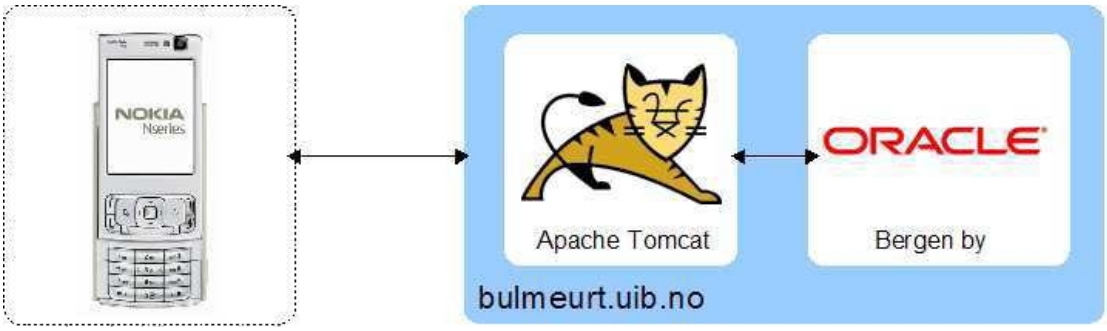


Figure 1: The MMIR3 prototype. Picture originally developed by Roe Fyllingsnes and Christian Hartvedt for the MMIR prototype.

3.2 The MMIR client

MMIR3, like MMIR2, tries to follow the MVC-framework. However, due to constraints caused by the J2ME graphics API, MVC is only partially followed. Never the less, most of the GUI-elements are divided into a view and a controller class, with most of the logic placed in the controller.

The client has these main features, represented in the menu under the following labels:

- **Search**: This is the primary feature of MMIR. It enables the user to perform a CBIR search against the BergenBy DB, using the mobile phone's integrated camera to provide a seed image. In MMIR3 this feature is expanded by providing the user the option of selecting a part of an image which the user considers more relevant for a CBIR search. The Search-feature returns a ranked result set with images that match the image used in the query.
- **Draw and search**: Draw and search presents the user with a white screen, 8 colours and a simple pencil-form. This makes it possible to draw simple sketches of objects using the Nokia 5800 XpressMusic mobile phone and a stylus. The sketch can be sent to the server as a CBIR query, just like with the normal Search-function. Draw and search returns a ranked result set with images that match the image used in the query.
- **Store image**: Storage of photo taken by the phone's integrated camera. Performs like Search (without object selection), but the user will not be presented with a result set.
- **Browse for image**: This works like Search (without object selection), but presents the user with a menu enabling him or her to use a stored image from the phone's disk rather than taking a photo with the camera. Browse for image returns a ranked result set with with images that match the image used in the query.

In addition to the previously mentioned features the client has two more options in the menu,"Settings" and "About". Settings provides the user with the option to disable / enable GPS, disable / enable downloading of multimedia (i.e. text and audio), and options related to the GPS range. The "About" feature contains information about the MMIR3 prototype as well as a list of developers.

3.3 The MMIR3 server

The MMIR3 server is a Java servlet running on bulmeurt.uib.no. The server enables the MMIR3 application running on the phone to perform actions it otherwise wouldn't be able to perform both due to the limited functionality exposed through J2ME, and because of the limited processing and storage capacity of the mobile phone's MMI3 it is intended to run on.
The MMIR3 server uses the data delivered from the client to perform queries and inserts on the database. In addition to the database that performs GPS and CBIR queries, most of the MMIR3 logic lies in the MMIR3 server.

Basically the server receives a command bundled with necessary data, and performs some action on the BergenBy DB. The command states what should be done, e.g. performing a regular CBIR search, storing an image to the database or performing a search using the image of a drawing.

In a default scenario where the user is using the «Search»-feature, the server receives an image, a time-stamp from when the photo was taken, a GPS-coordinate (if applicable[1]), and a range. The server will then perform a CBIR search on the Bergen By DB using Oracle's proprietary image search functionality, pick all the returned images, bundle them with text and audio related to the object shown in the image (if applicable[2]). The client will then be told that the search is completed, and will send a new command to the server requesting to download the first four images in the result set.

4 Prototype documentation

The overall structure of the MMIR prototype is thoroughly documented in the report for MMIR2, and will therefore not be described further in this report (Hellevang, 2008). Noteworthy changes are the ones described in the list of requirements, as well as a change in the weighs used for CBIR in the BergenBy DB.

1 There are a few situations where GPS coordinates are not sent. GPS can be enabled / disabled in the Settings menu, and if disabled they will not be sent to the server. Also, if no GPS coordinates are found by the time the image is taken, they will not be sent.
2 This feature may be enabled / disabled in the MMIR3 settings menu.

Changes made to the BergenBy database are described by Thor Møller, and will therefore not be elaborated on in this report (Møller, 2009).

4.1 Changes in weights

The weights are changed from MMIR2 to MMIR3. The changes are based on testing performed by Christoph Carlson to find the best weighs for CBIR search (Carlson, 2009).

The new weights are:

- **Shape**: 0.2
- **Color**[3]: 0.55
- **Texture**: 0.15
- **Spatial**: 0.1

The **threshold** has been changed to 50, as this seems to be a better threshold for search by object specification than the previous 25.

4.2 Query-by-sketch

Query-by-sketch is a simple drawing tool within MMIR3. It's function is to enable the user to perform CBIR queries for objects which he or she currently isn't in front of, and therefore cannot photograph. The intention is for the user to draw a quick sketch of the object he or she wants to find, and send this image to the MMIR3-server like a normal photograph. See Figure 2 and 3 for example from MMIR3.

3 Spelled “color” and not with the proper spelling of “colour” since Oracle (and the variable in MMIR3) is spelled with the American spelling.

Figure 2: Mobile sketch of Edvard Grieg statue

Figure 3: Edvard Grieg statue

Query-by-sketch is implemented in MMIR3 as *DrawImageView* and *DrawImageViewController*. DrawImageView is a Java-class extending the J2ME abstract class **Canvas**. Canvas is a class which enables the application to access low-level events and graphics calls for drawing on the mobile phone's display.

DrawImageView is designed to be used with a touch enabled mobile phone like the Nokia 5800 XpressMusic. It extends the Java class Canvas, and implements the Canvas-native functions **pointerPressed()**, **pointerReleased()** and **pointerDragged()**, in addition to the required Canvas-method **paint(Graphics g)**. The pointer-methods are used to listen to events performed on the screen, i.e. if a stylus touches the screen, the pointerPressed() and pointerReleased() will be called when the stylus touches and stops touching the screen, respectively. The pointerDragged()-method is called when the stylus is dragged across the screen. Combining these methods, DrawImageView records the actions a user performs on the touch screen, and using the native paint(Graphics g)-method the result is drawn on the screen, creating a simple sketching-application.

4.2.1 DrawImageView's double buffering

J2ME doesn't support the extraction of images from the Canvas. That is, if a user draws something on the Canvas, it's not possible, using J2ME, to take that drawing and save it as an image. Therefore DrawImageView uses a technique known as **double buffering**, which basically means that instead of drawing directly to the Canvas, the application draws to an image, which is painted to the screen

DrawImageView's paint method using double buffering:

```
protected void paint(Graphics g) {
    if (!init) {
      // init metod removed due to space constraints
    }
    buffer.setColor(colour.getRed(), colour.getGreen(), colour.getBlue());
    if (pressed) {
        buffer.drawLine(xcoord, ycoord, xcoord+1, ycoord);
        buffer.drawLine(xcoord, ycoord+1, xcoord+1, ycoord+1);
    }
    if (dragged) {
        buffer.drawLine(xcoord_old, ycoord_old, xcoord, ycoord);
    }
    g.drawImage(bufferImage, 0, 0, Graphics.LEFT | Graphics.TOP);
    pressed = false;
    dragged = false;
}
```

Graphics is here defined as DrawImageView's graphic object. The paint method is called at regular intervals, and if a change has occurred (e.g. somebody activating the pressed-variable by pressing a stylus to the screen), a drawing-action is performed on the **buffer** using the updated x and y coordinates. Any change happening in the buffer is reflected in the image **bufferImage**, which in turn is painted on the Canvas' graphical object, g, by the use of g.drawImage(). Thus, any changes performed on the screen are painted to the image, which is painted on the screen.

4.2.2 DrawImageView bitmap conversion

If a user is content with the sketch he or she has drawn, he can select the option “Search” from DrawImageView's menu. This will activate DrawImageViewController, which in turn will send the image to a method, getByteArray(Image image). getByteArray() converts the image from a Java Image-object to a byte array containing all the different colours of all the pixels in the image, and DrawImageViewController sends this byte array to the MMIR3-server along with necessary metadata and the proper command.

The method that creates the byte array from J2ME Image:

```
public byte[] getByteArray(Image image) {
	int raw[] = new int[image.getWidth() * image.getHeight()];
	image.getRGB(raw, 0, image.getWidth(), 0, 0, image.getWidth(),
	image.getHeight());
	byte rawByte[] = new byte[image.getWidth() * image.getHeight() * 3];
	int n = 0;
	for(int i = 0; i < raw.length; i++) {
		int RGB = raw[i];
		int r = (RGB & 0xff0000) >> 16;
		int g = (RGB & 0xff00) >> 8;
		int b = RGB & 0xff;
		rawByte[n] = (byte)b;
		rawByte[n + 1] = (byte)g;
		rawByte[n + 2] = (byte)r;
		n += 3;
	}
	raw = null;
	return rawByte;
}
```

In the MMIR3-server, there is a similar function like the one presented above. This converts the image from a byte stream into a Java BufferedImage object, which is necessary to store it to disk and use it as a search image. See the method toImage(byte[] data, int w, int h), located in the CaimDAO-class in the MMIR3-server source code for implementation details.

J2ME does not include a JPEG encoder. Images taken by the Camera can be saved as JPEG, but images created outside the camera, like the one created by the query-by-sketch tool isn't able to access the same functions as the camera. Therefore the image is sent to the MMIR3 server in an uncompressed state as a bitmap. This means that the amount of data sent to the server using the query-by-sketch tool is greater than the data for images taken by the camera (the ones from the "Search"-tool), and needs to be converted to JPEG in MMIR3-server. The MMIR3 server includes a JPEG-encoder which encodes the image to JPEG, making it possible to perform queries on the BergenBy DB.

4.3 Object selection

Object selection is a new feature added to MMIR3. It extends the old camera search by adding the possibility to select parts of an image, with the result that only the selected part is sent as a query, while the parts not selected are discarded. The intended use for this function is to improve the seed

image by removing what the user considers to be unnecessary noise from the image.

Figure 4: Ole Bull statue

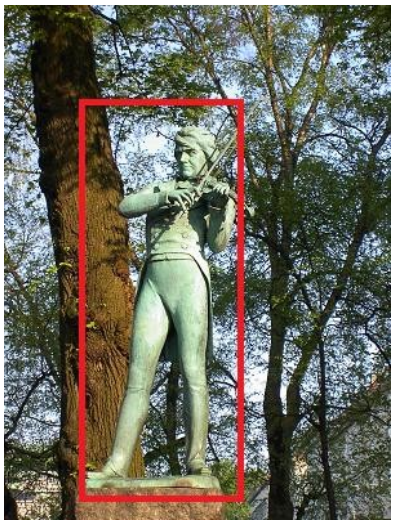


Figure 5: Ole Bull statue with object selection

Take for instance these images of the Ole Bull statue (Figure 4, Figure 5). The image on the left displays the statue and the surrounding vegetation, while the image on the right displays the same image with the statue marked / selected. With this specific selection only the Ole Bull statue will be sent to the server, while the rest of the image is discarded.

4.3.1 Client side

Object selection is implemented in the MMIR3 client by using drawing-functions similar to the functionality described in the query-by-sketch feature. It is divided into two Java classes, CaptureImageView and CaptureImageViewController. CaptureImageView is responsible for capturing images with camera, and for allowing the user to select relevant parts of the image. CaptureImageViewController is responsible for cropping the selected parts of an image and sending this to the NetWorker class, which communicates with the MMIR3-server.

The main difference between the painting-functionality in DrawImageView and CaptureImageView is that while the query-by-sketch feature draws to a buffer which it paints to the Canvas, the object selection feature paints directly to the Canvas, and only cuts the image when the image is ready to be used.

While it's still true that you cannot convert a Canvas to an image in J2ME, and something like a double buffer is required to extract an image from Canvas – CaptureImageView doesn't need the Canvas, it has the image captured by the camera.

When CaptureImageView captures an image with the camera the image is stored in an image variable, **fullSizeImage**, which is then painted to the Camera. This means that instead of using double buffering like DrawImageView, it can just manipulate the original image instead of extracting data from the Canvas. Thus, any selection actions performed by the user on the Canvas (i.e. drawing a rectangle around an object) is reflected on the Canvas by painting a rectangle, while the original image is preserved as-is. Concurrently as the painting happens, the x and y coordinates relevant for the selection are stored in proper variables in CaptureImageView. The original image, fullSizeImage, isn't touched before the user indicates that he or she is happy with the selection, by pressing the "Search"-button.

The cropped image is, like the image from the sketch-by-query feature, sent to the server as an uncompressed bitmap, and is there converted to JPEG and used in a query.

4.3.2 Server side

When the client sends a cropped image to the server via the NetWorker class, it includes a command and various metadata in the message. The required metadata for a selection search is the command which tells the server what to do, as well as the height and the width of the image. Width and height are required because the cropped image, like the sketch, is sent in raw format and needs to be encoded as JPEG on the server side.

On the server, in the method doPost() located in the CaimServlet class, the commands and variables are first parsed and saved in variables.

```
String latitude         = req.getParameter("latitude");
String longitude        = req.getParameter("longitude");
String timestamp        = req.getParameter("timestamp");
String command          = req.getParameter("command");
width                   = req.getParameter("width");
height                  = req.getParameter("height");
if (req.getParameter("distance") != null) {
      distance = Integer.parseInt(req.getParameter("distance"));
} else {
      distance = 100;
}
```

The image is not parsed as it is not part of the message, but is rather sent when the server is notified that it should receive an image. After this, the command is checked to decide which action to take.

Here is the check for tilesearch, which is the command for performing selection searches:

```
else if (command.equalsIgnoreCase("tilesearch")){
      System.out.println("CAIMServlet: Tilesearch");
      dao.saveRawImage(img.getImageByteData(), Integer.parseInt(width),
                  Integer.parseInt(height));
      dao.processImageTileSearch();
      images = dao.getImageSet();
      dao.clearTable("tile_search_result_tab");

      // save image to temp table for temp storage
      dao.importTempImage(0, 0, 0);
}
```

As shown, the block is responsible for calling a number of methods necessary for performing the search. saveRawImage() is the method responsible for converting the byte[] bitmap to an JPEG image, and saving this to disk. When this is done, the search is performed on the Tile collection in the database[4] by calling processImageTileSearch(). ProcessImageTileSearch() is the method that tells the database that it should perform a query, and does this by calling the SERVER_SEARCH procedure, located in the TILE_SEARCH package.

When processImageTileSearch() has completed, the search has been performed in the database, and the images can be downloaded to the server. This is done by calling getImageSet(), and saving them to an ArrayList called **images**. getImageSet() is described further in chapter

4 Note that the selection search feature only performs search in the Tile table, not the Image table.

4.4 Text and audio retrieval

The purpose of searches is to *find* something you are looking for. In text retrieval you are looking for a text which contains one or more of the words you specify, or somehow conforms to the standards you have set for the search (e.g. dates, file size, authors) . This is also the case for CBIR. When you used MMIR and MMIR2 to search for an image, you would get a set of identical or similar images back from the server. This is all very good, CBIR works as intended, but for a tourist in the tourist scenario this isn't very helpful since the tourist is looking at the object of interest. In MMIR3 the prototype has been extended to also retrieve relevant text for an image, as well as relevant audio. The text will typically be a history about the building, the statue or the person depicted in the statue etc. The audio is currently a read aloud version of the same information you find in the text.

4.4.1 Implementation in the client

Text and audio retrieval is implemented in the MMIR3 mobile application as an optional feature which will download the relevant text and audio for an image / object at the same time the image set is downloaded. This means that if audio and text retrieval are enabled in the settings menu, both audio and text will be downloaded for each image returned from the query[5].

In the MMIR3 Java class **NetWorker**, located in the package **no.uib.caim.mmir3.util**, there is a method called executeGetImageSet(int number), which fetches the image set specified in the argument, i.e. if the method is called with e.g. 0 as an argument, it will fetch the four first images (since an image set is composed of four images in the MMIR2 and 3 prototypes). In addition to fetching the images, this method is also responsible for calling the appropriate methods for fetching of audio and text (exectuteGetObjectText() and executeGetObjectAudio() respectively), as well as bundling the audio and text with the images. When the user has performed a search, the executeGetImageSet()-method will be called automatically, and both images and audio / text will be downloaded (if enabled). This method is also called when the user indicates that he or she wants to see the next or previous four images in the result set.

5 If the user is using a 3G connection rather than wireless network it might be practical / suitable to disable this feature, as downloading the audio files is time consuming and costly using 3G.

The details of fetching the audio and text objects are implemented in a simple, albeit long, piece of code which works similarly to the fetching of images. See the MMIR3 client source code for more information.

After the text and audio have been bundled with the image, they will be presented to the user with the help of the regular FullScreenView and FullScreenViewController introduced in MMIR2. The user will have the option of playing the audio or reading the text.

Audio playback is provided with J2ME, using the Player-class. If the play-button is pressed the player will be created with an InputStream as an argument, reading the audio file from the mobile phone's memory card[6]. The following piece of code is located in the FullScreenViewController:

```
if (!playing){
	String path = MMIRDAO.getPath() + "/" + audioIndex + ".mp3";
	FileConnection fc;
	try {
		fc = (FileConnection)Connector.open(path, Connector.READ);
		InputStream is = fc.openInputStream();
		player = Manager.createPlayer(is, "audio/mp3");
		player.realize();
		player.start();
		playing = true;
	}
	catch (IOException e) {
		e.printStackTrace();
	}
	catch (MediaException e) {
		e.printStackTrace();
	}
}
else {
	try {
		player.stop();
		playing = false;;
	} catch (MediaException e) {
		e.printStackTrace();
	}
}
```

If audio isn't playing, a new player will be created. Otherwise, the player will be stopped. There is no pause feature, only start and stop.

6 The audio file is temporarily stored on the memory card to reduce the memory use (the application will crash otherwise), and to provide the player with a file to read from.

Displaying of text is provided by using the J2ME StringItem()-class.

```
else if (command == showTextCommand){
      removeCommands();
      this.stringItem = new StringItem(title, text);
      this.view.addCommand(showImageCommand);
      this.view.addCommand(playAudioCommand);
      this.view.deleteAll();
      this.view.append(stringItem);
}
```

This is a very simple piece of code, but basically a new StringItem-object is created, the proper commands for displaying of text are added to the view and the StringItem is added to the screen, thereby showing the text.

4.4.2 Implementation in the server

On the server side, fetching is implemented as two methods that perform queries to the BergenBy DB server, and two methods that send the data to the client upon request. The two methods performing queries to the database are named **getObjectText()** and **getObjectAudio()**, and both take the object and image id as arguments.

4.4.2.1 Downloading to server from BergenBy DB

When a CBIR query is performed on the server, the images found are downloaded to the server from the database using a method called getImageSet(), located in the Java class CaimDAO. This method first calls the getImage() or getTile() method, depending on whether the search is performed in the Tile or the Image table in the database. This will download all the images to the server. After this, getImageSet() proceeds to call the methods for retrieving audio and text for every image (getObjectAudio() and getObjectText()). That is, for every image returned by getImage() or getTile(), the method retrieving audio or text for that image is called once (e.g. a result set of 20 images will call the audio method 20 times, and similar the text method 20 times).

getObjectText() and getObjectAudio() both actively cache data on the server. Both the audio file and the text file related to a given object are stored in memory on the server if a search has been performed. Thus both the methods will check to see if the data is already downloaded – otherwise they proceed to execute a query against the server.

Both the getObjectText() and getObjectAudio() methods are rather long, so they will not be published in this document. See the full source code for implementation details. However, they both perform very basic queries, and then proceed to download the result from the database to the server, and these queries follow bellow:

Query for downloading text for a given image:

```
SELECT deref(y.text).id as Id, deref(y.text).summary as Summary,
deref(y.text).language as Language, deref(y.text).text as Text
FROM object x, table(x.texts) y, contains c
WHERE x.id = DEREF(c.object).id and deref(c.image).id= + imageId;
```

Query for downloading audio for a given image:

```
SELECT deref(y.audio).id as id, deref(y.audio).language as language,
deref(y.audio).audio as audio, deref(y.audio).narrator as narrator
FROM object x, table(x.audio) y, contains c
WHERE x.id = DEREF(c.object).id and deref(c.image).id= + imageId;
```

In both cases, imageId is the id of the image the method will fetch audio or text for.

4.4.2.2 Downloading to client from the MMIR3 server.

When the mobile phone requests to download an image set, it also requests to download the audio and text related to any image in the image set (if enabled in settings). This is done by executing this code snippet, located in **executeGetImageSet()** in the NetWorker class.

```
Vector imageSet = new Vector();
Text text;
Audio audio;

SetImage image;
for (int i = 0; i < 4; i++) {
      image = new SetImage(executeGetImage(currentImageSetNumber, i),
            0, "");
      image.setImageId(executeGetImageId(currentImageSetNumber, i));
      image.setImageTitle(executeGetImageTitle(currentImageSetNumber, i));

      if (fetchMultiMedia) {
            text = executeGetObjectText(currentImageSetNumber, i);
            image.setImageText(text);

            audio = executeGetObjectAudio(currentImageSetNumber, i);
            byte[] test = audio.getAudio();
            MMIRDAO.saveTempAudio(test, i);
            image.setAudioIndex(i);

            audio = null;
            test = null;
      }
      imageSet.addElement(image);
}
```

imageSet is here a list containing up to four **SetImage** objects. SetImage is a Java class created to bundle binary images with text, audio and relevant metadata. Currently SetImage has the following properties:

- byte[] imageData – the image stored in a byte array.
- int imageId – the id the image has in the database
- String imageTitle – the title of the image, retrieved from image caption in the database
- Text imageText – a Text-object containing the CLOB text as a String, the ID of the text in the database, information about the language of the text as well as a summary.
- int audioIndex – the index number corresponding to the index the image has in the result set. This is used to identify and retrieve the audio temporarily stored on the mobile phone's memory card.

Each of the methods called in the code snippet will download relevant information or data related to the image. For example executeGetImage() will download an image, identified by the current set number and the current image number. Similarly getObjectText() will download the text and put it into a text-file, while getObjectAudio() will download the audio, which is then stored on the mobile phones memory card. Both methods are rather simple, and basically perform a call to the MMIR3 server via a HTTP.GET request, but due to the length of the implementation they are not included in this report. See the full source code for more information.

An example of the HTTP.GET request executeGetObjectText() sends to the server is:

http://bulmeurt.uib.no:8080/caimserver/mmir?command=getObjectText&setNumber=0&pictureNumber=0

The url is marked with the command tag, which states what the client wants from the server; the setNumber tag, which states which set it wants to download text from; and the pictureNumber tag, which states which setImage in the set it wants to download text from. In the example described above, it will download the text from the first setImage, in the first set in the resultset. This is similar for retrieval of audio, but the command is switched from getObjectText to getObjectAudio.

When this URL is executed as a HTTP.GET request against the MMIR server, it will parse the

content of the string and the value of the variables, and perform actions based on the contents of the variables it parses. Since it is a HTTP.GET request, it will be picked up by the method doGet() in the CaimServlet, located in no.uib.caim.servlet in the MMIR server. doGet() checks the command and calls the proper methods based on the input:

```
else if (command.equals("getObjectText")) {
      getObjectText(setNumber, pictureNumber);
      System.out.println("CAIMServlet: fetchText");
}
else if (command.equals("getObjectAudio"))  {
      getObjectAudio(setNumber, pictureNumber);
      System.out.println("CAIMServlet: fetchAudio");
}
```

When either getObjectText() or getObjectAudio() is called on the MMIR server[7], they find the information requested from the result set stored in the server and sends this to the client that requested it.

5 Evaluation

After the development of MMIR3 was completed, testing of the Object specification and Query-by-sketch features was performed.

5.1 Query-by-sketch evaluation

Testing was performed by drawing images on the query-by-sketch feature on the MMIR3 prototype, using the Nokia 5800 XpressMusic mobile phone and stylus. Most of the sketches are quick sketches, with drawing time being approximately 5 minutes for each sketch. A total of eight sketches were created, and the tests were performed using three different CBIR weights.

Some of the results are generated by feeding the sketches created on MMIR3 through VISI3's similar image search features.

The average precision score seems to be:

P4	**P8**	**P12**	**P16**
0.09	0.08	0.06	0.06

7 Note that this is the getObjectText() and getObjectAudio() in the server, and must not be confused with the methods with same names in the MMIR3 client.

Note that the results are calculated as an average for three different tests, using the same images and the Oracle CBIR treshold **0.25, 0.50** and **0.75**.

The average result for Query-by-sketch is rather low. This might be caused by the fact that most of the images in the BergenBy DB are colour photographs with a lot of detail, while the images created by the Query-by-sketch feature are low-resolution images with a limited set of colour, typically containing very little detail.

Implementing more colours in the Query-by-sketch feature, as well as expanding the BergenBy DB with more sketches (either hand drawn or computer generated) of the different objects, might improve the performance.

What is interesting in the results from Query-by-sketch is that in two of the three test cases (using CBIR treshold 0.50 and 0.75 respectively), three of eight images returned a relevant result in the first four images. This means that even though the average precision for all the tests is poor, using the four-image model of retrieval, 3 of 8 eight searches included a relevant response in the 1st result set. If we score precision for each page, i.e. counting how many pages have relevant image(s), the precision averages for treshold 50 would be:

P4	**P8**	**P12**	**P16**
0.38	0.31	0.25	0.22

5.2 Object specification evaluation

Testing was performed by letting the testers walk the streets of Bergen, manually photographing the objects located in BergenBy DB using the object selection feature of the MMIR3 prototype. The mobile phone used was the same as the one used in the development of the prototype: the Nokia 5800 XpressMusic mobile phone. The testers subjectivly decided which parts of an image which was to be deemed important, and performed the selection as they saw fit.

A total of 65 searches were performed, ranging between one and three searches per object, divided on a total of 32 objects. Some of the objects did only get one search, as they were ranked as difficult to photograph because of their location, the current weather or other reasons making it hard to get varied photos. Some of the objects was photographed three times. The reason for this was most often very poor results in the two preceding searches, and the testers deciding that the object could

be photographed from a better angle or distance.

The average precision seems to be:

P4	**P8**	**P12**	**P16**
0.04	0.04	0.04	0.03

The performance is rather low. This might be caused by the fact that the selection search does not search in the normal image table at BergenBy DB, but rather in a specially designed tile table. The tile table contains every image in the image table, divided into nine parts, with the intention that some of these tiles are good matches for the object selection performed on the MMIR3 prototype. However, since these parts are divided by an algorithm and not a human, the result images might not capture the object in the photo very good.

To improve the results, another algorithm for dividing the images can be created, dividing the images into pieces which overlap somewhat. In addition to this, the object specification feature can be extended to search not only in the tile table, but also in the normal image table, hopefully increasing the overall precision.

Although the precision score is rather low, if we look at the results using the four-image model of retrieval, we can see that that for every result containing four images, the precision score is much higher than the one presented when we looked at only one image at a time. This seems to be:

P4	**P8**	**P12**	**P16**
0.17	0.15	0.13	0.13

6 Future development

- Add feature making it possible to see the last used search image by adding menu entry pointing to it, with the option of making selections in the image, and using it for new queries
- Add feature making it possible to use selection in CBIR search on Image table
- Add feature making it possible to use selection in CBIR search in both Image and Tile table
- Add feature making it possible to use image from result set in new query

- Improve search speed by storing smaller version of images for phone in database, rather than resizing on the fly like today.
- Use caching in mobile phone to reduce download necessary. For example could the amount of data needed to be downloaded be greatly reduced by saving the images either in memory or on disk.

7 User manual

This chapter contains a walk through of the new features in MMIR3.

7.1 Main menu

As in MMIR2, the main menu (Figure 6) is the first thing the user will see when he or she starts the program. From here the user will be able to access all the different features and technologies available in MMIR3.

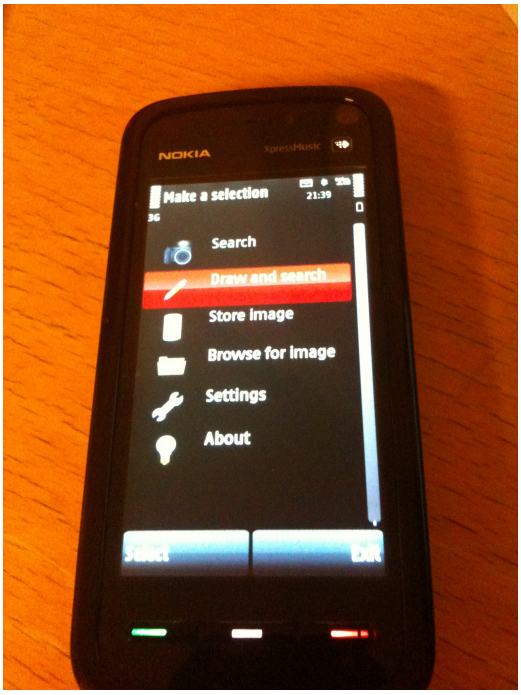


Figure 6: Main menu

MMIR3 contains some changes in the Main Menu.

- The **CBIR Search** feature is now named **Search**

- **CBIR Store** has changed name to **Store image**
- The new query-by-sketch feature is implemented under the label **Draw and search**.

Other than that, the Main Menu is the same as in MMIR2.

7.2 Object selection (Search button)

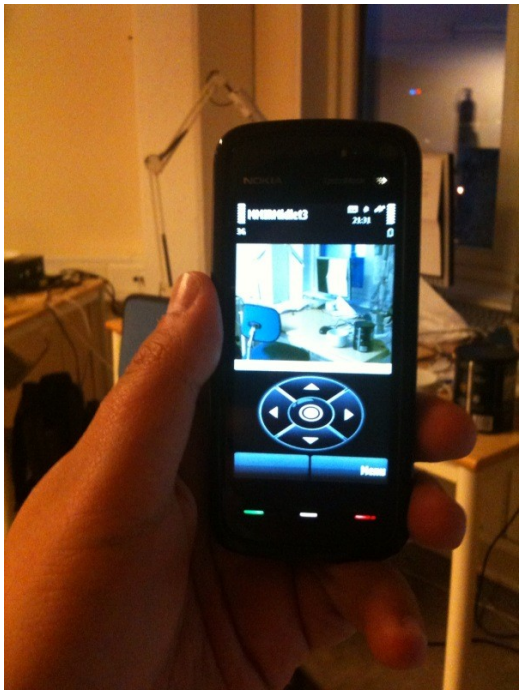


Figure 7: Camera view

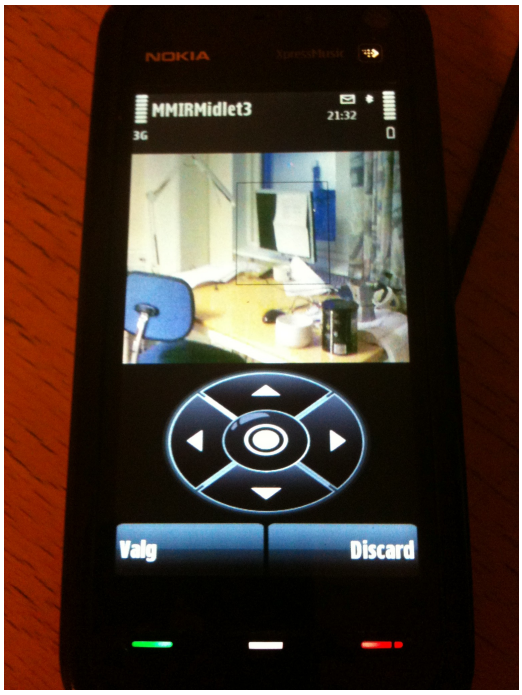


Figure 8: Captured image with selection performed

When the user enters the Search menu choice, he or she will be presented with a screen sporting six buttons and a live view of the camera (see Figure 7). This is the default camera view, and is used for both regular CBIR search and object specification CBIR search. There will also be a red or green dot indicating whether the GPS has found a position (not shown in image).

The user has the option of either pressing the **centre** button, or pressing the "**Menu**"-button located in the lower right corner. Pressing the **menu** button will take the user back to the Main Menu, while pressing the **centre** button will capture an image and present this on the screen.

Assuming that the user chooses to take a photo using the **center** button, the resulting image will be displayed on the screen, along with two buttons. Figure 8 shows the MMI3 prototype with a photo shown on the screen. On the left is the **Option / Valg** button[8], and on the right you will find the

8 In Figure 8 the option button is named "Valg", due to the fact that the Nokia 5800 XpressMusic mobile phone used

Discard-button.

The user will also be able to perform selections in the image by dragging the stylus across the screen as if drawing a line from one point to another. Shown in Figure 8, the user has selected the LCD monitor by dragging the stylus from the top left corner of the monitor, to the bottom right corner of the monitor, thereby creating a rectangle.

The **Option / Valg** button will present the user with the option of either performing a search (by pressing the **Search**-button), or clearing any selection performed on the screen (by pressing the **Clear selection** button) (Figure 9).

If the user presses the **Search** button, one of two things will happen:

- If an object selection is performed, the image will be processed and sent to the server as a selection search / object specification search.

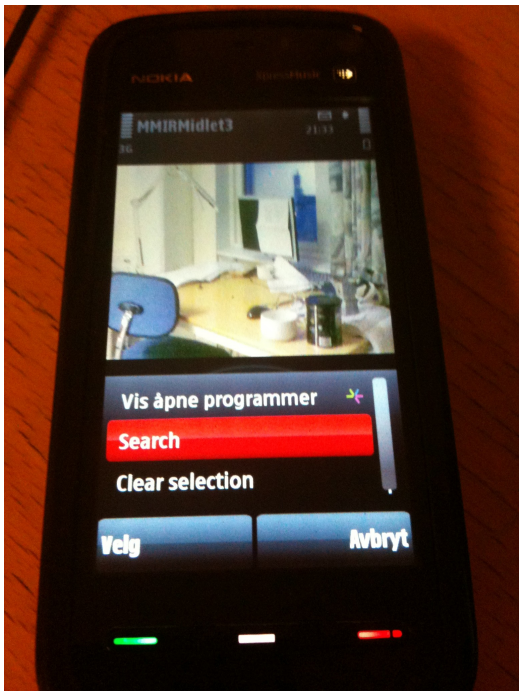


Figure 9: Option button pressed in Search view

- If no object selection is performed, the image will be processed and sent to the server as a CBIR search.

for testing was set up with Norwegian as language rather than English.

If the user presses the **Clear selection**-button, any selections on the screen will disappear, and the user is free to either perform regular CBIR searches or making new selections.

If the user presses the **Search** button, a selection search will be performed. The details on how this is done are described further in section **4.3 Object selection**.

7.3 Query-by-sketch (Draw and search button)

When the user enters the **Draw and search** option from the Main Menu, he or she is accessing the new query-by-sketch feature. The user will be presented with a white canvas and seven buttons. Only two of the buttons are active here: the **Option / Valg** button located in the lower left corner, and the **Menu** button, located in the lower right corner. See Figure 10.

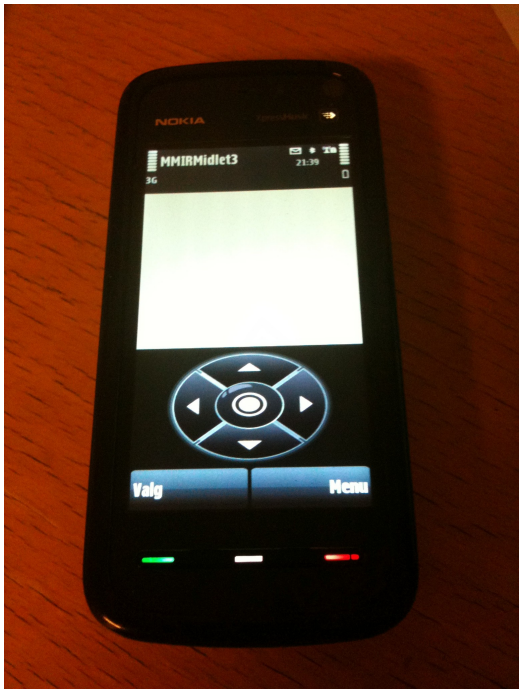


Figure 10: Query-by-sketch

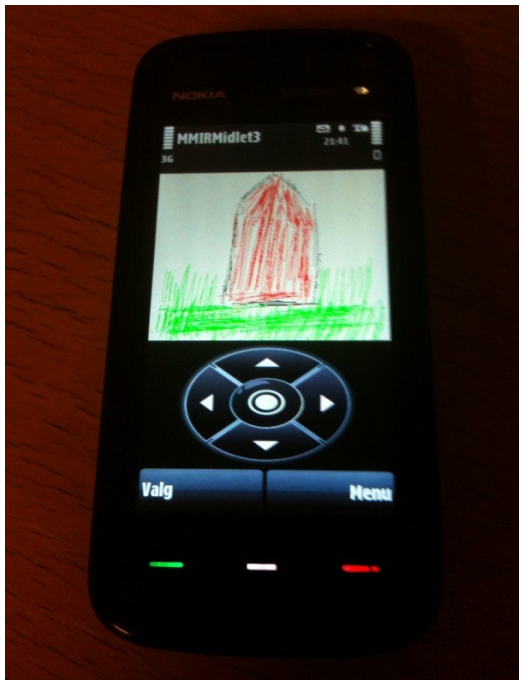


Figure 11: Query-by-sketch depicting a nice, red house

The white canvas is a painting board, and its intention is for users to sketch quick sketches of buildings or objects they wish to find information about. Sketches are made by dragging the phone's stylus across the screen, creating lines and dots. Figure 11 depicts the canvas showing a quick sketch.

Pressing the option button will open a menu with the following entries: **Search**, **Clear screen**, and **Select colour**. Selecting the **Search** button will activate an image search, using the sketch as a seed image (as described in section **4.2 Query-by-sketch**). Selecting the **Clear screen** button will remove the image painted on the screen, and the user is again presented with the white screen. Selecting the **Select colour** button will open a separate menu listing out the various colours implemented in MMIR3, making it possible for the user to select a different colour for his "pencil". See Figure 12 for example.

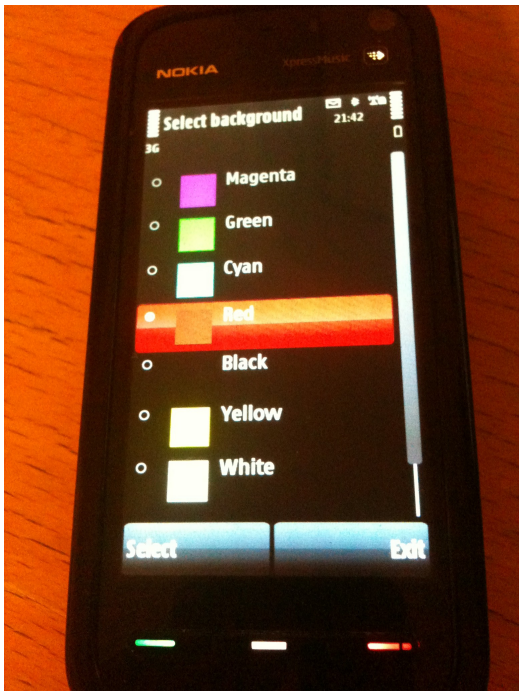


Figure 12: Selection of colour in query-by-sketch

7.4 Downloading of text and audio for image

After a user has performed an image search - be it a standard image search using the camera, a search using object selection or a search using sketches from the query-by-sketch tool - MMIR3 will automatically download the first four images in the result set, along with relevant audio and relevant text.

The result set will be displayed four images at a time, similar to the presentation in MMIR2. Figure 13 shows an example of this. The reason for presenting four images rather than e.g. two, six or eight is that we assume four images is a good compromise between quality and quantity. The user doesn't

have to click next too many times to get to e.g. the eight image, while the images remain large enough for the user to discern. The mobile phone used in the testing and development of MMIR3 utilizes a higher resolution than the mobile phone used for MMIR2, so the amount of images presented in MMIR3 could possibly be increased in future versions, without compromising on the quality (i.e. size) of the images. This was not done in MMIR3 as we wanted to maintain compatibility with MMIR2.

The result set has two buttons, the option button in the left hand corner, and the **Next** or **Menu** button, in the right hand corner. Whether it is next or menu in the right hand corner is dependent upon whether there are any more images in the result set.

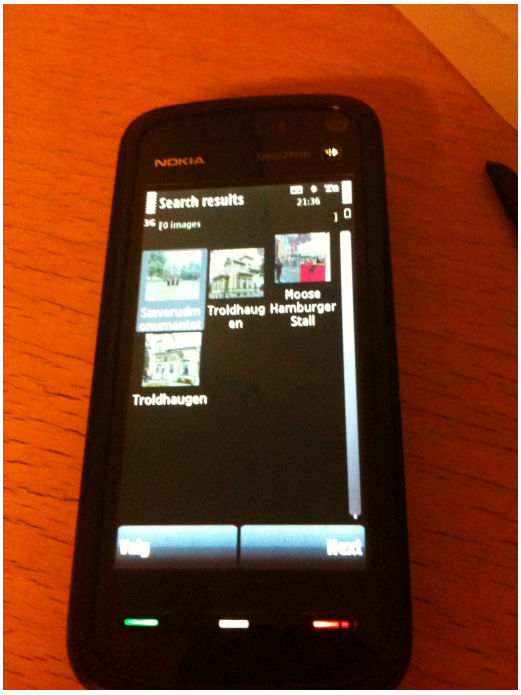


Figure 13: Result set

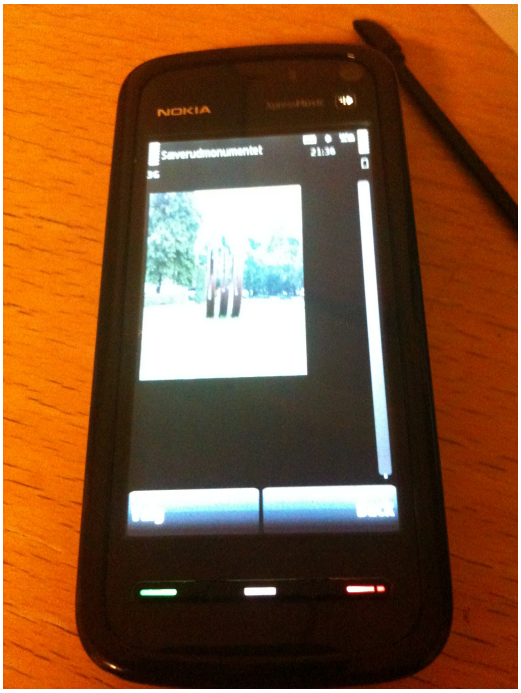


Figure 14: Fullscreen view

Pressing the **Next** button will download the next four[9] images in the result set, while pressing **Menu** will take the user back to the Main Menu. Pressing the **Option / Valg** button will give you another **Menu** button, and a button named **Open**, which will open a bigger version of the image.

The user has the possibility of selecting any of the four images on the screen, using the stylus. The image selected is the one opened in higher resolution using the **Open**-button.

9 Or less, if there aren't four images left in the result set.

Assuming the user has pressed **Open** (Figure 14), he or she will see a larger version of the image, and be presented with another **Option** button, as well as a **Back** button. The **Back** button will take the user back to the result set (Figure 13), while pressing the **Option / Valg** button will give the user the choice of either playing the audio or reading the text. This is done through the buttons **View info** and **Play audio**. See Figure 15.

Clicking the **View info** button will open a view of the text related to the object. Users have the option of either pressing **Back**, which will take them to the result set, or pressing **View image**, which will take them back to the image. See Figure 16.

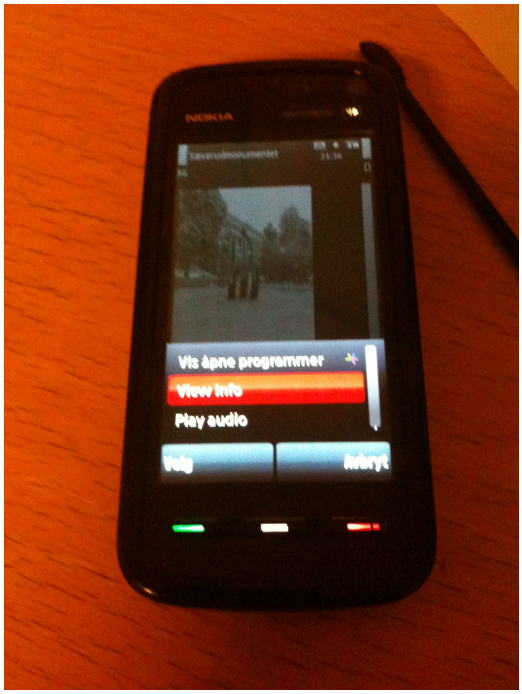


Figure 15: View info and play audio

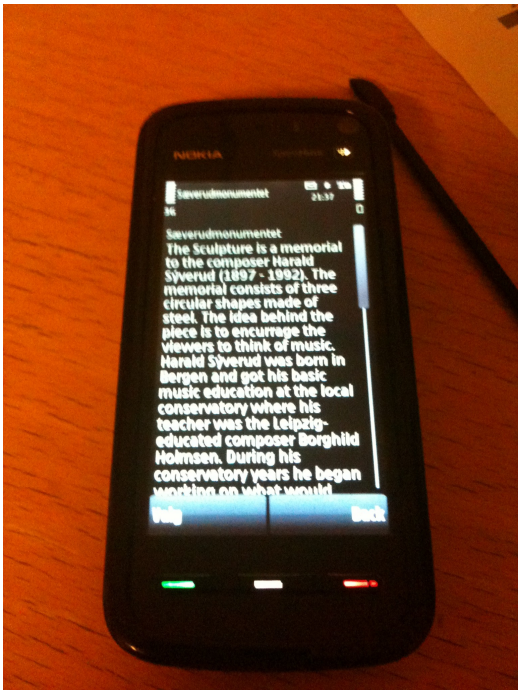


Figure 16: Text

8 References

Carlson, Christoph. (Sept. 2009). *VISI3 – Context Aware Image Retrieval*. CAIM-TR-8, Dept. of Information and Media Sciences, Univ. of Bergen.

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

Møller, Thor. (Sept. 2009). *Bergen By – a Multi-Modal Image Database*. CAIM-TR-9,

Dept. of Information and Media Sciences, Univ. of Bergen.

9 Appendix

Appendix A – Source code

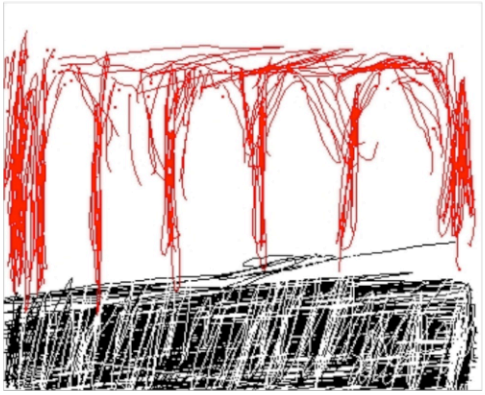
Due to copyright and space consideration the source code for the MMIR3 prototype (The client and the server) will not be included in this report. CAIM is free to distribute the source code as they see fit, so if you are in need of the source code for the MMIR3 client or server, contact CAIM-UiB.

Appendix B – Query by sketch / object selection images

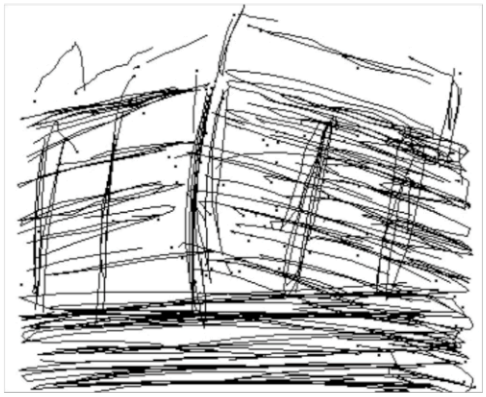
The images created during testing of the MMIR3 query by sketch and object selections features are included as Appendix B. Page 1 of appendix B presents the drawn seed images used for testing of query by sketch, while pages 2-7 show the parts of images selected by the object selection feature.

Version Name: mariakirken

Version Name: edvard grieg statue



Version Name: borsbygningen_1



Version Name: borsbygningen_2

Version Name: domkirken

Version Name: johanneskirken

Version Name: christian michelsen statue

Version Name: henrikibsen

Caption: Bergen kunstmuseeum, Lysverket

Caption: Bergen kunstmuseeum, Lysverket

Caption: Bergen kunstmuseeum, Lysverket

Caption: Bergen Kunsthall

Caption: Lille lungegårdsvannet

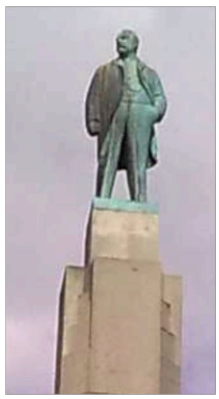
Caption: Lille lungegårdsvannet

Caption: Bergen Kunsthall

Caption: Vestlandske kunstindustrimuseum, Permanenten

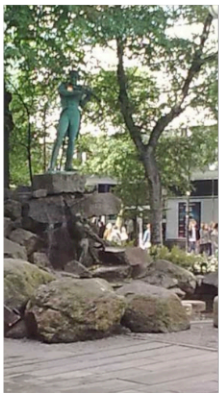
Caption: Festplassen

Caption: Christian Michelsens statue

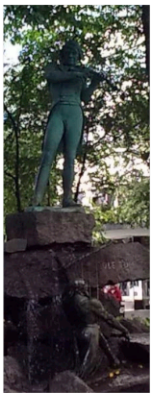


Caption: Christian Michelsens statue

Caption: Musikkpaviljongen

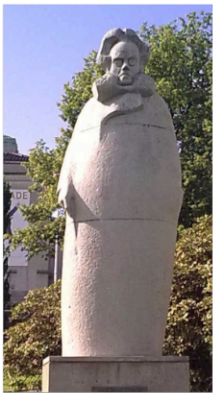


Caption: Ole Bull statue



Caption: Ole Bull statue

Caption: Henrik Ibsen statue



Caption: Henrik Ibsen statue

Caption: Den Nationale Scene

Caption: Den Nationale Scene

Caption: Den Nationale Scene

Caption: Akvariet i Bergen

Caption: Akvariet i Bergen

Caption: Totempålen på Nordnes

Caption: Totempålen på Nordnes

Caption: Totempålen på Nordnes

Caption: Nykirken

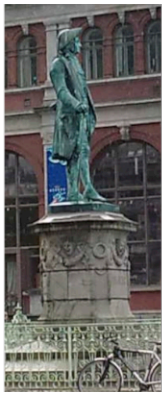
Caption: Den blå stein

Caption: Den blå stein

Caption: Sjøfartsmonumentet

Caption: Sjøfartsmonumentet

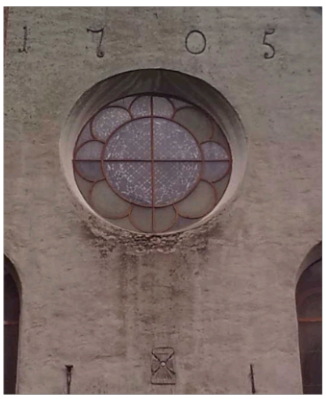
Caption: Ludvig Holberg statue



Caption: Ludvig Holberg statue

Caption: Zachariasbryggen

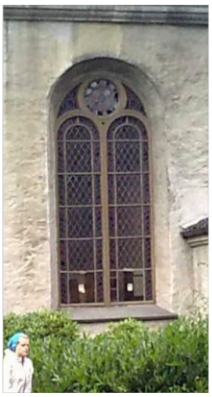
Caption: Zachariasbryggen



Caption: Korskirken

Caption: Korskirken

Caption: Korskirken

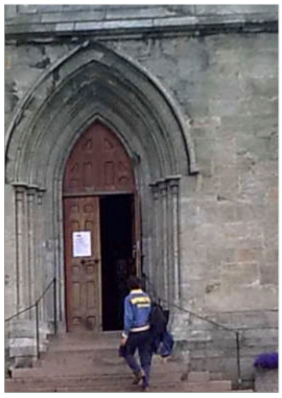


Caption: Korskirken

Caption: Domkirken

Caption: Domkirken

Caption: Domkirken



Caption: Domkirken

Caption: Stadsporten

Caption: Stadsporten

Caption: Sentrum Fløibanestasjon

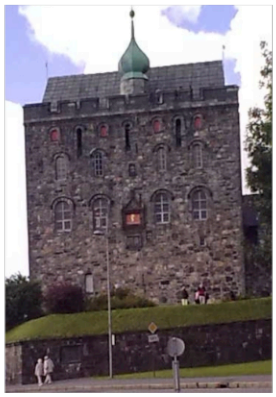
Caption: Sentrum Fløibanestasjon

Caption: Sentrum Fløibanestasjon

Caption: Bryggen

Caption: Bryggen

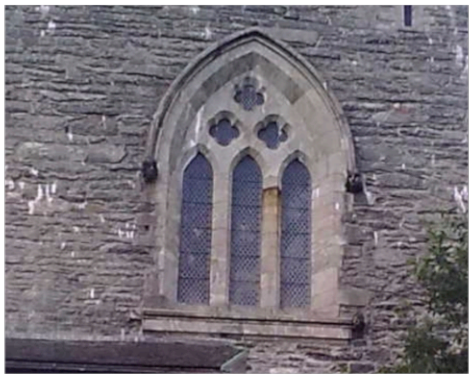
Caption: Rosenkrantztårnet



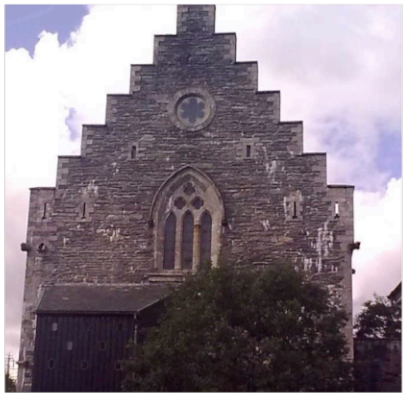
Caption: Rosenkrantztårnet

Caption: Rosenkrantztårnet

Caption: Håkonshallen



Caption: Håkonshallen

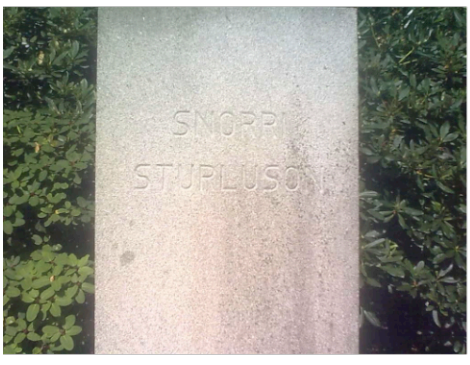


Caption: Håkonshallen

Caption: Mariakirken

Caption: Mariakirken

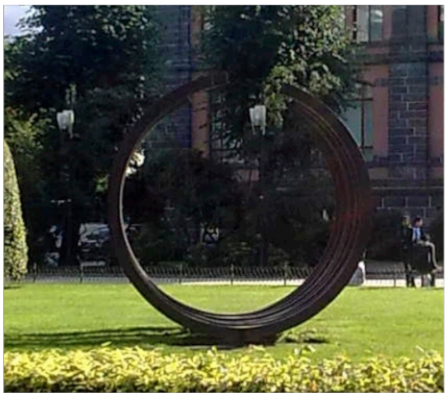
Caption: Mariakirken



Caption: Mariakirken

Caption: Børsbygningen

Caption: Børsbygningen



Caption: Sæverudmonumentet

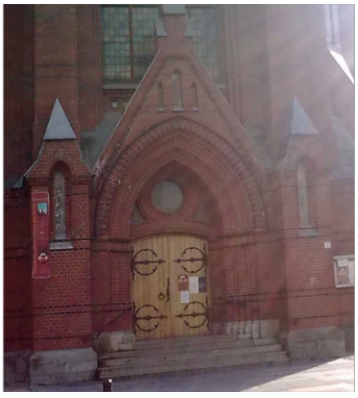
Caption: Sæverudmonumentet

Caption: Grieghallen

Caption: Grieghallen

Caption: Naturhistorisk museum

Caption: Naturhistorisk museum



Caption: Johanneskirken

Caption: Kulturhistorisk museum

Caption: Kulturhistorisk museum