

**A Framework for Visual Features Database Creation for
Building Recognition on Mobile Devices**

by

Marco Conti

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment of the requirements

for the Degree of

**Master of Science in Computer Science
(Interactive Entertainment Technology)**

University of Dublin, Trinity College

September 2010

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Marco Conti

September 13, 2010

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Marco Conti

September 13, 2010

Acknowledgments

I would like to thank my supervisor Dr. Gerard Lacey for providing the initial idea for this dissertation and for the advices he provided through the duration of this project.

I would also like to thank my classmates in the MSc. IET course for the fun and the hard times we shared in our late working nights.

MARCO CONTI

University of Dublin, Trinity College

September 2010

A Framework for Visual Features Database Creation for Building Recognition on Mobile Devices

Marco Conti

University of Dublin, Trinity College, 2010

Supervisor: Gerard Lacey

We propose the design and development of a framework for the creation of small visual features database. This database is to be used on mobile devices to perform building recognition on a self-contained “*tell me what I am looking at*” application using two inputs: GPS data and camera images.

The main contribution of our approach is exploring the automated creation of a compact local visual features database to be installed on the mobile device. Using a local database is justified by scenarios where a data connection to a remote server is not available or too expensive (e.g. tourists using data roaming abroad).

Creating a compact database requires a balance between various constraints. The number of visual features in the database will affects both the size of the database on the limited storage of a mobile platform and the computation time of the image matching. However, having a small number of features in the database also results in poor results. This project evaluate the use of a genetic algorithm that will select the best parameters to build the database using visual features clustering.

Contents

Acknowledgments	iv
Abstract	v
List of Tables	ix
List of Figures	x
Chapter 1 Introduction	1
1.1 Overview and motivations	1
1.2 Structure of this document	4
Chapter 2 State of the art	6
2.1 Visual features extractor and descriptors	6
2.1.1 SIFT	6
2.1.2 Evolution of SIFT	8
2.2 Building recognition for augmented reality	9
2.3 Commercial applications	11
2.3.1 Location based	12
2.3.2 Image recognition based	13
Chapter 3 Background	15
3.1 Android	15
3.1.1 Developing for the Android	16

3.2	SURF	17
3.3	Nearest neighbour search	20
3.4	Genetic algorithm	21
3.5	Clustering	23
3.5.1	K-Mean	23
3.5.2	Quality Threshold	24
Chapter 4 Design		26
4.1	Overview	26
4.2	Location data	27
4.3	Shared functionality	28
4.3.1	Feature descriptor	28
4.3.2	Feature matching	28
4.4	Feature database	30
4.4.1	Training sets	30
4.4.2	Clustering	31
4.5	Genetic algorithm	33
4.5.1	Fitness function	35
4.5.2	Parameters tweaked	37
4.6	Mobile prototype	39
4.6.1	Interface	39
4.6.2	Matching and database	40
4.7	Tests	41
Chapter 5 Implementation		42
5.1	Software and Hardware	42
5.2	Shared components	43
5.2.1	SURF	43
5.2.2	Feature matching	44
5.3	Database creation	45
5.3.1	Clustering	45

5.3.2	Visual feedback	45
5.3.3	Genetic algorithm	46
5.4	Android prototype	47
5.4.1	Interface	47
5.4.2	Database and matching	48
5.5	Tests	49
5.5.1	Images acquisition	49
Chapter 6	Results	50
6.1	Genetic algorithm	50
6.2	Database creation	52
6.2.1	ALL FEATURES approach	52
6.2.2	CLUSTERING approach	54
6.2.3	TWO STAGES CLUSTERING approach	58
6.3	Matching time and database size on the Android device	59
6.4	Comparison with previous works	60
Chapter 7	Conclusions	63
7.1	Considerations	63
7.2	Future work	64
7.2.1	Tests and comparison	65
7.2.2	Database creation	65
7.2.3	Feature matching	66
7.2.4	SURF	66
Bibliography		67

List of Tables

4.1	EASY set of buildings	30
4.2	CHALLENGING set of buildings	31
6.1	Genetic algorithm evaluations for fitness function comparison, <i>ALL FEATURES</i> approach, camera images	52
6.2	GA and tests results for <i>ALL FEATURES</i> approach, camera images, 1000 iterations .	53
6.3	GA and tests results for <i>ALL FEATURES</i> approach, mobile images, 500 iterations . .	54
6.4	GA and tests results for <i>CLUSTERING</i> approach, camera images, 500 iterations . . .	55
6.5	GA and tests results for <i>CLUSTERING</i> approach, mobile images, 500 iterations . . .	58
6.6	GA results for <i>TWO STAGES CLUSTERING</i> with camera images	59
6.7	Matching times on the Android device	60
6.8	Disk space occupied on the Android device	61
6.9	Comparison with previous works	61

List of Figures

1.1	The concept of our “ <i>tell me what I am looking at</i> ” application	2
1.2	The self-contained application inputs	3
1.3	The framework pipeline	4
2.1	An example of metafeatures on the Christchurch Cathedral in Dublin	11
3.1	The Android stack architecture (Copyright Google Inc., Apache 2.0 licence)	16
3.2	Box filters approximating the Laplacian of Gaussian for the xy , x and y direction respectively	19
3.3	Haar wavelets used to compute the response in the x and y direction respectively . . .	19
4.1	The matching step	29
4.2	Database creation approaches	31
4.3	Wrong cluster assignment when considering only the cluster centre	32
4.4	The fitness evaluation pipeline	35
4.5	The PARABOLOID (left) and PAIRWISE (right) fitness functions	36
4.6	The parameters (red) tweaked during database creation	38
4.7	The complete set of parameters (red) tweaked by the genetic algorithm	39
5.1	A screen shot of the visual interface used for testing and debugging	46
5.2	Screen shots of the Android prototype on the emulator, before (left) and after (right) matching	48
5.3	The relational database schema	48

6.1	Convergence of the genetic algorithm with the <i>ALL FEATURES</i> approach, camera images	51
-----	--	----

Chapter 1

Introduction

1.1 Overview and motivations

Mobile phones offer a unique combination of features: they are easily accessible, can be carried everywhere, have photo/video capabilities and are getting powerful enough to perform complex tasks. They are an attractive platform for *see-through* augmented reality applications. This type of applications acquires video frames through the camera, processes these images and then displays the live video on the device screen with some overlaid information and graphics. The mobile platform is effectively transformed into a looking glass the user can use to explore the world.

This project aims to use mobile augmented reality in buildings recognition. Our goal is to have a “*tell me what I am looking at*” application usable to explore a city through the viewfinder (see Figure 1.1). The application would not rely exclusively on location (GPS/compass data) but will also “*see*” what the user is pointing the camera at. This will make the difference not only when the GPS is not accurate enough (and this is often the case in city centres with narrow streets and tall buildings) but also when there is an occlusion between the user and the object - a commercial billboard, a tree, another building. With a location-only approach, the user might be pointing the device at a hot-dog stand and see it labeled as a bank, just because there is a bank on the other side of the road. Our approach wants to avoid erroneous labelling caused both by occlusion or location inaccuracy.



Figure 1.1: The concept of our “tell me what I am looking at” application

This result can be achieved by performing an image matching between the image captured by the camera and an image database. The images that we want to match (in our case, the facades of various buildings) need to be acquired during the preliminary step. The images are then used to create a database of *visual features* that are extracted using a feature extractor. Visual features are points in the image that stand out because of some distinctive characteristics in the surrounding region (e.g. sharp edges, corners, etc.); they can be mathematically described and compared with other features for similarity. Once a features database has been created, new images can be matched with the database by extracting the features from the new image and looking for similar features in the database. This technique is known in literature as *Context-Based Image Retrieval* (CIBR) and is a widely explored topic.

The images used to build the database can be *geotagged*. This means that they can be tagged with the location where the picture was taken and that the visual features extracted from that image will maintain this location tag. This is where the location data and the image matching approaches are combined: when matching an image on the mobile device, the current position of the device is acquired through GPS and the database is queried for the visual features that have been tagged with the same location. Since the matching time increase with the number of compared features, limiting the search space to the features that are expected in that geographical area will reduce the computational time.

This approach has already been explored in literature and has recently become popular in commercial applications on various mobile platforms. Most of the solutions already available on a mobile device are based on various degrees on a remote server. Using the client-server paradigm, the mobile application capture an image and then either send the picture (or the extracted features) to a server for matching or query the server to download the visual features associated with the current geographical cell. All these approaches obviously require a data connection and this where our project main contribution is. Our

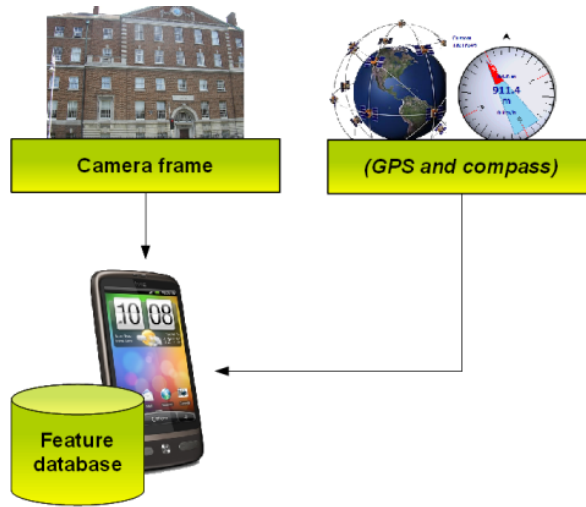


Figure 1.2: The self-contained application inputs

final goal is to have a self-contained application installable on the mobile device that doesn't require any data connection; the database will be transferred only once during the application installation. This is a crucial aspect for some applications, the most obvious being tourism-related ones, for when travelling abroad data roaming costs are an obstacle to client-server approach. Our idea is illustrated in Figure 1.2.

There is of course a reason why this approach hasn't been popular so far. Features databases can be huge and the storage space available on a mobile device is limited. Furthermore, some application simply send the image (or the visual features extracted) to the remote server and this is where the matching step is executed, as this can require too much time to be performed on the limited computational resources of the mobile device. However, mobile devices are evolving, they have better CPUs and storage capacity (both internal and external) and new publishing channels allow developers to create a number of versions of an application with location-specific content and make them easily available to download for users.

This approach is validated in this project and a self-contained solution is demonstrated. The key is to have a small database so that a big number of facades can be included on the mobile device itself and the matching step can be fast enough. However, the database should also contain enough features to have good matches and minimise the identification errors and this contrasts with the previous requirement. To find the best balance between these two constraints, a genetic algorithm has been

implemented. The genetic algorithm will find the best database for the given input (i.e. training images for a given location) without the need of tweaking the database by hand. The creation of databases is therefore automatised and can be performed in batch for various location.

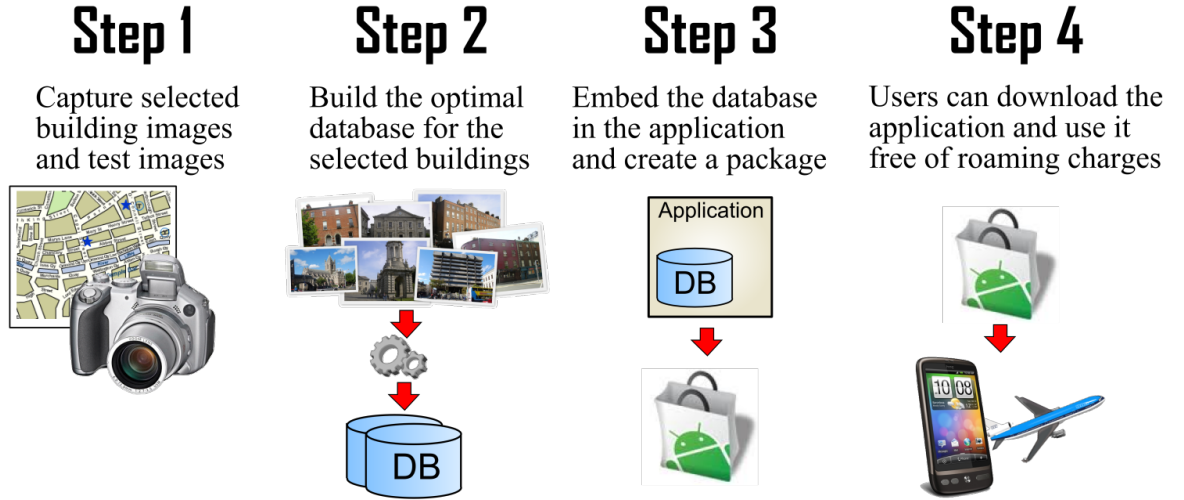


Figure 1.3: The framework pipeline

The framework implemented in this project is composed of two main components: a database creation application, running on a PC, and a building recognition prototype running on an Android device. The intended framework usage is summarised in Figure 1.3.

1.2 Structure of this document

The rest of this document is organised as follow.

In Chapter 2 the reader will find an assessment of the state of the art covering feature extraction technologies as well as research and commercial works addressing the same area of interest as our project.

Chapter 3 covers the background on technologies, methodologies and algorithms that have been considered in this project.

Chapter 4 presents the design of our framework and motivate the choices that have been made.

Chapter 5 illustrates details of the implementation for both the PC component and the Android application.

In Chapter 6 we show the collected results and evaluate our framework and the various approaches that have been included in it.

Finally, Chapter 7 spends some words on the conclusions we can draw from the results and on how the framework can be improved and expanded.

Chapter 2

State of the art

The area of image matching, more specifically building recognition, has been already widely explored both in the academic and commercial context. We will now discuss the state of the art on applications for building recognition on handheld devices and on some key underlying technologies.

2.1 Visual features extractor and descriptors

Our review will start from visual features extractors and descriptors, as they form the core of this project. Visual features are points of interest in an image that present some distinctive traits in a local neighbourhood of the point. Ideally a feature should be reproducible and stable to local and global perturbation such as noise, illumination changes, rotations, distortions.

A wide range of choices for features extractor and descriptors is available in literature. Describing or simply listing all the them falls outside the purpose of this document, so we just focus a few selected interesting ones providing a quick overview rather than an exhaustive mathematical description.

2.1.1 SIFT

The breakthrough in this field - in terms of computational speed and stability - was achieved by Lowe with the Scale-Invariant Feature Transform (SIFT) in 1999 [22] and then subsequently refined [23].

Lowe’s novel approach is based on concepts from biological vision resulting from studies on how the human vision works. The key is using local descriptors for features, i.e. descriptors that characterise small-scale features of the neighbourhood of the interest point in the image.

The SIFT algorithm starts by detecting *key points* in the image. Key points are locations in the image that are invariant to translation, scaling and rotation but also robust to noise and small distortions. Key points are searched in the image in the scale space, that is evaluating the image at different scales. They are identified by local maxima or minima in a Difference of Gaussian (DoG) function, computed efficiently by constructing a Gaussian-convolution pyramid of images with different σ values. In order to be an extreme, a point has to have a larger or smaller value than its eight neighbours in the current scale and the nine in the scale above and below.

Once a key point has been detected, its location has to be defined with sub-pixel accuracy, as the maxima or minima might not lie exactly on a pixel. In-between pixel values can be interpolated by a Taylor expansion of the DoG scale-space function around the original key point position. The position of the maxima or minima is found by solving for the derivative of the expansion.

Not all the maxima/minima will be stable or repeatable if some noise/distortion is applied to the image. Low-contrast features, where the distance to the closest other extrema is below a given threshold, are discarded. Points along edges are also discarded as their location is not stable. A point lying on an edge will have a strong gradient response across the edge direction and a smaller one on the perpendicular direction. Points on edges can then be detected and discarded by looking at the ratio between the two responses (computed as eigenvalues of an Hessian matrix, see 3.2).

The remaining points are assigned an orientation so that they can be described independently of rotations in the image. A histogram of gradient orientations is computed on the points surrounding each key point. The highest peak in the histogram is assigned as the orientation of the key point. Any other peaks above 80% of the highest peak are also considered by creating a new key point in the same position and scale and the orientation of the secondary peak, effectively generating a new feature.

The key points are finally described as a 128-dimensional vector by sampling the magnitude of gradients in a 4x4 grid of neighbours aligned to the point orientation; the neighbours are sampled by selecting the magnitude in 8 different directions (hence the 128 elements: 8x4x4). The set of feature

descriptors is now complete. Each feature descriptor include scale, orientation and position in the image and a 128-dimensional vector describing the neighbourhood of the key point.

The region-based SIFT descriptors outperforms any previous feature description ([25]) and are easy to compare using Euclidean distance on a 128-dimensional space. SIFT become a popular approach for image matching and has been the foundation of numerous image matching application ([4, 20, 37]).

2.1.2 Evolution of SIFT

SIFT has been refined and optimized and has been the starting point for other local features descriptors, such as *Gradient Location-Orientation Histogram* (GLOH), PCA-SIFT and *Speeded-Up Robust Features* (SURF). The first two extends the SIFT descriptors without affecting the key point detection step, while the later one provides a different extraction approach.

GLOH [25] is an extension of the SIFT descriptor that also describe the neighbourhood using a polar grid in addition to the 4x4 rectangular grid used by SIFT; this results in a bigger descriptor that is more accurate in describing the neighbourhood but also more computationally intensive to handle. To overcome this downside the descriptor is then reduced using Principal Component Analysis (PCA) to a 128 vector.

PCA-SIFT [17] is another extension of the SIFT descriptors that uses PCA to reduce a 3042-dimensional vector generated concatenating the horizontal and vertical gradient maps for a 41x41 patch centred at the key point. The resulting vector is a 36-dimensional vector, the smaller size resulting in a faster feature comparisons at the cost of accuracy.

The last feature descriptor, SURF [2], is the most recent of the ones listed here. Various comparisons between these feature descriptors are available in literature [36, 35, 25] and SURF (or its rotation variant version U-SURF) is reported to be faster and/or more robust than the other candidates. This is one of the main reason we decided to use SURF in our project and it will be discussed in detail in section 3.2.

2.2 Building recognition for augmented reality

Various building recognition approaches can be found in literature and they mostly rely on the feature extractors mentioned in 2.1. We will now quickly skim through some papers and comment on their approach and limitations.

One early concept of augmented reality for building recognition on mobile devices is mentioned in [30]. This paper assume that the user is looking at facades of building, hence at planar surface. A *wide-baseline matching* algorithm is implemented, where main edges of the building, assumed to be straight lines on a plane almost perpendicular to the camera, are used to rectify camera frames. Once the image is free of perspective distortions, a feature detection and description is run. The detection step is based on Harris corner detector and the descriptors comes from the RGB values of nearby pixels. The resulting features are compared with a database at two different scales; the matching features will produce the parameters for scale and translation transformation using RANSAC [10] and votes for a given combination.

Having the transformation from the reference image to the current frame, this allows for a silhouette of the detected building to be overlaid on the captured image. This early approach at visual features based building matching for mobile devices is somehow penalised by a poor feature extractor and descriptor. The performances average at 10 second per query on a regular 2004 PC.

The faster and more robust SIFT is used in [37], where a two stages approach is adopted. First, the image is compared with a database of images using “local” gradient histogram, where local refers to the direction of the perspective lines of the building. This technique is used to filter out negative matches before applying the second step, SIFT features matching, only on potential matches. The two stages approach of this paper succeeds in effectively reducing the search space for the matching phase on the test database used. The implementation is running at 2 seconds per query on a 2005 PC and would probably benefit from a faster feature detector, however no information is given on the size of the database.

An actual portable device, as opposed to the PCs used in the previous papers, is used in [28]. A combination of GPS, gyroscope and a tablet PC runs an edge tracking application capable of identifying and tracking building position. This paper presents an alternative approach compared to the

previous examined. Instead of using an image database to match visual features extracted from the camera frame, this paper use 3D textured models of the buildings that are rendered in an internal buffer of the device. An edge detector is run both on the rendered image and on the camera captured one and the movement of the camera is estimated (also including information from the other physical sensors). This process goes on as long as there is a previous estimation of the position for reference. This is not true when the application is started or when the continuity of the tracking is interrupted. In this cases, all the possible position of the camera in relation to the 3D model (according to the GPS data and accuracy) are evaluated by rendering the 3D model and then comparing the corners detected with FAST corner detector on both all the rendered images and the current frame. This recovery step is slow (some seconds) but once the tracking is running the processing speed is an impressive 17 Hz. The limitations of this approach are in depending on a continuous tracking and on a textured 3D model of the buildings. While city-wide textured models can now be easily available (i.e. Google Earth models), building an image-only database is still quite less demanding. Also mobile devices other than tablet PCs might have problem in rendering 3D models at the required frame rate, and finally we have no information on the size of the database.

The closest paper to our proposal is [32], where a SURF based image matching is used conjunction with a GPS on a normal mobile phone. The application created by the authors displays information about the building seen through the camera. This application require a connection to a remote server that, given the position as detected with the GPS, returns a set of features to match against the current captured image. The matching of features is computed using *kd*-search trees and the SURF algorithm is optimised by the authors obtaining interesting speed performances.

Additionally, the features in the database are clustered to identify strong features and reduce the number of features in the database by replacing them with *metafeatures*. Metafeatures are features that recur in various view of the same building. The position, orientation, size and descriptor vector of the extracted local feature can differs but they are all originated by the same physical feature (see Figure 2.1). Metafeatures can be detected by the distance between the vectors of descriptors; if two features are close enough, they might be generated by the same physical feature. The real scenario however can a bit more complicate as the same physical features can be repeated (e.g. a row of identical windows) and therefore generating the same visual features. The authors cluster features

into metafeatures using a graph-based approach. The graph used is generated from the results of a preliminary automatic labelling step performed on the database; this step is basically an image matching step using SURF to identify pictures containing the same buildings.



Figure 2.1: An example of metafeatures on the Christchurch Cathedral in Dublin

[32] has been a main inspiration for this project. We want to investigate the use of a local database in this type of application as opposed to having the remote database used in the paper. We also want to explore the automatic optimisation of the database for a given set of building using a different clustering approach for metafeatures.

2.3 Commercial applications

We believe localization based mobile augmented reality is still a new field not explored to the full extent of its potential by commercial companies. Since 2008 some commercial products have been released on the consumer market and in 2010 a big name like Google entered the field, while a lot of relatively simple applications are currently getting popular on mobile marketplaces. Reasons for this recent interest might be related to the new generation of consumer level mobiles, which now provide a widely available valid platform for this kind of application.

We now give a brief survey of the most prominent application related to location based mobile augmented reality and mobile feature detection that overlap to some extent with our proposal, with the two most used platform being Google Android-powered mobiles (see 3.1) and Apple's iPhones.

2.3.1 Location based

We will first cover mobile augmented reality applications that use the location data to display information in on the mobile display on top of the camera video feed.

The first commercial project to attract some attention was presented by Nokia in 2006, when Mobile Augmented Reality Application - **MARA** was presented as a sensor based augmented reality system for mobile [16]. The project used a GPS device connected to a Nokia S60 mobile, equipped with a standard camera, to overlay location based information on the video captured by the camera. Not many details were released about this project, now discontinued.

Both Layar [19] and Wikitude [26] are examples of currently available mobile augmented reality applications that rely on the location data to display directions and label buildings. Layar displays small icons and text in the direction of interesting location (which are fetched from a remote database) depending on to the direction the camera is pointing to according to GPS, electronic compass and accelerometer data. It was first announced in May 2009 and is now available for Android and iPhone platforms.

Wikitude is similar to Layar in displaying icons and text according to orientation data acquired through GPS, compass and accelerometer. It was launched in August 2009 and allows for community created content to be added on the remote database the mobile phones connect to. Wikitude is available on Android and iPhone.

Both these applications suffer from GPS and compass accuracy issues. While they perform well in optimal conditions, in many realistic scenarios the labels are notably offset and, due to drift in compass or accelerometer data, they can be slowly moving around even if the user is perfectly still. Additionally, the only value that discerns the visualisation of a given label is the distance as the crow flies. In case of a high number of points of interest in the nearby area, this will result in labels stacked on top of each other even if other buildings are interposing between the user and the points of interest, creating a cluttered feedback that doesn't reflect the visual perception of the user.

While applications resulting from the location based approach might look similar to the one proposed in our project, there is a substantial difference in how we get to the same result. In our approach, the location data is used only to reduce the search space of visual features to match, and the displayed

result is ultimately derived from an image matching process, while for these application the location data is the only input to determine if the user is pointing in the direction of a given building.

2.3.2 Image recognition based

Another set of commercial applications is more closely related to our approach in processing the camera frames to identify the object (be it a building or else) the user is pointing the device at.

Google released Google Goggles [13] in early 2010 as a multipurpose application capable of strong image recognition. Its uses ranges from books and wine labels recognition, to OCR and, most interesting for our point of view, landmarks recognition. This last feature is currently implemented on pictures taken with the camera and processed by a remote server and not on real time video. A simpler location based overlay (similar to the Layar or Wikitude approach) is provided with directions to shops and landmarks in real time.

Kooba [18] is a spin-off from the Computer Vision Lab at ETH Zurich, Switzerland, where SURF was designed. Currently they offer some visual search application (*kooba Visual Search*), automatic photo tagging (*Shooting Star*) and have an interesting video of *Smart Visuals* where buildings (among other things) are recognised on an Android device by image matching. Their visual matching database is reported to contain more than 10 millions of images and is of course accessed remotely by the application on the mobile device. However *Smart Visual* is still in its early stage and not available as a building recognition application.

While these products use our planned approach in detecting building, they rely on an external database that is accessed through a data connection. Our approach will use a local database where space constraints are strict but no data connection is required and this will be an advantage in some situations (e.g. data roaming costs).

A last markerless augmented reality application on mobile phone (Android) that is worth mentioning is Popcode ([9]). It overlays content (3D models) on markerless recognised images, tracking the movement so that the 3D models move according to the orientation and position of the recognised features. The overlay is triggered by a special activation marker (the Popcode logo) that contains a link to the remote resources to download and display but the feature database itself is stored on the

device.

Popcode has a Developer Kit that can be used to train Popcode to recognise any type of image (hence buildings too). However the documentation doesn't explain how the feature database is built. Also the documentation states that "training is currently quite a long process" but no precise time is reported. It would have been interesting to compare Popcode performances with our project but Popcode has been unveiled at the end of our project time frame (end of August 2010) and we couldn't investigate it in detail.

Chapter 3

Background

In this chapter we will cover technologies and methodologies that we will be referring to in the design and implementation.

3.1 Android

Android [12] is a software stack for mobile devices ideated by the Open Handset Alliance [33], a group of 78 technology and mobile companies, and developed by Google.

While traditionally mobile application had to be written in low-level C/C++ specific to a given hardware platform (or limited sets of platforms) [29], Android propose a different approach. Android exposes a common interface for developers independently of the underlying hardware. The same application developed for a given version of the Android interface will run on any device running that version and should adapt to the specific device characteristics (like screen resolution and available input devices). The onus of specialising the Android stack to the specific hardware is on the manufacturer and is transparent to application developers.

From a technical point of view, Android contains an operating system that is based on a Linux kernel and GNU software, a set of key application and a middleware layer providing a wide range of functionality. It is released under the open source Apache 2.0 license and its latest release at the time

of writing is Android 2.2 "Froyo" [12].

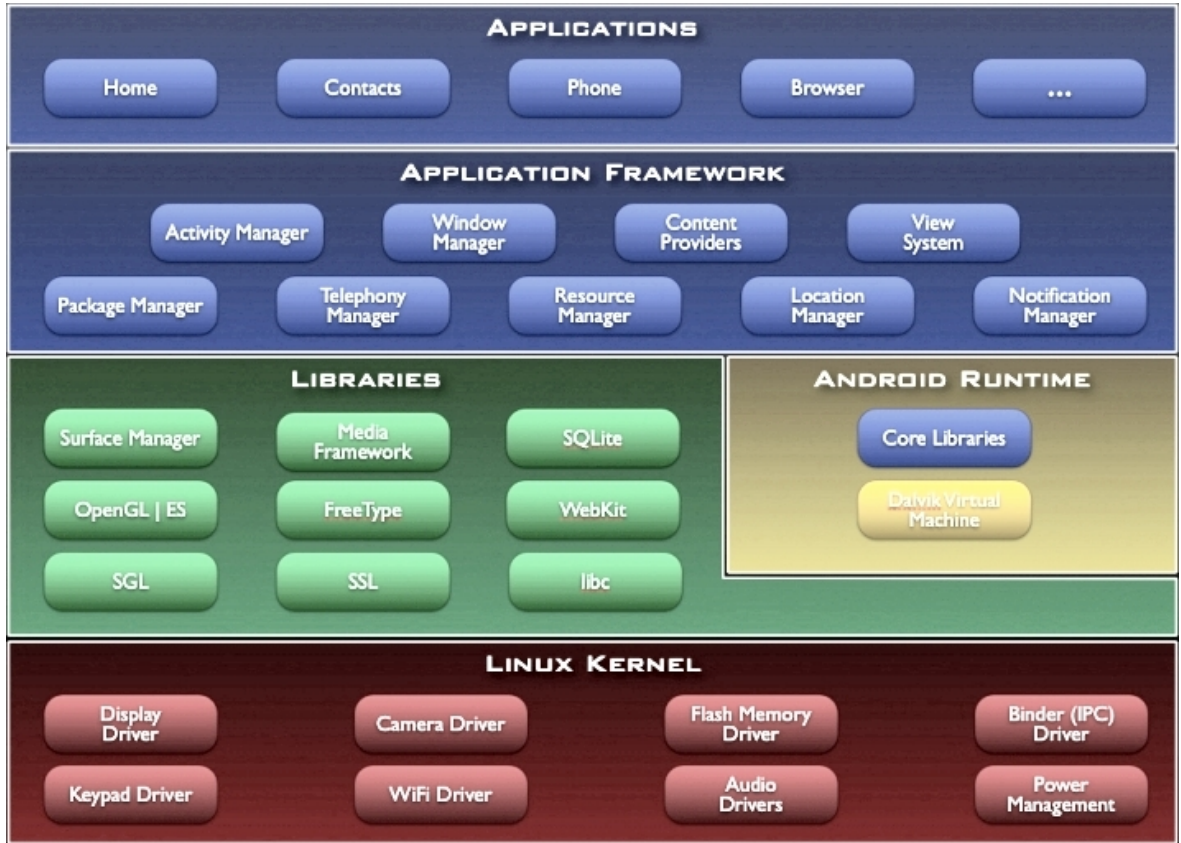


Figure 3.1: The Android stack architecture (Copyright Google Inc., Apache 2.0 licence)

Android applications are Java applications that run on Android's Dalvik virtual machine. Android's Java is a custom programming language related to the official Java 2 SE [27] from which it borrows the syntax and the main classes and framework structure. Java applications sit on top of a framework that allows for interaction with core libraries written in C/C++. These open source libraries provide features such as web browsing, 2D and 3D graphics, relational databases, multimedia playback and secure data transfer. A diagram of the architecture is shown in Figure 3.1.

3.1.1 Developing for the Android

Android requires the use of the *Android Software Development Kit* (SDK) [12] to develop applications for this environment. The SDK allows developers to write applications in Android's Java version;

but for high-performance code, developers can also use the *Android Native Development Kit* (NDK) to write components in C/C++ that can be then used from the main Java application. The NDK contains native system headers and libraries and a set of tools to generate Android-specific native code from C/C++ sources.

The NDK comes with a set of popular C/C++ headers like *libc*, *libm*, *OpenGL ES* and relies on the *Java Native Interface* (JNI) to interact with the containing Java application. However a notable lack in the NDK headers is the popular C++ *Standard Template Library* (STL) that provides functionality such as vectors and iterators.

We mentioned the importance of having a small database. Android applications have an size limit of 25 Mb which means that applications bigger than this size can be compiled but can not be installed on the device. Considering that the compiled bytecode will not usually be bigger than a couple of Mb, this leaves around 22Mb to be used for data and assets. There are method of overcoming this limit, namely saving data on the external memory of the device (SD memory card) using the PC or downloading, only once per application, the data to the external memory through a data connection. This will virtually increase the size limit of the data to the size of the external memory (usually 4 Gb) but requires additional steps from the final user before the application can be used.

3.2 SURF

SURF was first presented in 2006 as “*a novel scale- and rotation-invariant detector and descriptor*”. It shares the concept of local features descriptors based on the neighbourhood of the interest point already seen in SIFT, while it differs in how the interest points are selected and described. It uses the Fast Hessian detector for interest points selection and a novel feature descriptor. We will now briefly cover how SURF works as some details will be relevant in this project but without venturing deeply in the mathematical description.

The speed improvement achieved in SURF is mainly based on the use of *integral images*. An integral image can be rapidly computed from an input image and used to speed up the computation of the SURF descriptors for that image. The value of the integral image $I_{\Sigma}(x)$ in a point (x, y) is the sum

of all the pixel values of the input image I between the point and the origin.

$$I_{\Sigma}(x) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j) \quad (3.1)$$

The integral image allows for a fast calculation of the intensities over any upright rectangular area of the image by using only three additions regardless of the size of the image or of the area. Given a rectangular area determined by four A, B, C, D points, the sum of the intensities in the area R is:

$$R = I_{\Sigma}(A) - I_{\Sigma}(B) - I_{\Sigma}(C) + I_{\Sigma}(D) \quad (3.2)$$

This property is used in computing the determinant of the Hessian matrix. As in SIFT, SURF uses a 2x2 Hessian matrix of the image function $I(x, y)$ to detect maxima and minima of the function. The Hessian matrix of the image function at the scale σ in the point $X = (x, y)$ is:

$$H(X, \sigma) = \begin{vmatrix} L_{xx}(X, \sigma) & L_{xy}(X, \sigma) \\ L_{xy}(X, \sigma) & L_{yy}(X, \sigma) \end{vmatrix} \quad (3.3)$$

where $L_{xx}(X, \sigma)$ is the Laplacian of Gaussian with the image I in at the point X . The Hessian matrix describes the local curvature of a function using second-order partial derivatives. Blob-like features detection is based on the eigenvalues of the Hessian matrix: if eigenvalues in X have the same sign, that point is an extrema.

While in SIFT the Hessian matrix is used to discard previously detected key points that lies on an edge, in SURF the matrix is used to detect the key points. Additionally, while SIFT uses the Difference of Gaussian to approximate the computation of the Laplacian of Gaussian, SURF pushes the approximation further by using the box filters shown in Figure 3.2. Box filters have a rectangular shape and the convolution of the image with a box filter can then be efficiently calculated using the integral image formula 3.2.

A significant advantage in using the box filters with the integral image is that calculating the convolution has the same cost regardless of the size of the boxes. SURF exploits this fact to search the

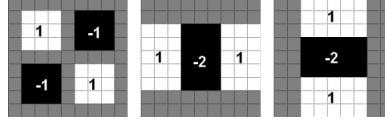


Figure 3.2: Box filters approximating the Laplacian of Gaussian for the xy , x and y direction respectively

size-space only by resizing the boxes instead of applying convolution and subsampling the image at different scale as in SIFT.

A last important feature of the Hessian matrix is the sign of the trace. This value is the sign of the Laplacian for the underling point and distinguishes bright blobs on dark background from dark blobs on dark background. It is a inexpensive property to compute and can be used for a faster comparison between features.

Successively, a threshold is applied to the determinant of the Hessian matrix so that low-contrast points are discarded. The value of the threshold here affects the number of detected interest points and will be used in our implementation to control the accuracy and speed of the matching (see 4).

To localise the interest point in the image, a non-maxima suppression is now performed by comparing each pixel to the 8 neighbours in image-space and 18 in scale-space (9 per adjacent scale). Once local extreme have been found, their position is refined to sub-pixel accuracy by interpolation using the Taylor expansion of the determinant of the Hessian, in the same fashion as with the Difference of Gaussian in SIFT.

At this point the SURF algorithm has detected the interest points at a given scale with sub-pixel accuracy and unstable points have been discarded. Integral images are then used again to speed up the creation of the descriptors.

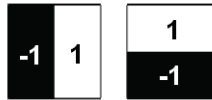


Figure 3.3: Haar wavelets used to compute the response in the x and y direction respectively

First an orientation is assigned to the feature by calculating the Haar wavelets responses in a circular neighbourhood of the point. The Haar wavelets have a rectangular shape (see Figure 3.3) and can

therefore be computed using the integral image. We won't cover the orientation assignment in detail as we are not using this information in our project. It's worth noting that there is a rotation-variant version of SURF, U-SURF, in which the orientation of the features is not computed.

Finally the feature descriptor is built using Haar wavelets again. A square window proportionate to the scale of the feature is built around the interest point and oriented along its orientation. This region is then split in 16 cells using a 4x4 grid and the wavelets response is computed in each cell for 25 points on a 5x5 grid. For each cell four values are computed. Given d_x as the responses of the wavelets on the x-direction (according to the point orientation) and d_y as those on the y-direction, each cell is associated to a 4-dimensional vector defined as:

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|) \quad (3.4)$$

Concatenating the vectors for all the cells will result in the final 64-dimensional descriptor of the feature.

3.3 Nearest neighbour search

Once visual features have been extracted from an image, they are matched against a set of features extracted from other images. All the feature descriptors covered in the previous section (see 2.1) contain a vector of real numbers. The simplest way to compare two features is then to compute the Euclidean distance (or the squared Euclidean distance) between these vector in a n -dimensional space. This computation is obviously slower if the dimension is higher, so descriptors with smaller vector (like the 64-dimensional SURF) are preferable over larger ones (like the 128-dimensional SIFT).

To compare a feature extracted from a test image with a features database, the closest feature in the database has to be found. This search can naively be performed in an exhaustive fashion, comparing the examined feature with all the features and selecting the one with the minimum distance. However, more efficient approximated approaches have been proposed.

The original SIFT paper uses an approximated nearest neighbour search called Best Bin First [3] designed by the same author. It is based on k -d trees data structure but adopt a different search

strategy. A k -d tree is created from recursively splitting the multidimensional data with a cut on the median point of the dimension that exhibits the greatest variance. The result of this recursion is a binary tree that shows higher resolution in regions where is needed. The Best Bin First search performs an approximate search by visiting closest bin first and imposing limiting the amount of leaf nodes visited, after which the closest neighbour found so far is returned.

The Best Bin First is reported to perform well with high-dimensional data like SIFT descriptors [23] both in accuracy and speed, but the performances degrade as the feature database grows in size. It would have probably performed better on SURF descriptors, being them half the size of a SIFT descriptor, however in our project we decided to use an exhaustive search approach for the simplicity in the implementation.

3.4 Genetic algorithm

A genetic algorithm is a search algorithms based on the mechanics of natural selection and genetics [11]. It mimics the process of natural evolution in searching for solutions to optimization and search problems.

The idea behind genetic algorithm is the Darwinian “*survival of the fittest*” concept. A population of individuals, each described by a series of *gene*, is considered. The population is able to generate some offspring in a sexual (two or more parents) or asexual (a single parent) fashion. The offspring will inherit a combination of the parents genes, with a chance of a random mutation occurring.

In this population, only the “fittest” are able to survive and reproduce generation after generation. Over time, this process will increase the “quality” or “fitness” of the population. It is evident from these statements how important it is to evaluate the fitness of an individual. In a simulated environment the fitness of an individual is determined by the genes that describe that individual. A *fitness function* is defined as

$$f : \mathbb{G} \rightarrow \mathbb{R} \tag{3.5}$$

where \mathbb{G} is the set of all the possible genes combinations. Given a combination of genes, the fitness function will return a real number (its *fitness value*) and is therefore possible to compare the fitness

between individuals.

While the fitness function has to contain a knowledge of the meaning of the genes to return the fitness values, the genetic algorithm itself is *blind* in the sense that it has no knowledge of the domain and why a certain combination of genes has a given fitness value. The search for an optimal solution doesn't come from exploiting the knowledge of the domain but from an evaluation of some random choices. However, even if a random component is present in the reproduction, a genetic algorithm is more than a simple random walk search. Genetic algorithms use random choices as a tool to explore the various regions of the search space and will keep exploring only the ones that return an improvement in the fitness value. The advantage of using a genetic algorithm is clearly in those situations where the search space is too complex to be described analytically or explored exhaustively.

A genetic algorithm might run for an indefinite period of time without finding the optimal solution and often we can't even tell if a given found solution is the optimal. For this reason some stopping criteria has to be defined so that the algorithm will not run forever, as even if we have an upper limit for the fitness function, we might not know if that value is achievable or even if we do, we don't know if there is a combination of genes that results in that value. Therefore some limit is set on the algorithm, such as the number of generations to be considered or the number of consecutive generations without a marked improvement that will be tolerated.

Wrapping it all together, a genetic algorithm is characterised by:

- a set of possible gene values
- a fitness function
- a population of individuals
- a reproduction strategy
- a stopping criteria

As we already mentioned, genetic algorithms are useful in exploring large search space with a complex domain that is not easily analysable. This type of algorithm can then be useful in some vision application and some attempts in this direction have been made in [31].

3.5 Clustering

Clustering is a method of unsupervised learning that attempts to find a structure in a collection of data or population. Conceptually a cluster is a collection of objects that are “similar” in some sense and clustering is the action of categorising the population into clusters. In order to have a concept of similarity, a distance measure has to be defined; a common measure is the Euclidean distance. Considering the distance measure, a cluster can be then defined as a collection of object that are close in space.

Several clustering algorithm are available. We will briefly cover the ones that have been considered for this project.

3.5.1 K-Mean

One of the simplest and most used clustering algorithm is *K-means* [24]. Given a desired number of clusters, this algorithm creates a partition of the population into k clusters where each element in the data belongs to the cluster with the closest centre (mean of its elements).

K-means starts by picking k random points from the population as clusters centres. All the points in the population are then assigned to the cluster with the closest centre. With this new assignment, the centre of all the clusters is then computed again. The process of assignment and computation of the centre is then repeated until the assignments don’t change further.

K-means is an iterative algorithm that will converge to a solution in time $\mathcal{O}(n^k d)$ where d is the dimension of the space [1]. However, since a random step is involved, repeated executions of the algorithm with the same input can results in different partitions. If the initial random choice is unfortunate, the resulting clustering can be ill-conditioned. Several variations of *K-Means* have been proposed. Among those, we examined *K-means++* and *ISODATA*.

K-Means++ [1] improves the initial random choice that can sometimes result in a poor clustering result. Effectively *K-Means++* is an algorithm to select the k initial clusters during the first step of *K-means*. Instead of using an unweighted random selection, just one initial cluster is selected. Then the remaining $k-1$ clusters are selected randomly one at time with a weighted probability proportional

to the distance from the already chosen centres. This ensures that the initial clusters are scattered around the whole population. This approach has proven to match or improve the performances of *K-mean* both on convergence speed and clustering errors.

ISODATA [34] is a more sophisticated algorithm that doesn't require to know the exact number of clusters. Instead, it takes as input a desired number of cluster that will be adjusted during the algorithm execution to match some constraints specified as input: the minimum number of samples per cluster, a threshold for standard deviation inside a cluster and a threshold for pairwise distances between clusters. Basically this algorithm follows every single iteration of *K-mean* by a series of checks that will adjust the clusters by merging or splitting them so that they will respect the specified constraints. Since these constraints can be conflicting the computation can be endless, so a maximum number of iteration is specified. *ISODATA* allows to have some control over the properties of the generated clusters, at the cost of having to (often empirically) choose more initial parameters.

3.5.2 Quality Threshold

A limitation of the clustering algorithms mentioned so far is requiring the number of clusters (or an approximation of it) as an input. This value is not easy to estimate and a bad choice can result in poor results. The *Quality Threshold (QT)* algorithm [14] uses a different approach so that the number of clusters is not to be given as input. An further difference compared to *K-Mean*-type algorithm is in the absence of any random component; *QT* is a deterministic algorithm generating the same clusters if repeated with the same input.

This algorithm is based on a maximum distance threshold, that is the only input parameter required. *QT* starts by considering each point as a candidate cluster and add other points to the clusters in order of distance until the threshold is reached. At this point the candidate cluster with the highest number of points is selected as a cluster. All the points in the cluster are removed from further consideration and the algorithm is applied again on the remaining set of point.

QT is a recursive algorithm that requires more computational time than *K-means* as it consider a high number of candidate clusters. However there is an advantage in the quality of the clusters for population where we have isolated points. *K-means*-style algorithms will force isolated points to be

included in a possibly unrelated cluster if k is not high enough, while QT will create as many clusters as needed to keep isolated points in separated isolated clusters.

Chapter 4

Design

4.1 Overview

The aim of our project is the creation of a self-contained building recognition application that runs on a mobile device with no need of a data connection to a remote server. In order to obtain this, a small visual features database has to be created and packed in the application so that it can be transferred to the mobile device only once during installation. We are particularly interested in the process of extracting a visual features database from a training set of images and we explored and compared various approaches.

In order to reach this goal, the project can be divided in three main logical steps

1. Evaluate various database creation approaches and select the best one according to the result of tests performed using that database
2. Create a database with the best approach we selected in step 1, using images of buildings collected with the same mobile device where the final application will be executed
3. Create the mobile prototype and pack the database created in step 2 in the application, so that can be installed on the mobile device

The design of our project actually differs from this breakdown as some steps have been merged. We

performed the evaluation of database creation approaches (step 1) and the creation of the actual database (step 2) at the same time. Since creating and testing a database is part of evaluating the different database creation approaches, this two steps have been joined. The database creation approaches have been evaluated by using the building images so that the resulting databases could be used immediately on the application.

The framework we designed is divided in two components. A set of application for the PC has been created to evaluate the different database creation approaches and to prepare the database that will be transferred on the mobile device (step 1 and 2). Then, a mobile prototype has been created such that, in conjunction with that database, it can recognise building and tell the user what the camera is pointing to (step 3). This prototype has been developed for the Android platform (see 3.1).

4.2 Location data

An ideal application would make use of the GPS data to filter the candidate buildings for matching. During database creation, a subdivision is defined so that the area of interest (city, district, etc.) is split in cells. The cells should be small so that the number of building present in a cell is low; however the size of the cell should also consider GPS inaccuracy, especially in narrow streets with tall buildings, a common scenario for this type of application. A possible approach is to use an irregular grid with a resolution adapting to the GPS precision in the given area.

This application can use the current location, acquired via the GPS, to select only the features that are expected in that area. This reduces the number of the features that have to be matched and therefore improves the speed of the matching step.

While part of an ideal approach, the GPS filtering has not been implemented in our project, as we focussed on the novel database creation. Adding a GPS filtering is not interesting for our research purposes so it has been left out. Instead of using GPS filtering, we assumed that this filtering will narrow the number of candidate building for matching to 5, a value coherent with the results obtained in [32].

4.3 Shared functionality

Both the PC component and the mobile component need to perform visual features extraction and matching. The PC component needs these features to build a features database and then test it, while the mobile application needs them to identify the buildings the camera is pointing to. A common core component is created to provide this functionality to both application.

4.3.1 Feature descriptor

We decide to use SURF for the feature extraction (see 3.2). Our decision is based on the ground that SURF is invariant to scale and rotation, so that images of the same building taken from various distances and angles will still be matched; it is also robust skew and perspective effects. Furthermore, the descriptor is not based on colour but on the intensity of the gradient, so even if the image is affected by some chromatic alteration (e.g. caused by different weather condition), the image can still be matched. Other feature descriptors and extractors with these characteristics exists but SURF matches or outperforms them in performance and accuracy (see 2.1).

SURF will extract feature descriptors that contain

- The orientation of the feature
- The scale of the feature
- The position of the feature in the image
- The sign of the Laplacian
- The 64-dimensional descriptor vector

4.3.2 Feature matching

In matching a feature extracted from a test image with the database, a comparison between features is performed. Our approach consider only the sign of the Laplacian and the descriptor vector in evaluating the dissimilarity between two features; the others elements of the descriptors are ignored. While using the other elements of the descriptor allows for a more sophisticated matching phase that

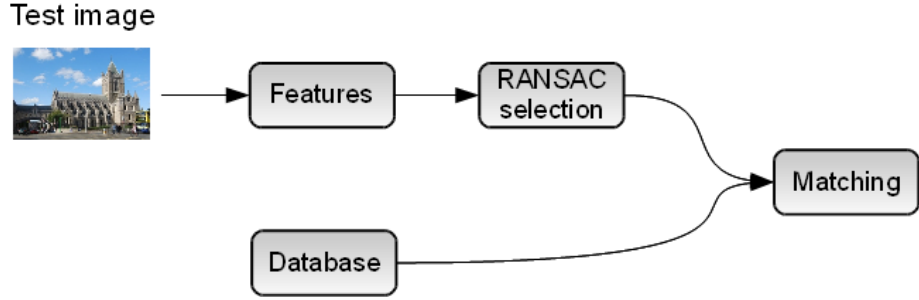


Figure 4.1: The matching step

can take into account the coherency of scale and orientation, as well as the geometrical disposition of the features in the image, we decided to use a simple matching for ease of implementation and to avoid additional computations that will have slow down the matching phase.

The matching is performed using votes. For each feature in the test image, the closest feature in the database is retrieved. The feature is associated with a building and a vote is added for that building. At the end of the process, the building with the highest number of votes is selected. We select the closest feature because two features from two different images of the same object, generated by the same physical feature, will most likely (if not surely) not have the same descriptor vector. Noise, deformations and occlusions will affect the two features such as they will be similar but not identical. However, the closest feature can not be selected unconditionally. There will always be a closest feature even if the feature that is being matched is a completely unrelated features. For this reason a threshold is set such that features more distant than the given threshold will not be matched. As suggested in [23], a valid global threshold adapting to different distributions in space might not exist. Instead, a threshold is set on the ratio between the closest feature and the second closest feature. If the ratio is close to 1, there is no single feature in the database that is clearly related to the one that is tested.

Ideally, the voting would be executed for all the feature extracted from the image. Since this can be time consuming, we use a RANSAC approach [10]. Only a number of random features is selected and matched. By choosing the features with an even probability the set of selected features is likely to be representative of the full set. The matching step is summarised in Figure 4.1.

4.4 Feature database

A good feature database is the key to have a matching application running fast and with a low error ratio. The database is created from one or more set of training images. In the ideal application, a set of images is picked for each GPS cell, while for our project two sets of pictures have been considered. These images depict a number of buildings that the database should be able to match and in order to obtain good results, each building should appear in more than one picture possibly taken in different lighting condition and at different times so that any occasional occlusion (like a bus) is not repeated. Of course, more pictures means more features in the database and hence a bigger database and a slower matching step.

4.4.1 Training sets

Two sets of buildings have been selected for this project. The buildings are scattered around the city centre and represent a variety of architectural styles and building periods; they are public or historical buildings that would fit in a tourist guide of Dublin's city centre. A list of the buildings used for our project is reported in tables 4.1 and 4.2.

Two sets of building have been defined to mimic the filtering that would be performed using the location data. The two sets, of 5 buildings each, are referred as the *EASY* and *CHALLENGING* set. Most of the pictures of buildings in the EASY set are clear, bright picture of the facade of the building; the CHALLENGING set on the other hand includes some buildings that are difficult to capture with a single picture either because of the shape or because there no physical spot in the location with a good view exclusive of the whole building.




Name	Trinity College Campanile	Christchurch Cathedral	General Post Office	National Maternity Hospital	St. Stephen's Green Shopping Centre
Sample image					

Table 4.1: EASY set of buildings






Name	Parliament House	Central Bank Building	Trinity College Dining Hall	Liberty Hall	St. Patrick's Cathedral
Sample image					

Table 4.2: CHALLENGING set of buildings

4.4.2 Clustering

Visual features are extracted from each training set and stored in a database. Our goal is to have a small database that is still able to return good matches; the size of the database is also linked to the time spent on the matching step as every feature in the database has to be considered during the matching step.

We considered three different approaches in the creation of the database, nicknamed *ALL FEATURES*, *CLUSTERING* and *TWO STAGES CLUSTERING*.

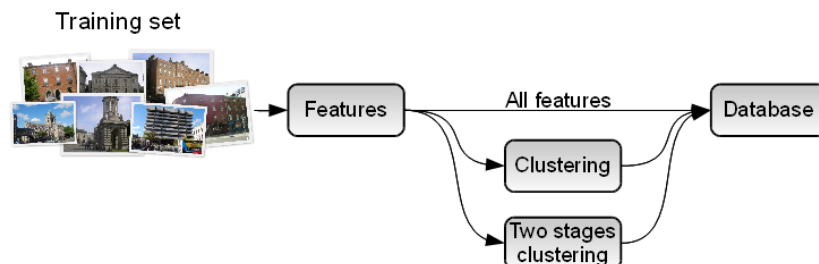


Figure 4.2: Database creation approaches

The ALL FEATURES approach

In the first approach (*ALL FEATURES*) all the features extracted from the training set are stored in the database. This means that the number of records in the database is equal to the number of features extracted; the more features, the bigger the database.

In this approach only a single image per building is used as training set. Since a first-to-second nearest neighbour ratio test is used, it was not possible to use more than one picture of the same building in the training set, as the same physical feature will result in two images would result in two very close

visual features that will then be discarded by the first-to-second nearest neighbour ratio test even if they are relevant features and their repetition is not an ambiguity.

The CLUSTERING approach

In the *CLUSTERING* approach the concept of *metafeatures* ([32]) has been used. In the set of features extracted from the training set there will be some recurring features generated by the same physical feature, or from identical physical features. These physical features can be unique for a given building, e.g. a particular window design, or can recur in various building, e.g. the street lights of a city. Grouping features by their similarity allows to identify such recurring features; this grouping is performed by clustering the features. Using a training set of three pictures per building, all the features are extracted from the training set and then clustered using a clustering algorithm. The result of this operation is a set of clusters each containing one or more features. The set of clusters is then refined to discard clusters that either include features from different buildings or that include features that occurs in too few images of the same building.

The first rule (multiple buildings) targets clusters that are not relevant in determining the building: features inside the clusters are not unique of a building and they won't be used for matching. The second rule (features from too few images) is enforced to eliminate features that will rarely be detected. While they can still be specific to a given building and therefore be useful for matching, in the quest for a small database precedence is given to features that are more likely to be detected used in the matching.

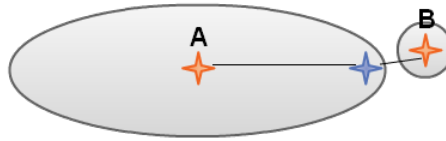


Figure 4.3: Wrong cluster assignment when considering only the cluster centre

Once the set of clusters has been filtered, only the centre of the cluster is stored in the database as a metafeature, discarding information about the single features in the cluster. This decision can result in feature matching errors as searching for the closer (meta) feature can now return a different cluster than the one the feature actually belongs to (see Figure 4.3). However by storing only the centre

the gain in space is evident and the cluster can be stored as if it was a regular feature, so the final application can work with features or metafeatures transparently.

The TWO STAGES CLUSTERING approach

Finally the last approach performs a two stage clustering. In the *TWO STAGES CLUSTERING* features are extracted from pictures of the same building and then clustered, as opposed to clustering all the features from all the building as in *CLUSTERING*. Once building specific clusters have been created, the set of cluster is refined by removing clusters with a small number of features. Finally, all clusters from all the buildings are compared and clusters from different buildings that are too close to each other are discarded as the contained features will not be useful in discerning different buildings. This approach has been designed in an effort to reduce clustering computation time, as clustering a smaller number of features (only features from the same building) reduces the clustering time sensibly. The additional refining step has to compare all the clusters, but hopefully the number of clusters is sensibly lower than the original number of features.

Clustering algorithms

Three clustering algorithms have been used in the project. These are *Kmeans++*, *ISODATA* and *QT* (see 3.5). These algorithms have different characteristics and we explored and compared their use in visual features clustering.

The refining step in the *TWO STAGES CLUSTERING* is performed using the *QT* algorithms.

4.5 Genetic algorithm

A lot of various parameters are involved in the techniques and methodologies described so far. For example, we have to decide a value for the SURF threshold (see 3.2), the number on RANSAC samples to use, and the parameters for the clustering algorithm we are using, such as the target standard deviation in the *ISODATA* algorithm.

The traditional approach in computer vision is to tweak these parameters by hand, by empirical tests or by exploiting the knowledge of the domain. Finding the right values is a search for a balance between various aspects, such as increasing the number of features in the database will increase the accuracy of the matches but reduce their speed.

However, when we are considering a set of different training images, as the case with our location cell-based approach, there might be trainings sets that perform better with values specialised for that specific set, while having global values used for all the sets doesn't allow to exploit the specific characteristic of the individual set. Let's consider the case of a training set (an area of a city) with a lot of buildings that look identical if not for small fine features. If we set a high threshold for SURF, such that only the strongest features are extracted, the weakest discriminant features will be ignored and discerning one building from the other will be an ill-conditioned problem. On the other hand, if we have a set with very different buildings that results in non similar strong features, we don't want to include a number of useless weak features that will just increase the size of the database when the match can be robustly performed using just the strongest features.

Another example of set specific values can be giving in regards to clustering. We can think of a building that has a number of very identical features, so that those features fill all fit in a cluster with a small variance or distance threshold but increasing the threshold will include some outliers. Another building, on the other hand, might have a number of similar features that are more spaced and the optimal threshold to include them in the same cluster would be higher.

Of course these values do not depends exclusively on the buildings that we want to match but also on the other visual features present in the same cell and that are not part of our training set. For example, we might have a large threshold for the size of the clusters to include all the similar features of a building in the same cluster. But this might cause the cluster to include outliers when considering non-positive images (e.g. if all the road signs in the area present a close feature). For this reason, a given set of values has to be validate by performing some test matches with both positive images (with buildings that should be matched) and negative images (without any building that should be matched) from the same cell, possibly covering a number of different views of that area.

Given the high number of parameters that can be tweaked and the fact that this has to be done for each cell, finding the optimal parameters by hand is not a practical approach. This project proposes

the use of a genetic algorithm (GA, see 3.4) to find good parameters automatically for each cell. At the end of its evaluation, the GA returns the best database it has found in the given number of iteration.

4.5.1 Fitness function

The GA is used to evaluate the various combination of parameters. A combination of parameters is good if the database created using that parameters is small in size and performs matches with a low error ratio and a high speed. However, since the matching step involves a comparison with all the features in the database (see 4.3.2), and the time spent for the comparison of two features is the same independently of the features involved, the speed of the matching step is directly proportional to the size of the database. So only one the matching time and the error ratio are considered, assuming the database size is directly proportional to the time.

The evaluation of a given set of parameters is divided in two steps. In the first step, the feature database is created using one of the three database creation approaches (the GA has been run once for each approach, *ALL FEATURES*, *CLUSTERING* and *TWO STEPS CLUSTERING*). In the second step, the database is used to match some positive and negative images; the error ratio and the total matching time are stored. The evaluation pipeline is shown in Figure 4.4.

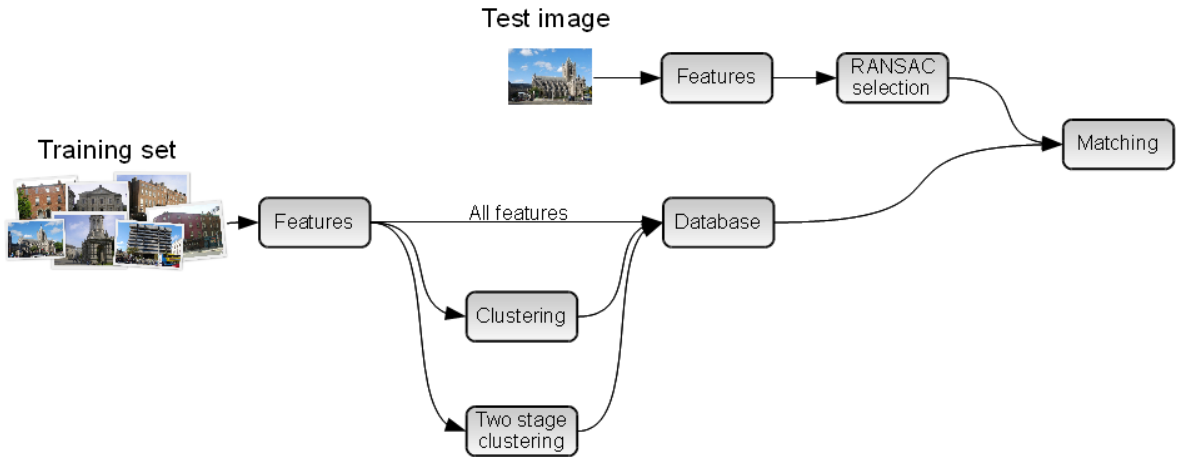


Figure 4.4: The fitness evaluation pipeline

A GA needs a fitness function that maps from a string of genes to a real value. Since the goodness of a set of parameters is based on two values (matching time and error ratio), a function that maps

these two values to a single value has to be designed.

We tried two different approaches in designing the fitness function. The first function we used is a paraboloid function that tries to reduce time and error ratio at the same time. Given an error ratio $e \in [0, 1]$ and a matching time $t > 0$ in milliseconds, the function *PARABOLOID* is defined as

$$\text{paraboloid}(e, t) = -a(1 + e)^2 - bt^2 \quad (4.1)$$

where a and b are constants defined to adjust the ratio between e and t as t is usually in the order of thousands. The *PARABOLOID* function is therefore a elliptic paraboloid with a maximum in $e = 0, t = 0$. However, a combination of parameters can exists such that even with an error ratio of 100%, the time can be close to zero and result in a better value for the fitness function compared to other combination with a slightly lower error ratio and bigger time. As an example, a combination with a very high SURF threshold will extract no features at all and the matching time will be in the order of ten milliseconds. To prevent this individuals from being selected, every individual with an error ratio above 95% is discarded by setting its fitness to $-\infty$ (or its machine equivalent).

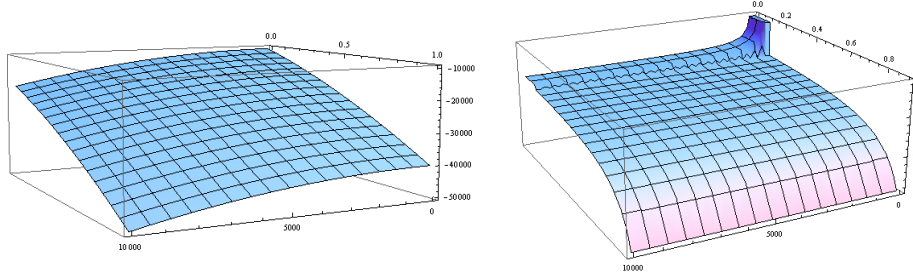


Figure 4.5: The *PARABOLOID* (left) and *PAIRWISE* (right) fitness functions

A second function has been defined as a pairwise function. The idea is to reduce the error ratio below an acceptable threshold and then minimise the time while remaining below the threshold. The *PAIRWISE* function is defined as

$$\text{pairwise}(e, t) = \begin{cases} \frac{1}{at} & \text{if } e < \epsilon \\ \frac{1}{\log(e)} & \text{otherwise} \end{cases} \quad (4.2)$$

where ϵ is the desired error ratio and a is a constant introduced to smooth the curvature of the function

for low t values. In our evaluations, we used $\epsilon = 0.1$

The plotted graphs of both function are shown in Figure 4.5.

4.5.2 Parameters tweaked

The genes considered by the GA are various parameters used in both the creation of the database and in the test matching step. Hence every individual in the population of the GA is a combination of parameters for the database creation and the test. The parameters that are considered in the creation of the database are:

- The SURF threshold for the extraction of features from the training set
- Whether to use SURF or U-SURF
- The clustering algorithm specific parameters in the CLUSTERING and TWO STAGE CLUSTERING approaches
- The minimum images per cluster in the CLUSTERING approach
- The minimum features per cluster in the TWO STAGE CLUSTERING approach
- The minimum distance threshold for clusters in the refining step of the TWO STAGE CLUSTERING approach

The parameters tweaked are summarised in Figure 4.6. On the choice between SURF and U-SURF, ideally an application using frontal pictures of buildings should expect to have the similar orientation for all the pictures and therefore the faster U-SURF can be used. However, we want to avoid forcing the user to stand exactly in front of the building as we want to allow for various different spots to take the pictures from. This can result in small rotation in the images and U-SURF is not expected to produce stable results if the rotation is more than a few degrees. On the other hand, U-SURF is faster as it skip the orientation assignment step. We decided to have the genetic algorithm selecting the best option.

Regarding the clustering algorithms, different parameters are evaluated according to the algorithm used. For the *Kmean++* algorithm, the only parameter is the number of clusters. Instead of providing

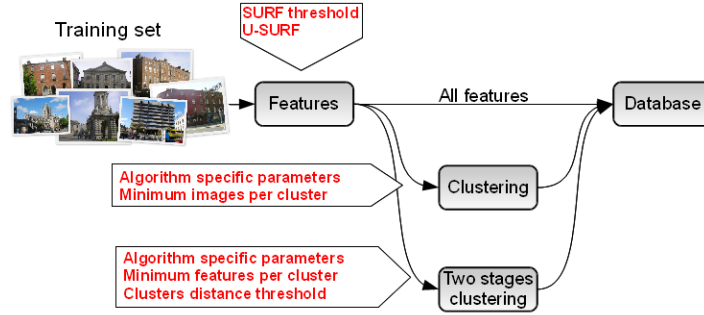


Figure 4.6: The parameters (red) tweaked during database creation

a hard value, a percentage in relation to the total number of feature is used.

For the *ISODATA* algorithm, the desired number of clusters is again a percentage, then other five parameters are used: the number of iterations, the standard variance threshold, the minimum number of elements per cluster, the maximum number of clusters to merge, and the pairwise distance threshold for merging.

Finally, the *QT* algorithm only requires a single parameters, the distance threshold.

Another set of genes is the used to perform the test matching step required by the fitness function to evaluate the database. The parameters in this case control how the features are extracted from the test images and how the match is performed. These parameters are:

- The SURF threshold for feature extraction
- Whether to use SURF or U-SURF
- The number of samples to use in RANSAC (expressed as a percentage)
- The first-to-second closest ratio

In total, the number of genes, including all the database creation approaches and the clustering algorithm, is 18. The complete set of parameters in the different steps is illustrated in Figure 4.7.

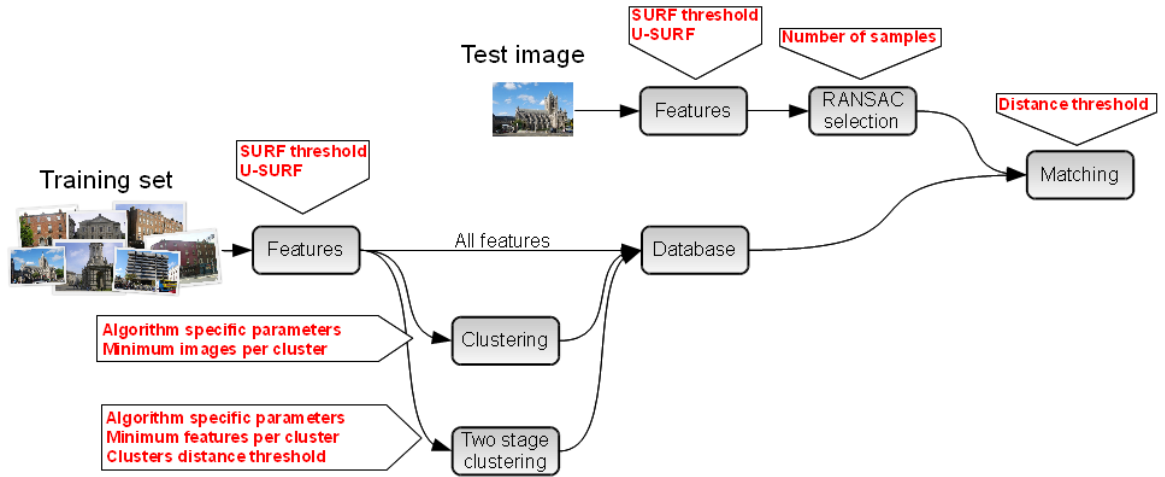


Figure 4.7: The complete set of parameters (red) tweaked by the genetic algorithm

4.6 Mobile prototype

The mobile component of the framework has been designed for the Android platform. Android is an attractive platform for a number of reasons: is open source, application can be developed without the need of a developer registration key and applications are written using a modified version of Java so that the code can (almost entirely) be shared between the Android and any other platform supporting Java.

Our Android application contains a database of features generated on the PC using the genetic algorithm discussed above (see 4.5). Images are captured from the camera and then processed to look for a matching building in the database. When a match is found, the user is informed of what the camera was pointing to.

4.6.1 Interface

The user interface for the mobile prototype implements a see-through approach where the user is presented the camera feed on the display. Camera frames are processed and when a building is recognised, a small icon of the building with the name of the building appears on one corner of the display. A complete application will probably need to provide more information, such as tourist information about the building, that can be visualised by tapping on the icon. Our prototype also

display a *status bar* on top of the display showing details about the matching step, such as the matching time and the number of features extracted.

Two modes of interaction with the user have been designed. The first one is a continuous mode, where frames are processed continuously as fast as possible. In this mode a frame is captured and processed without any user interaction; when the processing is terminated, the new current frame is acquired and processed, in an endless loop. The second mode is similar to the action of taking a picture as the user is requested to press a button to capture the current frame. A progress dialog is shown while the frame is processed (preventing any further capture) and then the result is displayed. The application will then be waiting for the next user's input to start the processing again.

4.6.2 Matching and database

As mentioned earlier, the features database is embedded in the application and hence stored in the device. In this prototype our database creation approach has been tested against the strictest space constraint, that is only the application package (with its 25Mb limit) has been considered as a storage option, as opposed to use an external memory card.

The feature database is stored in a relational database table. For each (meta) feature, the following information are available in the database:

- Descriptor vector (be it a real feature or the centre of a cluster in case of metafeatures)
- Sign of the Laplacian
- Building the (meta) feature belongs to

This structure allows to matching features as described earlier (see 4.3.2). In addition to the feature database, the prototype also require the values of the parameters for feature extraction from the current frame, the number of RANSAC samples and the first-to-second threshold. These parameters are the one computed by the GA in conjunction with the database that is being used. The matching step on the Android device is effectively the same matching step performed by the GA for the evaluation of the database (see 4.5).

4.7 Tests

For practical reasons, the mobile prototype has not been tested on the phone but tests have been completed on a PC with pictures acquired from the mobile device. This is an easier approach in terms of speed and repeatability since it avoids having to travel around the city to visit the various building. As the picture are taken with the mobile device, the results are still relevant.

A first test of the database validity has been performed by the GA itself (see 4.5). In evaluation a combination of parameters, the database is tested against positive and negative images. However, since these images are used to select the best database, they might be influencing the results in the sense that the parameters are adjusted to perform well only with those specific images. To further validate the database, once the best database has been selected by the GA, the database is tested with another distinct set of test images, both positive and negatives.

In the GA, each database is tested against 25 test pictures, 10 negative and 3 positive for each of the 5 buildings in the set. The additional test is performed with 10 negative pictures and 2 positive images for each of the 5 buildings in the set, for a total of 20 tests. The images from the additional test are not present in the GA test set or in the database training set.

In both in the GA and in the additional testing, the error ratio is defined as

$$E_{ratio} = \frac{Fp + Fn}{N} \quad (4.3)$$

where Fp is the number of false positives (images erroneously matched to the wrong building), Fn is the number of false negative (images that should be matched to a building but returned no match instead) and N is the number of images tested. The maximum value for the error ratio is obviously 1.0 while a perfect random assignment, considering five building classes as the negative match, should have an error ratio of 0.84.

Chapter 5

Implementation

In this chapter we will illustrate some details of the implementation of our framework that we omitted in the design section (see 4) and that we feel worth mentioning. We will not go into the details of the implementation but rather present an overview to give an idea of the challenges and issues faced during the implementation of the project.

5.1 Software and Hardware

As mentioned earlier, our framework is divided in two component, the database creation part, running on the PC, and the mobile application, running on an Android device. Both component have been mainly developed in Java. This choice has been made both because of the author's familiarity with this programming language and because Android applications have to be developed in its own version of java, so using Java on the PC component helped code reuse. The same result could also have been achieved using C/C++ code wrapped with JNI on the Android device but the different C/C++ compilers and headers would have make it more time consuming.

Three main Integrated Development Environments (IDE) have been used. To modify the C++ code of OpenSURF, Microsoft Visual Studio 2010 has been used. OpenSURF has been compiled using the Microsoft compiler and headers for the PC and the gcc compiler and Android NDK headers for

the Android. The database creation application and all the utility libraries coded in Java on the PC have been created using Sun NetBeans 6.9. For the Android application, Eclipse 3.6 from the Eclipse Foundation has been used, in conjunction with the Android ADT plug-in included in the Android SDK.

The PC we used in this project is a Dell Precision Workstation with an Intel Quad Core 2666Mhz CPU, 4 Gb of RAM and running Windows XP 32bit Service Pack 3.

During most of the duration of the project, no Android device was available, and the Android prototype has been developed on the emulator shipped with the Android SDK. An Android device was finally available at the last stage of the project and has been used only partially. The device is an HTC Desire phone running Android version 2.2 "Froyo".

5.2 Shared components

The shared component have been implemented as Java libraries that have been reused on both components. The shared components are the feature extraction and feature matching (see 4.3).

5.2.1 SURF

To implement the feature extraction, we used the OpenSURF library [7]. This is an open source implementation of the SURF algorithm that is also available for the Android platform. However, the OpenSURF Android source code relies on a custom made Android NDK that included the STL; at the time of writing that custom build of NDK was not available anymore. The C++ source code has then been modified by the author to remove the dependency on the missing libraries. The OpenSURF C++ is wrapped with JNI to be accessed from Java. This feature is already present in the original source code of the Android version and the wrapper has not been modified.

The genetic algorithm approach requires a high number of iterations. In each iteration, features are extracted from the training set, then the matching step is performed multiple times on images from the test set and again, features have to be extracted from the training images. In order to speed up the evaluation, a feature cache has been used. Instead of extracting the features from an image, with

a given SURF threshold, the features are extracted only once and the extracted features are saved into a cache. Every subsequent feature extraction for the same file, parameter and method, is performed by retrieving features from the cache.

The cache has been implemented in a MySQL database running on the PC in order to make it permanent rather than having a volatile cache reset at every new execution of the genetic algorithm. Since the SURF threshold parameter is used as a gene in the genetic algorithm, it is liable to a random mutation and can assume any real value between its bounds. The consequence of using a pure (pseudo) random threshold is that the likelihood of the same threshold being reused is close to zero. Instead of using a completely random threshold, 20 possible evenly-spaced values for the threshold are considered; when the value of the gene is randomised, a random value from the list is selected.

The information saved in the cache are the same that results from the extraction, i.e. the extracted features and the elapsed time for the extraction. This information is later used to compute the matching step time, given by the sum of the test image feature extraction and the actual feature matching time. The SURF cache is of course only available on the PC during database creation as the images tested are known beforehand. On the Android prototype, all the features are always extracted using the actual SURF algorithm.

5.2.2 Feature matching

In implementing the feature matching, the distance between the descriptor vector of the features has to be computed. The distance between two vectors p and q is evaluated using an Euclidean metric:

$$dist(p, q) = \sqrt{\sum_{i=1}^{64} (p_i - q_i)^2} \quad (5.1)$$

However, to avoid comparing all the n feature extracted with the m features in the database (with a complexity of nm), the sign of the Laplacian can be used for fast matching. As the sign of the Laplacian discerns dark blobs on light background from light blobs on dark background, there is no need to compare two features with a different sign as they will not be related to the same feature. If

we assume an uniform distribution of the sign of the Laplacian between the features, the cost of the comparison is reduced to $\frac{nm}{2}$.

5.3 Database creation

5.3.1 Clustering

The clustering algorithms considered (*Kmeans++*, *ISODATA* and *QT*) have been implemented by the author following the approach given on the original papers.

Clustering has proved to be a computation intensive operation. Distances between points (in our case, descriptors of features) are computed using an Euclidean distance on 64-dimensional vectors and all the clustering algorithms use this operation intensively. In order to speed up the clustering step, a distance matrix is computed as the first step of the clustering algorithm, such that distances are retrieved from the matrix rather than computed at every iteration. This solution works in conjunction with the *QT* algorithm, as the only distances computed are between points, but does not work with *Kmean++* and *ISODATA* as in this two last algorithms distances are computed between points and cluster centres, with clusters centres constantly changing and not know beforehand.

In the *TWO STAGE CLUSTERING* approach, the second step, in which clusters that are too close to each other are discarded, has been approximated using the *QT* algorithm on the centres of the clusters. The distance threshold under which clusters should be discarded is used as the *QT* distance threshold, then each cluster containing more than one centre is discarded.

5.3.2 Visual feedback

A number of various algorithms and methodologies have been implemented in this project. To have a feedback during the development and to ease the debug of the code, a visual interface has been created to manually perform all the operation that the genetic algorithm execute to evaluate a single individual (feature extraction, database creation, clustering, matching). We will not go into the details of this application as it has been used only as a development tool and is not part of the framework; a

screen shot of the interface is shown in Figure 5.1.

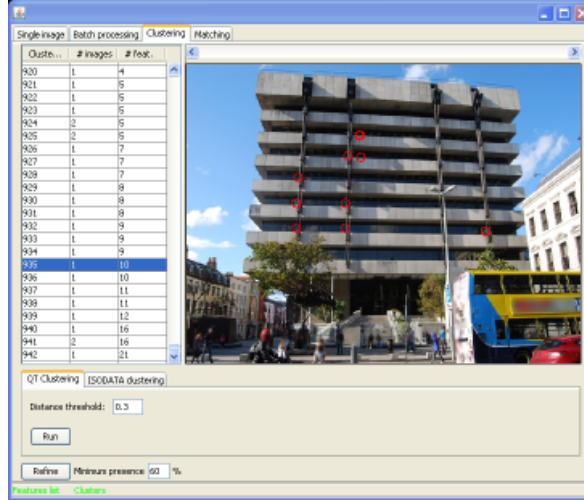


Figure 5.1: A screen shot of the visual interface used for testing and debugging

5.3.3 Genetic algorithm

The genetic algorithm code has been written by the author following the principles detailed in 3.4. Three type of genes have been implemented, according to the type of value they represent: integer, float and boolean. The individuals in the genetic algorithm are a list of 18 genes; the initial population, consisting of 25 individuals, is generated by assigning random values to the genes of each individual.

The population is evaluated using one of the two fitness functions defined in 4.5.1. Once the initial population has been evaluated, the best 7 individual are selected for reproduction. In order to prevent the algorithm from being stuck in a local maxima, an additional individual is randomly selected from the population, regardless of its fitness values.

This selection returns 8 individual that are then breded two by two, generating 4 new individual. These individual inherit the value of each gene from one of the parent, randomly chosen with an even probability. In an effort to avoid local maxima, there is a 20% mutation chance for each gene, i.e. a gene every five is assigned a total random value.

Some measures have been taken to have the genetic algorithm running in a reasonable time. The genetic algorithm has been implemented as a multi threaded application running the fitness evaluation

function on a pool of 4 thread. As the algorithm was run on a PC with a quad core CPU, its full computational power was exploited. The clustering step however is still an issue. With a low SURF threshold, an image can easily return more than one thousands features, resulting in a training set containing more than twenty thousands features, each being a 64-dimensional vector. This scenario reaches the memory or the stack limit (as QT is recursive) of the PC which is running a 32 bit operating system. Combination of parameters that reach this limit have therefore not been evaluated and the individual is assigned the lowest value possible for the fitness.

Furthermore, 500 or 1000 iterations of the genetic algorithm are computed, each evaluating 4 new individuals and if the clustering step takes more than a few minutes on the PC for a high number of individual, an execution of the algorithm could take days. In order to be able to run all the tests in the time frame of this project, clustering execution time has been capped at ten minutes and every individual requiring more than 10 minutes to be evaluated is detected by a timer, the evaluation aborted and the individual is discarded by assigning the lowest value possible for the fitness.

5.4 Android prototype

The Android prototype has been implemented by the author as a single self-contained application using the OpenSURF JNI wrapper. The images are captured through the camera at a resolution of 640x480 and processed by a separate thread.

5.4.1 Interface

The interface is implemented as a surface displaying the camera preview frames. On top of the interface, a status bar with debug information and a building label is shown. When a building is matched, a small icon of the building (pre-packed in the application) is shown in the bottom corner.

The user can switch between the two modes (see 4.6.1) via a user menu open with the menu button. Once the matching is completed, the phone also vibrates to notify the user.

The user interface, running in the emulator with a test image, is shown in Figure 5.2.



Figure 5.2: Screen shots of the Android prototype on the emulator, before (left) and after (right) matching

5.4.2 Database and matching

The feature database is stored in the SQLite relational database available to every Android application. In addition to the feature database, the SQLite database also contains the parameters that will be used for matching (SURF threshold for the captured image, number of RANSAC samples, etc.).

The feature database is transferred from the PC to the device via a text file containing the SQL commands to populate the relational database. The file is then packed as part of the assets of the device. When the application is started on the device, the version of the relational database is checked. If a new database is available, the old tables are dropped and the database is re-populated with the commands from the text file. A more efficient approach (in terms of file size) would be to create the SQLite database file on the PC then pack the SQLite database file within the application. However this approach is more complicate to code as the default Android-managed relational database has to be overridden with the custom database file. In our prototype the simpler approach was implemented.

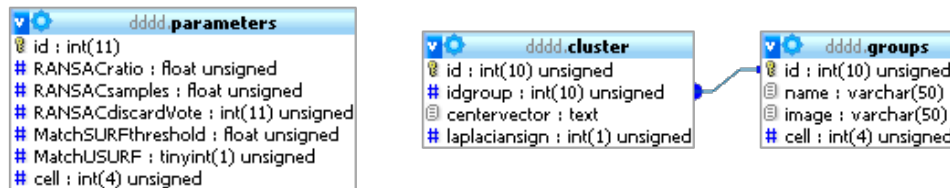


Figure 5.3: The relational database schema

The 64-dimensional descriptor vector of the features is stored in the relational database in a single text field. Storing the floats with their textual representation is not optimal, however the database creation file with the SQL commands, the one that matters for the application size, is effectively a text file, so a textual representation is used anyways. The field is then read and converted back to a

float vector when needed. A database diagram is shown in Figure 5.3.

5.5 Tests

5.5.1 Images acquisition

The images used for the training sets and for the test sets have been acquired manually by the author around the city centre of Dublin, then manually reviewed and edited by hand to remove any face and registration plate in order to avoid privacy issues.

Ideally the images should have been captured with the mobile device itself, so that they would have the same characteristics of the camera frames captured by the mobile application. However the mobile device was not available until the later stage of the project so the images have been captured using two digital cameras, a compact camera (Pentax Optio 30) and a reflex (Nikon D40).

Later in the project we had the chance to use an actual mobile device (HTC Desire) to capture the images. However as it was too late in the progress of the project to rerun all the test, only a few selected tests have been redone with the mobile captured images.

The main differences between the mobile captured images and the digital camera captured images are in the focus, exposition and white balance. Mobile acquired images are generally more dark and blurry than digital camera acquired ones.

Chapter 6

Results

We will now present the results of the various evaluation and comment on them. While more tests were planned, the long running time of some approaches - spanning days - and the fact that mobile-acquired images were not available until the very end of the project did not allow for a complete test of all the possible combinations of building sets, clustering algorithms and clustering approaches within the project time frame.

A broader range of tests has been run for the digital camera-acquired pictures, while only selected evaluations with the mobile-acquired pictures have been completed - namely the ones covering the EASY set.

In presenting the results, two different sets of values are shown. The first set, which we will be referring as *genetic algorithm evaluation*, are the values returned by the fitness function for the best database generated by the genetic algorithm, while the *additional test results* are the one obtained using a external test set on the best database generated by the genetic algorithm, as explained in 4.7.

6.1 Genetic algorithm

The first evaluation performed is a comparison of the two fitness function we designed (see 4.5.1). The aim of this comparison is both in validating the use of the genetic algorithm to find a good

combination of parameters and in evaluating the quality of the two fitness functions.

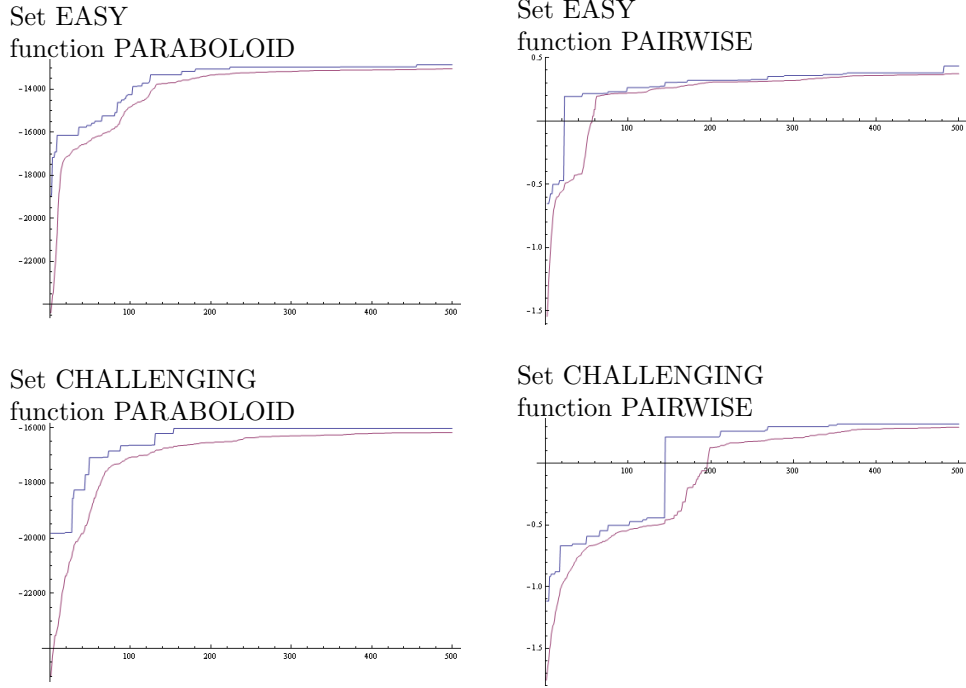


Figure 6.1: Convergence of the genetic algorithm with the *ALL FEATURES* approach, camera images

Four executions of the genetic algorithm have been evaluated. In the first, 500 iterations have been used using the simples database creation approach, *ALL FEATURES*, using the fitness function *PARABOLOID* on the EASY set acquired with the digital camera. In the second, the same setup has been repeated but for the use of the *PAIRWISE* fitness function. The third and the fourth evaluations have been run with the same setup, but on the CHALLENGING set acquired with the digital camera.

The *ALL FEATURES* approach has been chosen to validate the fitness functions as the approach itself is already know to be valid and used with many variants in literature (for example [23, 2, 5]), as opposed to the less explored *CLUSTERING* approach. All the setups converged to a solution, as shown in Figure 6.1 where the average (red) and best (blue) value in the population for the fitness functions is shown in relation to the number of iterations. What is interesting in this graphs are not the actual values but the shape of the curve. It's clear from the graphs that all setups converged to a valid solution, even if there is no guarantee that it is the best solution possible (more iterations might have revealed further improvement).

Set	Function	error ratio	matching time	GA run time
EASY	PARABOLOID	11.4%	602 ms	02:03 h
EASY	PAIRWISE	9.6%	731 ms	10:08 h
CHALLENGING	PARABOLOID	25.9%	347 ms	01:15 h
CHALLENGING	PAIRWISE	15.8%	750 ms	09:22 h

Table 6.1: Genetic algorithm evaluations for fitness function comparison, *ALL FEATURES* approach, camera images

In Table 6.1 we report the values of the function and the breakdown of error ratio and matching time that are used to compute the fitness function. The values in the table are relative to the best solution found and we also report the time it took to complete the 500 iterations. We can see that the *PAIRWISE* strategy, reducing error ratio before reducing time, results in a extremely long computation that is not guaranteed to produce marked improvements in the error ratio compared to the *PARABOLOID* approach. While on the EASY set the error ratios are comparable, on the *CHALLENGING* set the *PAIRWISE* function obtain a sensibly better error ratio but in a doubled matching time.

Considering these results it would be interesting to use both functions to test if the gap in error ratio and time will change in other circumstances. However due to the fact that the other approaches, *CLUSTERING* and *TWO STAGES CLUSTERING*, are expected to add heavily to the total running time, we decided for practical reasons to use only the *PARABOLOID* function in any following evaluation.

6.2 Database creation

6.2.1 ALL FEATURES approach

For the *ALL FEATURES* approach, a big database size is expected as all the features extracted from the training set will be included in the database. The genetic algorithm has been run both on the EASY and CHALLENGING set with digital camera acquired images for 1000 iterations, using the PARABOLOID function.

With camera acquired images, we obtain modest error ratios in the genetic algorithm evaluation for

Genetic algorithm evaluations				
Set	Err. ratio	Match time	Features in DB	GA running time
EASY	11.2%	566 ms	1419	04:06h
CHALLENGING	23.1%	475 ms	1101	02:34h

Additional tests results					
Test images	Set	Avg. error	Error std.	Avg. time	Time std.
Camera	EASY	18.5%	1.53%	424 ms	153.6 ms
Camera	CHALLENGING	34.4%	1.95%	436 ms	187.7 ms
Mobile	EASY	37.3%	1.33%	359 ms	142.7 ms

Table 6.2: GA and tests results for *ALL FEATURES* approach, camera images, 1000 iterations

the EASY set and worse results for the CHALLENGING set. When tested with other camera acquired images the databases revealed a higher error ratio but when the database created for the EASY set has tested with mobile images the error ratio increased more than three times. Results are shown in Table 6.2.

The the increase in the error ratio in the camera test can be blamed to our implementation of the matching step, were only voting is used and no consistency constraint are enforced. This makes the number of votes for a given building quite unstable and while the best solution found by the genetic algorithm performed well in the genetic algorithm evaluation, when considering images with different features even a small difference in the votes can make a sensible difference.

Considering the mobile acquired images, in addition to the aforementioned issues, the higher error ratio might also be caused by the different nature of the images acquired with the two different devices. The genetic algorithm has successively been re-run using mobile images to create the database and the test results, while still showing a considerable error ratio, improved significantly in the test phase, as shown in Table 6.3. We regard the mobile acquired pictures as being “harder” to deal with than the camera acquired EASY set and a similarity to the CHALLENGING set is highlighted by the error ratios.

An evident trait of the results is the high standard deviation in the matching time. This is due to the varying number of features extracted from the test images. Some test images naturally presents more features than others for the given SURF threshold and this makes the difference in extracting and comparing with the feature database as an exhaustive comparison is performed. The computational cost of the matching with the database is $\mathcal{O}(nm)$ where n is the number of features extracted and

then selected by RANSAC (where the RANSAC samples percentage is the same for all the images in the test) and m is the number of features in the database, constant for the test. Hence extracting double the features from an image would double matching time.

Genetic algorithm evaluation					
Set	Err. ratio	Match time	Features in DB	GA running time	
EASY	12.5%	370 ms	702	02:23h	

Additional test results					
Images	Set	Avg. error	Error std.	Avg. time	Time std.
Mobile	EASY	23.3%	2.1%	337 ms	132.4 ms

Table 6.3: GA and tests results for *ALL FEATURES* approach, mobile images, 500 iterations

Concluding, this approach is successful in matching some images but it can result in a relevant error ratio. This result was expected. On one hand, this approach is widely used and therefore at least some sort of positive matching was expected. On the other hand, some simplifications we implemented in the matching step, namely not checking any orientation, scale or geometric coherency, caused numerous errors. This evaluation however is useful in setting a reference that can be used to compare the other two approaches with.

6.2.2 CLUSTERING approach

The three clustering algorithm, *QT*, *Kmeans++* and *ISODATA*, have been evaluated for the *CLUSTERING* approach. As expected, the running time of the genetic algorithm drastically increased for this approach as each evaluation of the individual require a clustering operation that can span for minutes. Considering that in 500 iterations of the genetic algorithm 2025 individuals are evaluated, running times of one day were not unexpected.

We were expecting that the *CLUSTERING* strategy would have show a sensibly smaller database than the *ALL FEATURES* one with a lower error ratio, however the results obtained with the various algorithms display different characteristics. Results are illustrated in Table 6.4 and discussed in the following paragraphs.

Genetic algorithm evaluations

Clustering	Set	Err. ratio	Match time	Features in DB	GA running time
QT	EASY	08.0%	453 ms	878	03:10h
ISODATA	EASY	08.8%	298 ms	760	11:46h
KMEANS++	EASY	18.0%	248 ms	93	20:40h
QT	CHALLENGING	60.0%	156 ms	6	01:42h
ISODATA	CHALLENGING	60.0%	195 ms	10	07:37h
KMEANS++	CHALLENGING	60.0%	123 ms	8	11:34h

Additional tests results

Test images	Clustering	Set	Avg. error	Error std.	Avg. time	Time std.
Camera	QT	EASY	16.1%	4.48%	418 ms	142.9 ms
Camera	ISODATA	EASY	13.0%	1.88%	275 ms	79.0 ms
Camera	KMEANS++	EASY	27.7%	2.87%	235 ms	58.1 ms
Camera	QT	CHALLENGING	61.6%	0.00%	155 ms	17.0 ms
Camera	ISODATA	CHALLENGING	61.6%	0.00%	170 ms	25.5 ms
Camera	KMEANS++	CHALLENGING	61.6%	0.00%	156 ms	16.0 ms
Mobile	QT	EASY	28.0%	1.63%	334 ms	124.1 ms
Mobile	ISODATA	EASY	29.3%	2.49%	232 ms	68.7 ms
Mobile	KMEANS++	EASY	42.0%	3.39%	203 ms	51.5 ms

Table 6.4: GA and tests results for *CLUSTERING* approach, camera images, 500 iterations

The CHALLENGING set

Considering the CHALLENGING set, all algorithms failed to converge to a valid solution. The best database returned is in fact a database that marks all images as negative - hence the 60% error ratio, because 40% of the images in the GA test set are negative (this is 61.6% for the additional test set). The database returned is practically empty and the matching time is very low as there are just a few features to compare. Various reasons for this behaviour can be hypothesised.

First of all, we know there is at least a better solution - a combination of parameters that will create a cluster for each feature and discard no cluster, so that the resulting database will be identical to a database generated by the *ALL FEATURES* approach and this, for at least some combination of SURF threshold and RANSAC parameters, returns a better solution as seen in the previous *ALL FEATURES* tests.

The fact that this solution was not found can mean that the search space is too wide and the genetic algorithm, with the current setup of number of iterations and population size, was not able to explore it adequately. If this is the case, re-running the evaluation with a more generous setup would result at least in a (probably slow) convergence to that solution.

This result can also mean that the *PARABOLOID* function is not adequate. The matching time is extremely low compared to the other results, so probably in presence of an higher error ratio that can not be reduced under a certain value (as the *CHALLENGING* set was expected to have) the function pushes for a lower time over a lower error ratio. If this is the case, adjusting the a and b constant for the function (see 4.5.1) should prevent this behaviour. However, a reading of the logs of the genetic algorithm reveals that no lower value for the error ratio has ever been generated over the 500 iterations, so this reason is to be excluded.

Finally, we can hypothesise that the time and memory limit of the evaluation discarded combinations of parameters that would have otherwise contributed to a better solution. This explanation is supported by the fact that, in order to obtain a solution similar to the *ALL FEATURES* approach, a cluster has to be created for each feature. For the *Kmeans++* and *ISODATA* algorithm, having almost as many clusters as features is the slowest configuration possible to process. For the *QT* algorithm, having a cluster per feature means that at each iteration only a single point is removed from the set so a new call of the recursive algorithm is invoked per feature, possibly hitting the maximum recursion limit or the maximum stack size. According to some further investigation on this thesis, about 5% of the iterations in the non converging genetic algorithm evaluations result in a memory or time limit being hit.

The reason why having almost one cluster per feature is the best solution in this case is to be searched in the uniqueness of the features in this set - most likely there are few or no easily clusterable similar features in the set. We probably selected images that are too different from each other to produce similar features if not under a very few combinations of parameters.

Summarising, a combination of a restricted genetic algorithm setup and limited resources might be causing a convergence to an invalid solution. If run without limitation, we expect to obtain a solution that is similar to the *ALL FEATURES* approach, with probably only a minimal improvement on the database size, if any, due to those few features that were clustered.

The EASY set

The EASY set shows completely different results from the CHALLENGING set. Here a convergence to a good solution is reached but the improvement in relation to the *ALL FEATURES* approach varies with the algorithm used. In discussing this comparison however it should be considered that due to an longer running time, the *CLUSTERING* approach was evaluated with 500 iterations of the genetic algorithm as opposed to the 1000 of the *ALL FEATURES*.

Each algorithm converged to a solution that shows some improvements over the *ALL FEATURES* approach. The size of the database is extremely reduced in *Kmeans++* and this affects both negatively the error ratio and positively the matching time. The *ISODATA* and *QT* databases have a better error if compared to the *ALL FEATURES* even if the improvement is not drastic, while it is more marked on the databases size. This suggests that the reduction in the databases size resulted from eliminating (meta) features that were not crucial for an accurate matching and this is the validating our original idea.

The reason for better results obtained with the *ISODATA* rather than *Kmean++* probably lies on the quality of the clusters, since the first algorithm guarantee a limited variance in the clusters. The *QT* algorithm on the other hand uses a complete different approach in clustering but produces similar values to *ISODATA*.

Considering running time, *Kmean++* display the longest running time. While this is not a main concern for this project as the database creation step will be performed offline, further inspection in this issue revealed that, not surprisingly, the additional time compared to *ISODATA* is spent on the initial clusters selection (see 3.5.1) and that in the initial 5 iterations the time limit was hit for 15% of the individuals. This might have conditioned the algorithm as we don't know if those discarded combinations of values could have brought an improvement.

On the other extreme, *QT* algorithm is the fastest to evaluate, as expected from being the only that take advantage of a distance matrix (see 4.4.2).

The following external tests revealed again a poor performance with mobile images. The same evaluations - but only on the EASY set - have been re-run with a training set using mobile images. In re-running the evaluations with the more challenging mobile acquired images, an additional positive

image per building has been added to the training set.

Genetic algorithm evaluations						
Clustering	Set	Err. ratio	Match time	Features in DB	GA running time	
QT	EASY	18.3%	274 ms	139	07:48h	
ISODATA	EASY	18.8%	329 ms	173	55:39h	
KMEANS++	EASY	20.1%	155 ms	92	89:58h	

Additional tests results						
Test images	Clustering	Set	Avg. error	Error std.	Avg. time	Time std.
Mobile	QT	EASY	30.6%	1.33%	239 ms	76.1 ms
Mobile	ISODATA	EASY	22.6%	1.33%	281 ms	93.0 ms
Mobile	KMEANS++	EASY	32.6%	3.88%	214 ms	57.7 ms

Table 6.5: GA and tests results for *CLUSTERING* approach, mobile images, 500 iterations

The first evident result is the drastic increase of the running time. Adding one image per building, for a total of 5 images, pushed the running time from some hours to more than 3 days. However, the *QT* is still definitely the faster thanks to the distance matrix and has comparable error ratio and time values.

Results obtained with the *ALL FEATURES* approach with mobile images showed an high error ratio. The *CLUSTERING* strategy did not affect the error ratio sensibly for mobile images but improved the matching time and reduced the size of the database. While the error ratio still remains higher if compared to camera images, in both camera and mobile acquired images the *CLUSTERING* approach proved to be valid - if computable on the set - obtaining a drastic reduction in the database size. However, with a lower number of (meta) features in the database the error ratio tends to increase. Considering the wide improvement in the database size, probably our fitness function rewarded excessively the size improvement over the precision. Tweaking the fitness function should result in a more balanced solution.

6.2.3 TWO STAGES CLUSTERING approach

The last approach evaluated is the *TWO STAGES CLUSTERING*. This approach failed to return any valid solution at all with all the considered clustering algorithm and sets. The databases generated are the same seen in the *CLUSTERING* approach with the *CHALLENGING* set, however in this case the even the *EASY* set failed to converge to a valid solution as shown in Table 6.6. In the table, only

the results obtained with *QT* clustering are shown. The *Kmeans++* and *ISODATA* have identical results - they failed to converge to a valid solution.

The same considerations discussed for the CHALLENGING set in the *CLUSTERING* section apply. Moreover, since the failure was evident even with the EASY set, we can speculate that this approach is too restrictive in eliminating clusters.

Probably clusters created in the first step of this approach for a given building (see 4.4.2) are very likely to overlap with some other clusters created for other buildings; this might prove a fault in the strategy of considering each building independently. We were expecting to have at least some combination with a very low *QT* threshold for the second step so that some clusters would have survived the elimination; in this case, the surviving cluster centres probably do not reflect the shape of the cluster and the first-to-second ratio in matching features might discard them. If this is the case, the fitness function would select the configurations with no clusters at all because they are faster to match. Even if we are not able to formulate a definite assessment on the possible convergence of this approach without a deeper inspection of the behaviour and an unlimited genetic algorithm evaluation of this approach, we can still speculate that this approach doesn't present any advantage over the other two.

Genetic algorithm evaluations					
Clustering	Set	Err. ratio	Match time	Features in DB	GA running time
QT	EASY	60.0%	156 ms	71	02:54h
QT	CHALLENGING	60.0%	155 ms	71	01:53h

Table 6.6: GA results for *TWO STAGES CLUSTERING* with camera images

6.3 Matching time and database size on the Android device

Tests results with mobile acquired images have been computed on the PC for practical reasons - doing that on the Android device would have requested to repack and reinstall the application for each test. Using mobile acquired images on the PC produces realistic results as the Java class library used to perform the match is shared between the two implementations.

However the PC can run the tests in a fraction of the time it would take the Android device. For

this reason, while the values shown in the previous sections are useful in comparing the various approaches, the times reported don not reflect the expected matching times on the mobile device. In order to properly evaluate the times, selected tests have been repeated on the mobile device and the matching times have been recorded. Table 6.7 summarises the values obtained.

It is evident from the data that the bottleneck in the evaluation on the Android is in the SURF feature extraction. The matching step (RANSAC and voting) only accounts for a minimal contribution to the total time even if no particular efficient implementation has been used in the code. The matching time for a single iteration is also only loosely related to the number of features in the database as it is only affected by the higher value of the *ALL FEATURES* approach.

Test	PC avg. time	# Feat.	Mobile avg. time		
			Total	SURF	match
<i>ALL FEATURES</i> , EASY	370 ms	702	2625.4 ms	2606.6 ms	18.8 ms
<i>CLUSTERING, QT</i> ,EASY	426 ms	139	2574.2 ms	2563.5 ms	9.7 ms
<i>CLUSTERING, Kmeans++</i> ,EASY	155 ms	92	2294.8 ms	2285.8 ms	8.0 ms
<i>CLUSTERING, ISODATA</i> , EASY	329 ms	173	3142.8 ms	3138.4 ms	4.4 ms

Table 6.7: Matching times on the Android device

When presenting the results, the database size has been expressed in number of features. To have an idea of what are the consequences of these values on the Android application, the number of features can be converted in occupied disk space using our naive SQL file approach. Table 6.8 gives an idea of the capacity of the Android device in relation to the size of the database.

6.4 Comparison with previous works

Comparisons with related works are not easy due to various factor. The main contribution of our project is in having a compact local database on a mobile device. Some previous works in literature perform the match on the PC, with an unspecified feature database size [21, 30, 37]. On the other hand, mobile approaches, including commercial product, mostly rely on a remote server with no or little information on the database size ([32, 13]) or have a complete different approach that is not easily compared ([28]). Additionally, while papers usually report precise descriptions of the tests performed, no such thing is available for commercial products. We decided to compare our results

Number of features	Kb	Max. cells on Android
100	82 Kb	256
150	120 Kb	175
200	164 Kb	128
1000	820 Kb	25
1400	1148 Kb	18

Table 6.8: Disk space occupied on the Android device

with two closely related, but remote server based, works: [32] and Google Goggles [13].

In [32] the number of features has been considered in order to reduce bandwidth and even if we can not compare the size of the database on the storage, we have an indication of how many features are transferred for each cell - reported to be limited to 10000. The matching time, that include network latency for data transfer, refers to a matching step computed on the mobile device and is therefore comparable to our results.

We also selected Google Goggles, in spite of the lack of testing data, to show a comparison with a leading market application. In the comparison user perceived values are considered so the matching time in this case will include the network latency and data transfer time. Google Goggles can match landmarks building, or any other type of image, without GPS data. This means that each images is matched to a huge database including thousands if not millions of images in an impressively short time; however, the delay perceived by the user is quite long.

	Error ratio	Match time	Features in database
Google Goggles	n/a	> 5000 ms	n/a
[32]	7%	2800 ms	10000
Our approach	22.6%	3142 ms	173

Table 6.9: Comparison with previous works

The results we included in the comparison, shown in Table 6.9, are the best results we obtained for the EASY set on the mobile phone. The comparison includes matching time as perceived by the user. Our approach outperforms Google Goggles in matching time, even if no easy comparison can be made about error ratio - which we expect to be lower on Google Goggles. When comparing with [32], we obtain a higher error ratio with a comparable matching time. However, the number of features used for the matching is sensibly smaller.

Our approach is successful in producing small databases, some ten (*ALL FEATURES*) to hundred

times *CLUSTERING* smaller than [32]. However error ratio is an open issue and this suggests that a better balance between errors and size has to be found. We did not had the chance to implemented a more refined matching step like the one seen in [32], so it is no clear how many matching errors are related to the database size and how our error ratio would get closer to values in [32] just by improving the matching phase maintaining the same database size.

Chapter 7

Conclusions

7.1 Considerations

We explored and tested various approaches for a compact visual feature database creation using genetic algorithms to be used for building recognition on a mobile platform.

Our feature database creation strategy has been validated by tests. While some combinations of clustering algorithms and clustering approaches that we experimented did not result in a valid feature database, others have proven to be valid and resulting in a compact database. In any case, a genetic algorithm proven to be a good tool to automate the creation of a number of such databases, in a unsupervised learning fashion, even if enough iterations should be evaluated should be evaluated and the fitness function has to be tweaked accurately in order to converge to an useful solution.

In particular, the *ALL FEATURES* approach resulted in a sensible error ratio, a moderate size and a slow matching time, while the *TWO STEPS CLUSTERING* computation failed to converge to a useful solution. On the other hand, the *CLUSTERING* approach is able to return good results in database size with a comparable error ratio when the building set has a reasonable complexity. This means that the *ALL FEATURES* approach can be used for more difficult sets, while for sets that express a good result with the *CLUSTERING* approach, the later can be used. The advantage of using a genetic algorithm is that this process can be automated instead of having to search for ideal

parameters for the two approaches by hand. The main drawbacks are the computation time required to find a good database and the demanding hardware requirements in order to complete the clustering without running out of memory.

We weren't able to perform additional tests exploring different numbers of images in the training set due to the additional time required by adding more features to cluster; also, in the database creation the clustering was interrupted and discarded when the resource limit (time or memory) was reached. Furthermore, for time constraints we had to limit the size of the population and the number of iterations in the genetic algorithm. The results we obtained might therefore be suboptimal.

It is clear that the initial training set is key. Not surprisingly, a good training set should include multiple pictures of the same building taken in various weather conditions in different days, and negative picture should represent a good variety of the visual features available in the surrounding area. The pictures should also be acquired with the same device that will be used for the application, or should at least be a mix of pictures acquired with various devices. Tests run on the mobile device (or images captured with it) are still performing worse than expected. The reasons lie in both a somewhat unstable matching step and in a non trivial picture set.

A proper comparison with other approaches in literature and on the market is difficult both for the difference in the architecture (local database versus remote database) and for the lack of data. The commercial applications don't disclose the internals (e.g. where the matching step is performed or the size of the database) and no proper tests regarding the error ratio are available. Considering other works in literature, [32] outperforms our solution in error ratio with a comparable speed but our database size is sensibly smaller. In this regard we achieved our goal of creating a compact feature database.

7.2 Future work

This project has multiple hooks for future enhancements. Improvements can be obtained in the database creation and in the feature matching and the project could benefit of some more tests to have a better understanding of the proposed approaches.

7.2.1 Tests and comparison

The performances of our framework have not been tested extensively with mobile-acquired pictures. Given time and proper hardware, all the test could be re-run with no clustering time limit and hopefully without incurring in memory limit too often (i.e. using a 64 bit system). A complete suite of test with mobile-acquired pictures, also increasing the number of images in the training set, is the first addition to this project to be considered.

Given the proper time, the use of other settings for the genetic algorithm can be tested, such as using the PAIRWISE function and increasing population size and number of iterations.

Additionally, an extensive comparison with similar works, possibly using the same images and comparing database size, would help assess the value of this contribution.

7.2.2 Database creation

Regarding the use of a genetic algorithm, our strategy of having the algorithm tweaking a high number of parameters might have not been ideal. Patterns should be searched in the returned results as there might be some parameter values, or range of values, that always perform better than others. In this case, using a genetic algorithm that randomise the value in a wider range is not optimal; if some values are clearly better for all the possible databases, those values should be fixed to avoid useless explorations of the genetic algorithm in a region of the search space where no good solutions can be found.

Visual features clustering proved to be a valid approach to reduce database size and improve robustness. There are certainly code-level refinements that can be obtained but also other clustering algorithms can be explored. We only used clustering algorithms that returns a partition of the features (i.e. each point is in one and only one cluster) and other strategies, such as fuzzy clustering, can be explored.

A good improvements in performances should be obtained by using a approximated nearest neighbour algorithm both in clustering and feature matching. Best-Bin-First [3] or another *k-tree* based approaches should result in a speed up without affecting accuracy drastically, even if the current results

do not suggest improving the speed of the matching step as being a priority.

Also, our strategy of discarding clusters that contains features from different buildings might have been too rough. A possible improvement would be to use probabilities in assigning more than one building to a cluster.

Finally, clusters should be treated as more than just a centre. The standard deviation or the shape should be considered when matching features with clusters. However, this approach would require some changes in the database structure in order to store both regular features and clusters with these additional information - they will no longer be simple metafeatures.

On the storage space side, using a text file with SQL commands is far from perfect. A proper way of saving a database in a binary format compatible with Android's SQLite should be developed.

7.2.3 Feature matching

In our project we obtained good results by simply comparing features and keeping track of votes for a given building. This process can be refined by considering other relation between the matched features. Checking for coherency in size, orientation and/or geometric relations, as seen in [22, 32], would result in a more robust but potentially slower matching step. Having a solid value for size, orientation and translation would also allow us to overlay some graphics on the actual building in the captured image (such as a silhouette or an icon). However, the impact of this modification on clustering and how the clusters are stored has to be considered.

7.2.4 SURF

While SURF is considered to be a fast feature extractor and descriptor, it can still be improved especially in regard to a specific hardware platform [5, 6]. Optimisations in the OpenSURF code should be considered as it is clearly a bottleneck on the mobile application. Improving the speed on the extraction would allow to spend more time on matching if more robust but intensive matching strategies are to be implemented.

Bibliography

- [1] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
- [2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, 2008.
- [3] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *In Proc. IEEE Conf. Comp. Vision Patt. Recog*, pages 1000–1006, 1997.
- [4] M. Brown and D. G. Lowe. Recognising panoramas. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 1218, Washington, DC, USA, 2003. IEEE Computer Society.
- [5] Wei-Chao Chen, Yingen Xiong, Jiang Gao, Natasha Gelfand, and Radek Grzeszczuk. Efficient extraction of robust image features on mobile devices. In *ISMAR '07: Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–2, Washington, DC, USA, 2007. IEEE Computer Society.
- [6] V.F. da Camara Neto and M.F.M. Campos. An improved methodology for image feature matching. In *Computer Graphics and Image Processing (SIBGRAPI), 2009 XXII Brazilian Symposium on*, pages 307 –314, October 2009.

- [7] Chris Evans. Opensurf. <http://www.chrisevansdev.com/computer-vision-opensurf.html>, retrieved 01/09/2010.
- [8] Christopher Evans. Notes on the opensurf library. Technical Report CSTR-09-001, University of Bristol, January 2009.
- [9] Extra Reality Limited. Popcode. <http://www.popcode.info>", retrieved 05/09/2010.
- [10] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.
- [11] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
- [12] Google Inc. Android. <http://www.android.com/>, retrieved 01/09/2010.
- [13] Google Inc. Google goggles. <http://www.google.com/mobile/goggles/>, retrieved 01/09/2010.
- [14] Laurie J. Heyer, Semyon Kruglyak, and Shibu Yooseph. Exploring Expression Data: Identification and Analysis of Coexpressed Genes. *Genome Research*, 9(11):1106–1115, 1999.
- [15] John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.
- [16] M. Kahari and D. Murphy. Mara - sensor based augmented reality system for mobile imaging device. In *ISMAR: Proceedings of the International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2006.
- [17] Yan Ke and R. Sukthankar. Pca-sift: a more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–506 – II–513 Vol.2, june-2 july 2004.
- [18] Kooaba AG. Kooaba. <http://www.kooaba.com/>, retrieved 01/09/2010.
- [19] Layar. Layar. <http://layar.com>, retrieved 01/09/2010.
- [20] Xiaowei Li, Changchang Wu, Christopher Zach, Svetlana Lazebnik, and Jan-Michael Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. In *ECCV*

- '08: *Proceedings of the 10th European Conference on Computer Vision*, pages 427–440, Berlin, Heidelberg, 2008. Springer-Verlag.
- [21] Frank Lorenz Wendt, Stéphane Bres, Bruno Tellez, and Robert Laurini. Markerless outdoor localisation based on sift descriptors for mobile applications. In *ICISP '08: Proceedings of the 3rd international conference on Image and Signal Processing*, pages 439–446, Berlin, Heidelberg, 2008. Springer-Verlag.
 - [22] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1150, Washington, DC, USA, 1999. IEEE Computer Society.
 - [23] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.
 - [24] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.
 - [25] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, 2005.
 - [26] Mobilizy GmbH. Wikitude. <http://www.wikitude.org/>, retrieved 01/09/2010.
 - [27] Oracle Corporation. Java for developers. <http://www.oracle.com/technetwork/java>, retrieved 05/09/2010.
 - [28] Gerhard Reitmayr and Tom Drummond. Going out: robust model-based tracking for outdoor augmented reality. In *ISMAR '06: Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 109–118, Washington, DC, USA, 2006. IEEE Computer Society.
 - [29] Meier Reto. *Professional Android 2 Application Development*. Wiley Publishing, 2010.
 - [30] Duncan Robertson and Roberto Cipolla. An image-based system for urban navigation. In *in BMVC*, pages 819–828, 2004.

- [31] J.E. Smith, T.C. Fogarty, and I.R. Johnson. Genetic selection of features for clustering and classification. In *Genetic Algorithms in Image Processing and Vision, IEE Colloquium on*, 1994.
- [32] Gabriel Takacs, Vijay Chandrasekhar, Natasha Gelfand, Yingen Xiong, Wei-Chao Chen, Thanos Bismpiagiannis, Radek Grzeszczuk, Kari Pulli, and Bernd Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In *MIR '08: Proceeding of the 1st ACM international conference on Multimedia information retrieval*, pages 427–434, New York, NY, USA, 2008. ACM.
- [33] The Open Handset Alliance. Open handset alliance. <http://www.openhandsetalliance.com>, retrieved 05/09/2010.
- [34] J.T. Tou and R.C. Gonzalez. *Pattern recognition principles*. Addison-Wesley, 1974.
- [35] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2008.
- [36] Christoffer Valgren and Achim J. Lilienthal. Sift, surf & seasons: Appearance-based long-term localization in outdoor environments. *Robotics and Autonomous Systems*, 58(2):149 – 156, 2010. Selected papers from the 2007 European Conference on Mobile Robots (ECMR '07).
- [37] Wei Zhang and Jana Kosecka. Localization based on building recognition. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, page 21, Washington, DC, USA, 2005. IEEE Computer Society.