

OBJECT RECOGNITION ON ANDROID MOBILE PLATFORM USING  
SPEEDED UP ROBUST FEATURES

by

Vivek Kumar Tyagi

A Thesis Submitted to the Faculty of  
The College of Computer Science and Engineering  
in Partial Fulfillment of the requirements for the Degree of  
Master of Science

Florida Atlantic University

Boca Raton, Florida

August 2010

© Copyright by Vivek Kumar Tyagi 2010

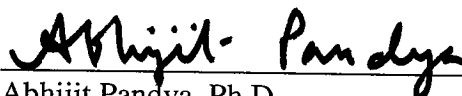
OBJECT RECOGNITION ON ANDROID MOBILE PLATFORM USING  
SPEEDED UP ROBUST FEATURES

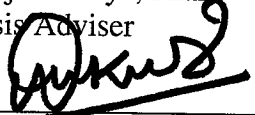
by

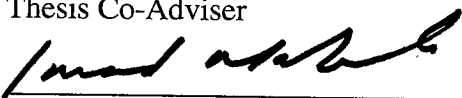
Vivek Kumar Tyagi

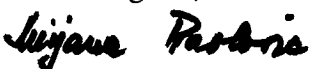
This thesis was prepared under the direction of the candidate's thesis adviser, Dr. Abhijit Pandya and thesis co-adviser Dr. Ankur Agarwal, Department of Computer & Electrical Engineering and Computer Science, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

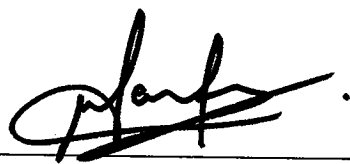
SUPERVISORY COMMITTEE:

  
Abhijit Pandya, Ph.D.  
Thesis Adviser

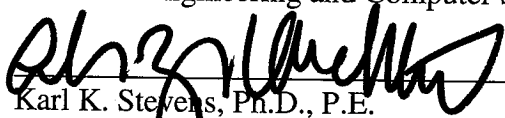
  
Ankur Agarwal, Ph.D.  
Thesis Co-Adviser

  
Imad Mahgoub, Ph.D.

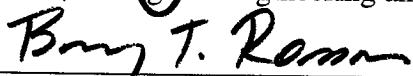
  
Mirjana Pavlovic, M.D., Ph.D.



Borko Furht, Ph.D.  
Chairman, Department of Computer &  
Electrical Engineering and Computer Science



Karl K. Stevens, Ph.D., P.E.  
Dean, College of Engineering and Computer Science

  
Barry T. Rosson, Ph.D.  
Dean, Graduate College

July 20, 2010  
Date

## ACKNOWLEDGEMENTS

It is a pleasure to thank all who made this thesis a success. I am indebted to my supervisors Dr. Abhijit Pandya and Dr. Ankur Agarwal for giving me this wonderful opportunity to work under their guidance throughout my Master's thesis. Their enthusiasm, inspiration and great efforts to explain things clearly and in a simple way helped me to achieve my goals in this study. I would like to thank Dr. Abhijit Pandya and Dr. Sam Hsu for giving me the opportunity to work as Research Assistant on the Motorola Test Vector grant. The grant supported me in achieving my academic goals.

I would like to thank to Dr. Imad Mahgoub for providing support and offering the right direction. I would also like to thank Dr. Borko Furht for pointing me in the right direction and Jean Mangiaracina for her guidance through administrative hurdles.

I want to thank my family for believing in me. To my brother, Ritesh Tyagi, for supporting me. My mother Smt Mithlesh Tyagi for her love and blessing. My father Late Shri Dhrampal Singh Tyagi for his love and blessings. I offer my regards and blessings to all of those who supported me in any respect during the completion of my thesis, especially my friends. But above of all, Thank you God.

## ABSTRACT

Author: Vivek Kumar Tyagi

Title: Object Recognition on Android Mobile Platform Using Speeded Up Robust Features

Institution: Florida Atlantic University

Thesis Advisors: Dr. Abhijit Pandya  
Dr. Ankur Agarwal

Degree: Master of Science

Year: 2010

In recent years there has been great interest in implementing object recognition frame work on mobile phones. This has stemmed from the fact the advances in object recognition algorithm and mobile phone capabilities have built a congenial ecosystem. Application developers on mobile platforms are trying to utilize the object recognition technology to build better human computer interfaces. This approach is in the nascent phase and proper application framework is required. In this thesis, we propose a framework to overcome design challenges and provide an evaluation methodology to assess the system performance. We use the emerging Android mobile platform to implement and test the framework. We performed a case study using the proposal and reported the test result. This assessment will help developers make wise decisions about their application design. Furthermore, the Android API developers could use this

information to provide better interfaces to the third party developers. The design and evaluation methodology could be extended to other mobile platforms for a wider consumer base.

OBJECT RECOGNITION ON ANDROID MOBILE PLATFORM USING  
SPEEDED UP ROBUST FEATURES

List of Tables .....	x
List of Figures .....	xi
Chapter - 1 Introduction.....	1
1.1 Background.....	1
1.2 Motivation.....	3
1.3 Problem Statement .....	4
1.4 Contribution .....	5
1.5 Thesis Overview .....	6
Chapter - 2 Background .....	7
2.1 Background.....	7
2.2 Object Recognition Framework.....	10
2.3 Speeded Up Robust Features .....	12
2.4 Related Work .....	13
Chapter - 3 Matching Algorithm.....	16
3.1 Classification Theory .....	16
3.2 Matching Algorithm.....	17
3.3 Receiver Operating Curve (ROC).....	19
Chapter - 4 System Architecture.....	22

4.1	Overview.....	22
4.2	System Architecture.....	22
4.3	Mobile Component .....	23
4.4	Server Component .....	28
Chapter - 5 Case Study .....		34
5.1	Overview.....	34
5.2	Client Test Platform.....	34
5.3	Android Application Development Setup.....	35
5.4	Server Side Implementation.....	36
5.5	Application Flow Chart.....	37
5.6	Data Flow .....	39
5.7	Test Methodology .....	41
5.8	Latency.....	42
5.9	Effect of Field Condition .....	42
Chapter - 6 Results and Conclusion.....		45
6.1	Threshold Selection .....	45
6.2	Latency.....	47
6.3	Rotation Effect .....	48
6.4	Light Intensity.....	50
6.5	Perspective Transformation .....	51
6.6	Scale.....	52
Chapter - 7 Conclusion and Future Direction.....		54
7.1	Conclusion .....	54



7.2	Future Work .....	55
	Appendix A Source Code .....	57
	References.....	63

## LIST OF TABLES

Table 1 Confusion Matrix.....	19
Table 2 ROC values .....	46

## LIST OF FIGURES

Figure 2-1 Object Recognition Framework .....	10
Figure 3-1 Receiver Operating Curve.....	20
Figure 4-1 System Architecture .....	23
Figure 4-2 Android Application Framework .....	24
Figure 4-3 Client Interface Application .....	26
Figure 4-4 Object Recognition Application.....	27
Figure 4-5 Step One for Populating Database -- Image Set Acquisition.....	29
Figure 4-6 Step Two for Populating Database – Converting Image to Feature Vector....	29
Figure 4-7 Step Three for Populating Database – Transfer of Data to MySQL.....	30
Figure 4-8 Sample Database Schema for Server .....	32
Figure 4-9 Matching Algorithm Application.....	33
Figure 5-1 Application Flow Chart.....	39
Figure 5-2 Data Flow Diagram.....	41
Figure 5-3 Response Curve.....	43
Figure 6-1 ROC Curve for Case Study .....	47
Figure 6-2 Processing Time Distribution.....	48
Figure 6-3 Affine Transformation Legend .....	48
Figure 6-4 Rotation Response Curve.....	49
Figure 6-5 Light Intensity Legend .....	50

Figure 6-6 Light Intensity Response.....	50
Figure 6-7 Perspective Transformation Legend .....	51
Figure 6-8 Perspective Transformation Response Curve .....	51
Figure 6-9 Scaling Legend.....	52
Figure 6-10 Scaling Response Curve.....	52
Figure 6-11 Consolidated Test Results .....	53

## CHAPTER - 1 INTRODUCTION

### 1.1 Background

In recent years, there have been great advances in the field of computer vision. On the theoretical front, the emergence of robust feature vectors, such as the Scale-invariant feature, transform (SIFT) [1], and speeded up robust features (SURF) [2], have increased the accuracy of object recognition algorithms. The central idea of feature-based object recognition algorithms lies in finding interest points, often occurring at intensity discontinuity, that are invariant to change due to scale, illumination, and affine transformation. Technologically, there has been rapid improvement of image acquisition devices. This has resulted in an increase in camera resolution and a decrease in camera price. Due to these factors, computer vision applications moved out of laboratories and entered the main stream. Face recognition, automatic photo tagging, and image search systems now are widely used applications.

Another area of new development is the mobile/handheld devices. Mobile phones are no longer small, resource restrictive, isolated devices. They now are equipped with the computing power equivalent of desktops from five years ago. They have high resolution screens and are equipped with good quality cameras. There have been great strides in the data connectivity of these devices. This has resulted in a client server architecture application ecosystem for mobile phones. With dramatic improvement in these areas, efforts are now underway to put object recognition technology on

mobile phones. Developers have started experimenting with this new technology to expand the scope of human computer interaction. Initial application of this technology has been in the area of barcode readers. Barcode reader applications implement the software version of barcode readers. They use the camera on mobile phones as the input device. The software decodes the barcode and sends the information over the wireless network to the application server. The server then responds back to the client with relevant information. This concept is further extended to use the raw image captured from a camera as the input variable, instead of a barcode. Although the overall processing steps are similar in both cases, the complexity of the problem is increased many times. Information in images is not as explicit as in barcodes. So, much more complex algorithms are required to extract the relevant information. This results in higher demand on the computing resources. The quantity of information to be transferred over the data network also greatly increases and has to be optimized to keep the cost and load on the network low. The algorithm for matching the information from the database on the server side has greater increased latency as the number and the dimension of the search vector increase. This area is still relatively new and the proper framework and design methodology is required to fully utilize its potential. In this thesis we propose a framework to implement object recognition technology on the mobile phones with an Android platform. We analyze the stages of processing to get optimized performance and low cost. This will help third party developers to write better computer vision applications. This analysis could be used by Android developers to improve the API to enhance the performance of the system. Finally researchers could use this framework to further extend the work to other mobile platforms.

## 1.2 Motivation

Studies have shown that among the computing devices owned by individuals, a mobile phone is the one that is most likely to be carried. With the omnipresent coverage of data connectivity on mobile phones, these devices also act as the end point of information exchange over the Internet. In the past, most of the information access has been in terms of textual interaction between the device and source. For example, a search on the Internet would be done by entering a key string in the search box. However, with new innovations in object recognition technology and increasing computational power of mobile phones, there are new avenues opening up for human computer interface. The way users interact with the surrounding and information available over Internet is moving from textual to image based. With a robust object recognition framework, application developers now can provide new ways to provide a richer user experience. Some examples of direct applications could be as follows:

- Object recognition framework will help build visual search applications, wherein the user can get information just by clicking the picture of the object of interest.
- Augmented reality is another emerging application. With this feature, information can be overlaid real time on the camera's view finder. For example, having directions overlaid on the real time view of the surrounding area rather than on a schematic map.
- Real time Optical Character Recognition (OCR) and translation applications are a great help while navigating in a foreign country. With this application, the user can get a translation of a sign in different language just by pointing the camera at the sign.

- A collusion avoidance and guidance application for the visually impaired would help improve their quality of life. Here the user can have the surrounding information captured, processed and read out.
- Proposed framework can be extended to stem cell tracking and identification.
- The framework can be used to build application for medical imaging wherein specific anomalies, like tumor, can be identified from a given image.
- Object recognition via mobile phones is a contributing technology for web 3.0.

Development of an efficient and accurate object recognition framework is the key to all of the above mentioned applications.

### **1.3 Problem Statement**

Implementing an object recognition framework has the following challenges.

#### **1.3.1 Distortions at Acquisition Stage**

Cameras on the mobile devices do not provide distortion-free images; these images have geometric and photometric distortions. These distortions could result in increased false negative results for the system.

#### **1.3.2 Time Constraints**

For mainstream applications, the system must meet certain time constraints. If the time taken to provide results is high, the application will fail to provide an interactive user experience.

#### **1.3.3 Computing Constraints**

Although the processing power on Mobile devices has increased, object recognition algorithms take a significant amount of computing resources. They have to be carefully implemented so as not to dominate all the resources on the device.



### **1.3.4 Bandwidth Constraints**

The data transferred over the mobile network still is a costly affair. Also the data speeds are slow compared to other modes of transfers. This has a direct impact on the cost effectiveness of the solution as well as the latency. Therefore, the application should send as little information as possible over the network.

### **1.3.5 System Architecture and Test Framework**

For implementing an efficient application in this domain, we require a end-to-end system architecture and test framework.

## **1.4 Contribution**

In this section we discuss the contribution of this thesis. System architecture developed in this thesis consists of an object recognition framework for mobile devices and an implementation of classification algorithms. Case studies also were conducted using a mobile device, and the results are presented.

### **1.4.1 System Architecture for Application**

We provide a client server-based system architecture for an object recognition application on an Android mobile platform.

### **1.4.2 SURF Implementation**

For overcoming geometric and photometric distortions, we present an implementation of a SURF algorithm on Android.

### **1.4.3 Load Balancing**

We provide a framework for choosing the computational balance between the client side and server side computation.

#### **1.4.4 Evaluation Framework**

We provide an evaluation framework of the proposed system, which can be extended further to any system for mobile object recognition.

#### **1.4.5 Case Study**

We perform a case study using the architecture, and the results are reported.

#### **1.4.6 Classification Algorithm**

A nearest neighbor search-based classification algorithm is proposed for matching feature vectors.

### **1.5 Thesis Overview**

In chapter 2, we provide the background and related work for this thesis topic. In chapter 3 we present the architecture of the proposed system. Chapter 4 describes a case study for implementing the proposed architecture, along with a test methodology. Chapter 5 discloses the results of the tests conducted on the case study, conclusions and future work of the thesis. Finally, we provide the code for the implementation in Appendix A.

## CHAPTER - 2 BACKGROUND

### 2.1 Background

Object recognition has been an active area of research for many years now. There have been various approaches to this problem and over the years the algorithms have become more robust. The initial research had focused on taking clues from image understanding in humans. Along these lines, a recognition by components approach was proposed by Biederman [19]. The earlier application and evaluation had been on human face recognition [20][21][22][23]. The reason for choosing face detection was the availability of vast amount of test images and the fact that this was the only practical application of computer vision at that time. With the improvement in camera technology and the explosion of Internet connectivity, object recognition now has become a multiclass problem with varied applications. Most object recognition techniques have converged to using features vectors as a representation of an image. These features require two operations. First we need criteria to find the location of these features in an image; the algorithms that are used to do this are called feature detectors. After the location and region of the feature have been found, we require a representation of these regions. The methods to describe the regions are called feature descriptors. Properties of an ideal feature descriptor would be as follows [25]. The features should be localized rather than representing the image as a whole. Having local features helps in overcoming

difficulties due to occlusion and clutter. It is desirable that the features be invariant to common transformations like scale, rotation and light conditions. The feature should be robust so that anomalies like noise, blur, discretization, and compression, etc. do not have a big impact on the feature. The feature should be distinctive so that an individual feature can be matched to a large database of objects. It should be possible to generate the feature even from small objects, so that fine details could be generated. The feature should be computationally efficient so that it could be utilized to build close to real time applications.

Some of the prominent feature detectors are as follows. Harris-Laplace [27] are detected by the scale-adapted Harris function [26] and selected in scale-space by the Laplacian-of-Gaussian operator [26]. Harris-Laplace detects corner-like structures and is invariant to rotation and scale changes. Hessian-Laplace regions [30][31] are localized in space at the local maxima of the Hessian determinant [29] and in scale at the local maxima of the Laplacian-of-Gaussian. These are invariant to rotation and scale changes.

Hessian-Affine regions [32] are invariant to affine image transformations. Localization and scale are estimated by the Hessian-Laplace detector and the affine neighborhood is determined by the affine adaptation process.

After the region has been detected, it is encoded using a descriptor vector. The descriptor algorithm vector takes the parameters from the detected region and converts it to feature vectors. There are many region descriptor algorithms available; SIFT is the most popular and robust among them. A SIFT descriptor is a 3D histogram of gradient location and orientation, where location is quantized into a 4x4 location grid and the gradient angle is quantized into 8 orientations. The resulting descriptor is of dimension 128. Each

orientation plane represents the gradient magnitude corresponding to a given orientation. To obtain illumination invariance, the descriptor is normalized by the square root of the sum of squared components.

A shape context descriptor is similar to the SIFT descriptor, but is based on edges. Shape context is a 3D histogram of edge point locations and orientations. Edges are extracted by the Canny detector.

A PCA-SIFT [33] descriptor is a vector of image gradients in x and y direction, computed within the support region. The gradient region is sampled at 39x39 locations; therefore the vector is of dimension 3042. The dimension is reduced to 36 with principle component analysis [34] PCA.

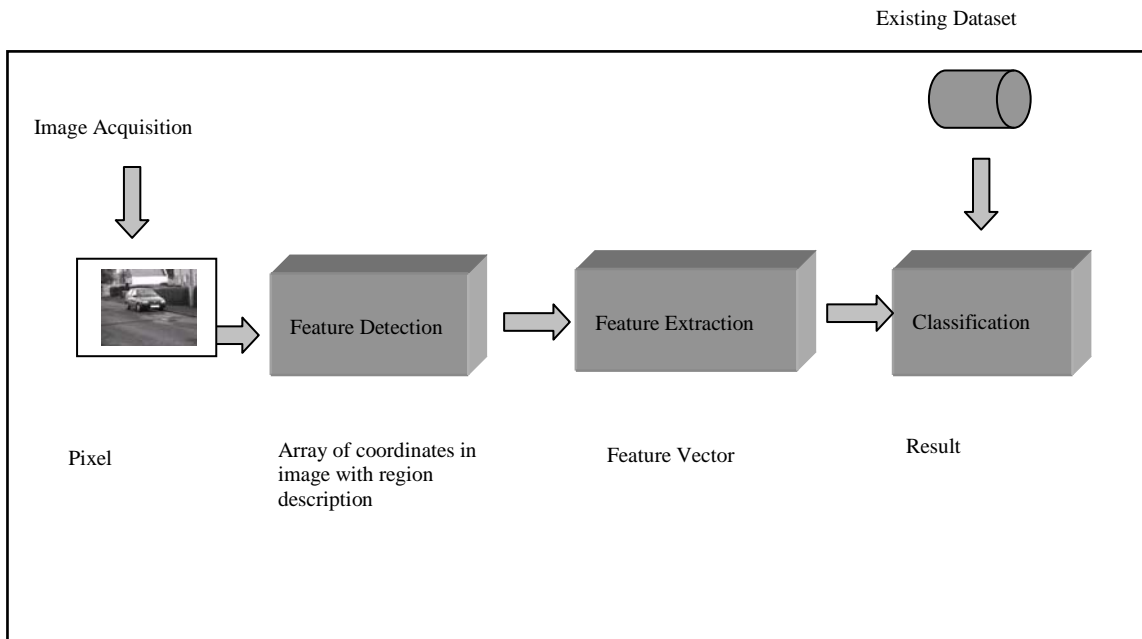
A spin image [36][37] is a histogram of quantized pixel locations and intensity values. The intensity of a normalized patch is quantized into 10 bins. A 10 bin normalized histogram is computed for each of 5 rings centered on the region. The dimension of the spin descriptor is 50.

Among these descriptors is the Speed up Robust Feature (SURF) proposed by Herbert Bay et al. in 2006. It is a robust feature, invariant to scale rotation and brightness, and it is partially inspired by SIFT. SURF finds interest points in the image by a Fast-Hessian Detector. It uses "Integral Image" for faster calculation of intensities in rectangular regions. Haar wavelet functions are used to calculate descriptor of the interest point's surrounding area. It is claimed to be more robust against a different image transformation than SIFT. At the same time, it is less computationally intensive than SIFT. This makes it an ideal candidate for use in a computationally constrained

environment such as mobile devices. For this reason, we are using SURF in this thesis for interested point detection and description.

## 2.2 Object Recognition Framework

In this section, we discuss the main steps in an object recognition framework, illustrated in Figure 2-1. Regardless of whether the implementation is a client server model or a monolith application, the steps are the same.



**Figure 2-1 Object Recognition Framework**

### 2.2.1 Image Acquisition

The first step in any object recognition framework is the acquisition of image by camera. Here camera refers to the digital camera found on handheld devices. The camera converts the analog light into an array of intensity pixel data. Most of the object recognition algorithm works on grayscale images. The role of color in object recognition is debatable and few studies have indicated that it is useful only under certain conditions [38] The camera's ability to capture distortion free images greatly affects the matching

algorithm. Geometric and photometric distortion may add noise to the image data, which could result in false negatives. The output of the camera is a pixel matrix with normalized pixel intensity.

### **2.2.2 Feature Detection**

The feature detection step involves finding “interesting” points in an image. The points could be corners, edges, etc. The important criteria for these points are repeatability and invariance. This means that the algorithm should be able to find the same points in multiple images of the object. Some of the algorithms used are discussed in section 2.1. The output of the feature detection algorithm would be a numeric description of the region.

### **2.2.3 Feature Extraction**

After the interest point has been found, the next step is to encode it. The encoding process involves taking data from the feature detection step and converting it into a feature vector. SIFT and SURF are the foremost algorithms in this domain. The choice of algorithm and optimization is a major area of work when it comes to mobile platforms. As these are the most computational intensive steps in the framework, care has to be taken in selecting the right one. For an interactive application, it is important that the results are presented as fast as possible. In this regard, SURF has a better performance over SIFT. The output of the feature extraction step is a set of feature vector. The dimension of the vector depends on the algorithm used; for example, SIFT generates a vector of dimension 128, whereas SURF generates a vector of size 64.

#### **2.2.4 Classification**

The classification step gives the final answer to the class of the object. That is to say, in this step we get to know “what” this object is. The classification step can be implemented in many ways and the choice of the methods is dictated by the tradeoff between speed and accuracy. The central idea in the entire classification algorithm is the notion of “distance.” In a two dimensional vector space, a distance can be imaged as a straight line between the two points. However, most of the classification algorithms have to deal with multidimensional vector space. The theoretical aspect of classification is discussed in detail in section 2.4. The output of the classification step is a “label” or identifier of the object present in the initial image.

After the traditional object recognition is complete, we can search further for more information about the object, depending on the application. For example, if we know the ISBN number of the book in the image, we might want to retrieve the information about its price, author information, etc.

#### **2.3 Speeded Up Robust Features**

The speeded Up Robust Feature is an image detector and descriptor proposed by Herbert Bay et al. in 2006 [2]. It is a robust feature, invariant to scale rotation and brightness. It is partially inspired by SIFT, claims to be more robust against different image transformation than SIFT. At the same time, it is less computationally intensive than SIFT. This makes it an ideal candidate for use in a computationally constrained environment, like mobile devices. SURF can be used for object recognition, object tracking, augmented reality, and 3D reconstruction. The original SURF algorithm is composed of three stages. In the first stage, interest points are found in the image by a



Fast-Hessian Detector. In the second stage, Haar wavelet responses for both x and y directions are calculated around the interest point and the most dominant direction is chosen to achieve rotation invariance. In the last stage, Haar wavelet functions are used to calculate a descriptor of interest points surrounding the area. The obtained descriptor is invariant against changes in scale, rotation, and brightness. The SURF algorithm contains several optimizations. The most significant improvement in calculation speed is achieved by use of “Integral Image,” which allows fast calculation of filter responses used in all previously mentioned stages.

## **2.4 Related Work**

The application of object recognition to mobile platforms is an area of ongoing research. Even though the current systems are complex, lengthy and prone to error, careful analysis of user behavior has shown that the end user appears content to use these systems. The application of this system started with a simple application involving barcodes. Barcode scanners are comparatively easier to implement as the information is explicitly represented. Also, the barcodes are one dimensional data and hence the complexity of computation is low. The direct application of barcode scanner application is to help the user find pricing information about the products for which they are shopping [51]. With the advent of an efficient object recognition algorithm like SURF, the application were extended to recognize book covers and CD covers [54]. These application delivers acceptable performance and accuracy. Apart from object recognition, augmented reality is another area of application. Augmented reality is the technique of overlaying relevant computer generated information over a live image in real time. Depending on the context, the real world can be viewed as a canvas for information. The

Augmented reality application relies on object recognition algorithms to infer the context and overlay the information. There has been several implementations of augmented reality, both on the iPhone [56] and Android [55]

The most ambitious application of object recognition on mobile platform has been with Google goggles [57] by Google. This application aims to recognize any object in the image captured by the phone. It can recognize varied classes of objects including barcode, visiting cards, book covers, locations, etc. The framework for building such applications are very similar. The focus is on finding a robust and computationally effected feature descriptor for the image. SIFT has been proven to be a robust feature for object detection. However, SIFT is very demanding in terms of computational resources. This is where SURF has proven to provide a good balance of robustness and computational efficiency. As such, SURF has been proposed as a good choice for an building object recognition framework for mobile devices.

Similar to this thesis, there has been various implementations of the SURF algorithm on Android. Implementation of SURF on Android. One of the approaches is the use the OpenCV library for Android. OpenCV has been ported on android as a library [16] and can be used for implementing SURF. Another approach is to take the desktop version of SURF and try to port it to the Android platform. The SURF algorithm has been implemented as libraries in different languages like java, C++ C# and even in ActionScript for Adobe Flash Platform[6]. The original implementation is available as a closed source library.[2] Open source implementation of SURF is available as C++ and C# libraries [3] [5] For Java, the algorithm has been implemented as a stand alone library [8]and a plug-in to ImageJ [1]Software [7]. Some implementations of SURF rely on

other libraries for low level routines, like parallel SURF [12] (based on Pan-o-manic) and OpenCV SURF (based on OpenCV [10][54]), or are a part of a generic library such as Pan-o-manic [11]. In addition to general purpose languages, SURF also has been implemented on Matlab (SURFmex[13]) and has several GPU implementations like speeded up SURF [14] and GPU SURF.

Android is a Java-like programming environment. It also supports native code via JNI. So either we can take the java implementation of SURF and then port it to the Android environment or we can take the native implementation of SURF and use the native function via Java Native Interface (JNI). A good comparison of the two approaches has been provided in the implementation of AndSurf. For this thesis, we have taken a java-based SURF implementation JopenSurf [18] and ported it to the Android environment.

Similar framework has been proposed in previous works [17]. However the implementation of a test framework to assess the effect of various conditions is a major contribution of this thesis. Most of the related work on the field presents the results as a measure of the accuracy of the system. There was no methodology of testing the accuracy under different test conditions. In this thesis we provide this framework.

## CHAPTER - 3 MATCHING ALGORITHM

### 3.1 Classification Theory

Classification is an area of study under machine learning, where the goal is to place the observations into groups (classes) based upon their quantitative attributes. Observations for our context are the feature vector of the input image, which we get after the feature extraction stage (section 2.2.3). Classification has two distinct types. We may be given a set of observations with the aim of establishing the existence of classes or clusters in the data; or we may know for certain that there are so many classes, and the aim is to establish a rule whereby we can classify a new observation into one of the existing classes. The former type is known as Unsupervised Learning (or Clustering), the latter as Supervised Learning. For the current context we will be using supervised learning. Therefore, we start with the set of observations for which the class value is known and then for a new observation the classifier has to find the most appropriate class value.

Let the training set be a set of feature vector-class pair

$$\{(i_1, c_1), (i_2, c_2), (i_3, c_3), \dots, (i_n, c_n)\}$$

Where  $i$  represents the SURF feature vector of an image.

$i \in I_d$  here  $d = 64$  and  $c \in C$  represents the image identifier, like the ISBN number of a book.

Then the goal is to produce a classifier  $H : I \rightarrow C$ , which will map any new feature vector  $i_x \in I_d$  to its true classification label  $c_x \in C$  by some rule.

This rule can be selected based upon the implementation. For this thesis, we chose the Euclidian distance as a measure of distance between the two vectors. For the search of class label from the database, we utilize the nearest neighbor search methodology.

### 3.1.1 Nearest Neighbor Search

In the nearest neighbor search strategy, the query vector is paired up with every feature vector stored in the database. Therefore, given a query vector  $i \in S_d$  and a set of features  $T$ , the nearest neighbor of  $i$  is the vector  $i_1 \in T$  with the smallest Euclidean distance. The query vector will be labeled with the same class as  $i_1$  if the ratio  $\frac{i_1}{i_2}$  between the two closest neighbors is smaller than a threshold  $\phi$ .

### 3.2 Matching Algorithm

As discussed in section 2.4, there are many ways in which a matching algorithm can be implemented. Here we use the Euclidian distance as measure for the similarity of the feature set. Each image has a set of SURF feature vectors associated with it, which are 64 dimensional vectors. The image retrieval system works by first calculating the match number, which is the measure of “similarity” between two images. It is calculated as follows.

Let  $I$  and  $J$  be two images. After applying the SURF algorithm, these images are represented as follows:

$$I = \{i_1, i_2, i_3, \dots, i_n\}$$

$$J = \{j_1, j_2, j_3, \dots, j_m\}$$

Where  $i, j \in R^d$  here  $d=64$ .

To compare the images we have to match each of the feature vectors in image I with that in image J, and then the reverse.

For this we use the Nearest Neighbor Search (Section 2.4.1) to find the match for vectors in  $I$ .

Let  $IJ_{match}$  be the number of vectors in  $I$  that found a match.

Repeating the same procedure, we find

$$JI_{match}$$

The total number of matches thus found is termed as Match Number:

$$\text{Match Number } M_{ij} = IJ_{match} + JI_{match}.$$

Match Ratio = Match Number / Total Number of Pairs

$$MR_{ij} = \frac{\text{Match Number}}{\text{Number of vectors in I} + \text{Number of vectors in J}} = \frac{M_{ij}}{|I| + |J|}$$

For simplicity  $MR_{\theta}$  is expressed as a percentage.

The Match Ratio is the measure of degree of similarity for the images  $I$  and  $J$ .

For the database search:

Let  $D \in \{d_1, d_2, d_3, \dots, d_n\}$  and  $q$  be the query image. We first calculate the Match

Ratio for the query image for each image in the database.

$$\{MR_{qd1}, MR_{qd2}, MR_{qd3}, \dots, MR_{qdn}\}$$

We then select a threshold for selection  $MR_{\theta}$ .

The Matched images are those that have  $MR \geq MR_{\theta}$ .

### 3.3 Receiver Operating Curve (ROC)

To start with ROC, we first need to define True Positive and False Positive states for our system.

For a two-class classification, the following would be the outcomes, as shown in Table 1:

		Predicted Class	
		Yes	No
Actual Class	Yes	True Positive (TP)	False Negative (FN)
	No	False Positive (FP)	True Negative (TN)

**Table 1 Confusion Matrix**

This representation is called a confusion matrix.

On this basis we define

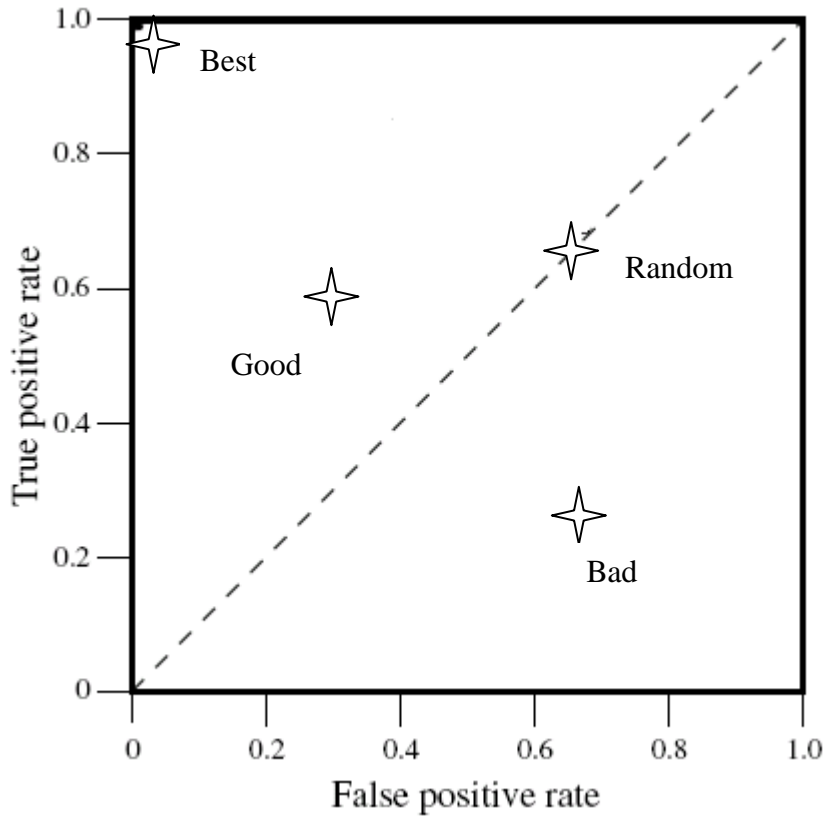
$$\text{True Positive Rate } TPR = \frac{\text{True Postive}}{\text{Total Numer of Positive}}$$

$$\text{Hence } TPR = \frac{TP}{TP + FN}$$

$$\text{False Positive Rate } FPR = \frac{\text{False Postive}}{\text{Total Number of Negative}}$$

$$\text{Hence } FPR = \frac{FP}{FP + TN}$$

A Receiver Operating Curve is a two dimensional graph in which the True Positive Rate (TPR) is plotted on the Y-axis and the False Positive Rate (FPR) is plotted on the X-axis, as shown in Figure 3-1.



**Figure 3-1 Receiver Operating Curve**

For a given classifier, the experiments are carried out with different threshold  $MR_{\Theta}$  values. The results of the test are plotted on the ROC curve. The points in the ROC curve space carry special meaning. The diagonal line  $y=x$  represents the strategy of randomly guessing a class. That is, if a classifier randomly guesses the positive class half the time, it can be expected to get half the positives and half the negatives correct; this yields the point (0.5, 0.5) in ROC space. Any classifier that appears in the lower triangle



performs worse than random guessing. A classifier that is above the diagonal and near to the Y axis is acceptable, because the TPR is greater than the FPR. That means the system is producing less of a false positive case. The best scenario is the upper left hand corner. In this case, the system is producing zero false positive cases and all the right positive cases. For carrying out the experiment, we take a set of test images with 50% of the images in the database and 50% of the images NOT from the data set.

For our experiments:

True positive (TP) state is when we take the input image of a book cover that is present in the database, and the system returns the match in the set of results returned.

True negative (TN) state is when we take the input image of a book cover that is NOT present in the database and the system returns zero results.

False positive (FP) state is when we take the input image of a book cover that is NOT present in the database and the system returns a set of results.

False Negative (FN) state is when we take the input image of a book cover that is present in the database and the system returns zero results.

Based upon the above-mentioned definitions, we can calculate True Positive Rate (TPR) and False Positive Rate (FPR) for our system.

With these definitions we carry out the experiments to find the best  $MR_{\theta}$ .

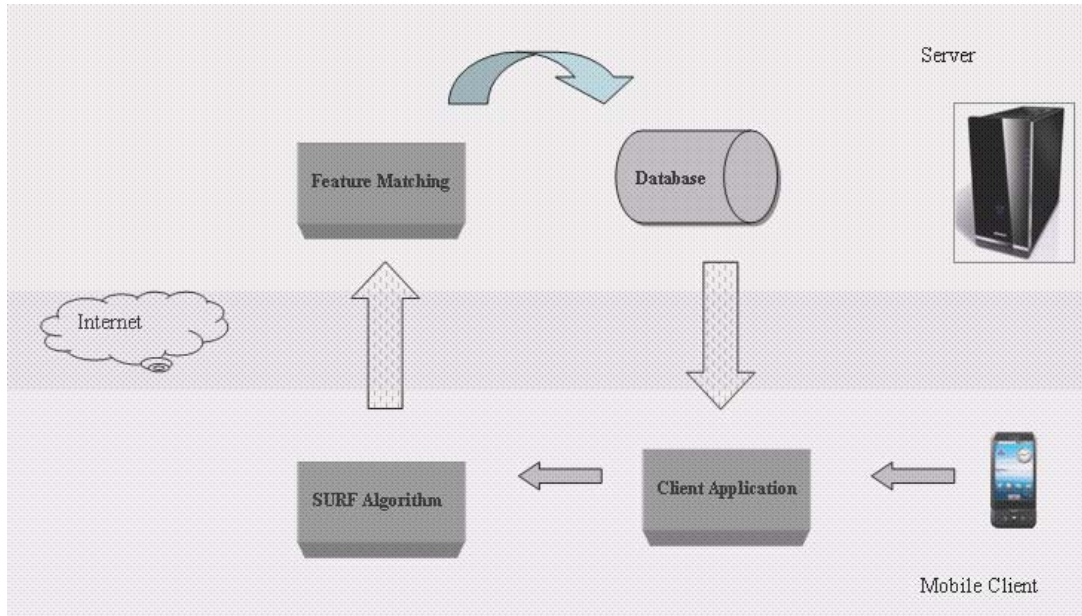
## CHAPTER - 4 SYSTEM ARCHITECTURE

### 4.1 Overview

The system is implemented using client server application architecture. The image processing and feature extraction steps are carried out on the client end. In this thesis, the client end application is a mobile application. The client extracts the SURF vectors from the image and sends it to the server over Hypertext Transfer Protocol (HTTP). The matching of features and information retrieval is done on the server side. The server communicates the result back to the client in Extensible Markup Language (XML) format. This XML data is parsed by the client application and is presented to the user on the mobile screen. On the mobile end, we implement the client application on the Android Mobile Platform. Android is the prevalent mobile framework with a rich ecosystem for application developers. Sever components are web applications that run on a J2EE platform. The J2EE technology is platform independent and has a rich set of web application framework libraries. The server hosts the information relevant to the application on a MySQL database. This allows for easy access to feature information and the metadata. The following section describes the architecture in detail.

### 4.2 System Architecture

The architecture for the proposed system is shown in Figure 4-1. The system runs in parts on the mobile phone (client) and on the server. The major components for the system are as follows:



**Figure 4-1 System Architecture**

Mobile Client component:

- Android platform.
- Client interface application.
- Object recognition algorithm.
- Client communication protocol.

Server Component:

- J2EE platform.
- MySQL database.
- Matching algorithm application.
- Server communication protocol.

### 4.3 Mobile Component

The application component on the mobile end is implemented on an Android platform. The mobile client is responsible for sending the feature vector to the server and

displaying the results of the classification. The first part of the functionality involves image acquisition and feature extraction. The Android platform provides us with a media framework to abstract the image acquisition stage. The feature extraction functionality is provided by implementing the SURF algorithm in Java. For displaying results, the client interface application parses the XML feed received from the server to display the results.

### 4.3.1 Android Platform

Android is an operating system with a Linux kernel as its heart, and it is targeted toward mobile devices. Apart from the Linux kernel, Android has various middle ware libraries to enable high quality applications. Figure 4-2 provides a glimpse into the Android application framework.

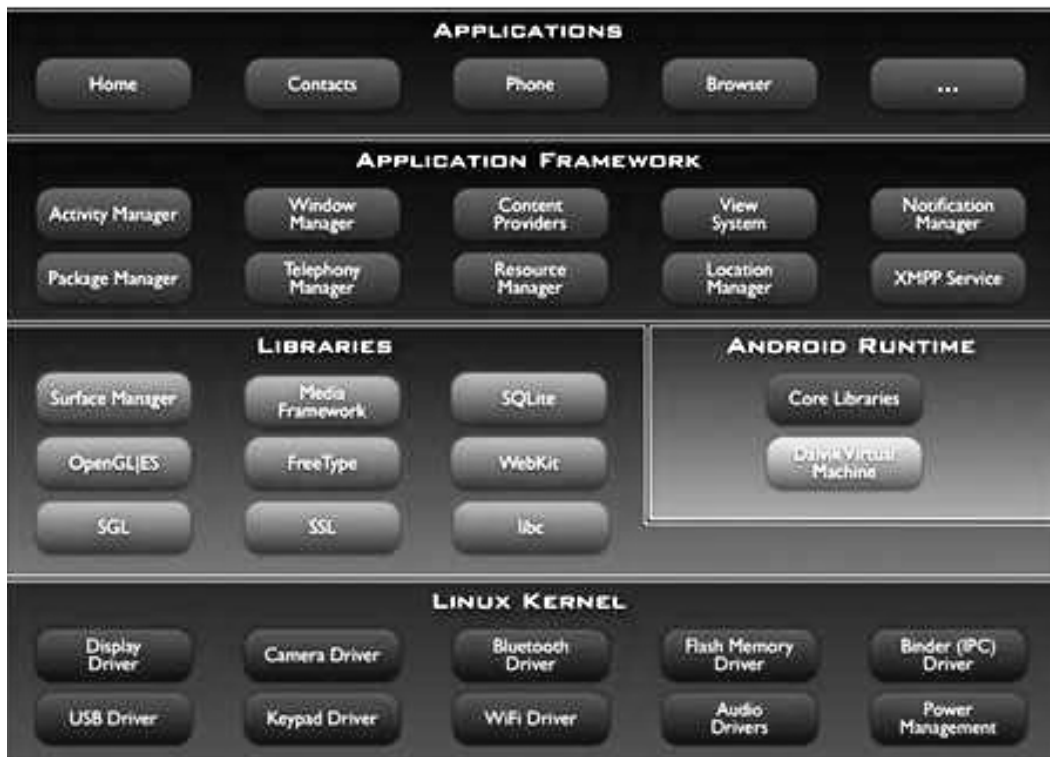


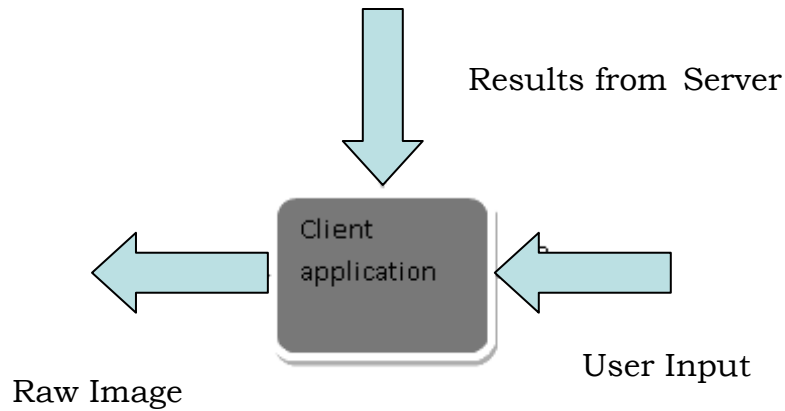
Figure 4-2 Android Application Framework

With the use of the Android application framework, the system has the following components:

- Abstraction of the camera hardware – Android provides APIs access to the camera hardware on the Mobile device. Using these APIs, we get direct access to the image captured by the camera. Android APIs handle all the low level details.
- Java like runtime environment for Algorithm implementation – the Android Framework implements Dalvik JVM. This JVM is very similar to Sun JAVA JVM, and gives us access to most of the java standard libraries, like collections, etc.
- User Interface APIs for building front end application – Android has a rich set of API for user interface development. Android supports inflatable XML layouts for easy customization. This helps in building an efficient and scalable user interface.
- Communication APIs for Data exchange with server – Android supports the java.net library and further extends it with android.net extension. Together these libraries provide a wide range of communication protocols like HTTP.
- Support for XML parsing – Android has built in support for XML parsing; XML is a convenient method for data transfer. With a built in parser, the XML results received from the server can be directly fed to the User Interface components.

#### **4.3.2 Client Interface Application**

Client Interface application is an Android application that executes in the Android runtime, as illustrated in Figure 4-3.



**Figure 4-3 Client Interface Application**

The client application has the following functionality:

- Provide user interface – The client application is the front end of the whole system. Using XML layouts from the Android APIs we build a highly interactive user interface. With the interface, the user can access the camera and acquire the image of interest.
- Link with the feature extraction components – This application provides a channel to forward the captured image to the feature extraction component.
- Interface server communication – This application enables the listener to accept communication from the server.
- Display match results – This application also is responsible for displaying the results from the server. It invokes the XML parser and presents the result to the user.
- Latency logging – This utilizes the profiler to track the time spent on the components on the mobile side.

### 4.3.3 Object Recognition Algorithm

The Object recognition algorithm (Figure 4-4) is the heart of the Client component. Here the SURF algorithm (discussed in the previous section) is implemented. The performance of the overall application depends mostly on the efficient implementation of this application. In this thesis, we implemented the whole application in java. This is to make use of standard APIs provided by the Android framework.

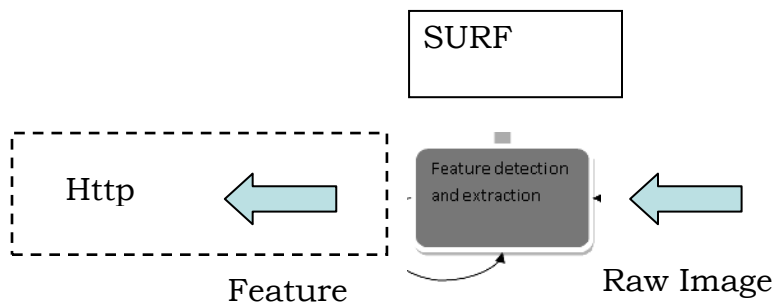


Figure 4-4 Object Recognition Application

Functions:

- Identify point of interest – Using the SURF algorithm, the application finds the point of interest in the given image.
- Extract feature vector – At the point of interest, a 64 valued feature vector is extracted.
- Interface server communication – This application implements the HTTP communication protocol to send this feature vector to the server for image identification.

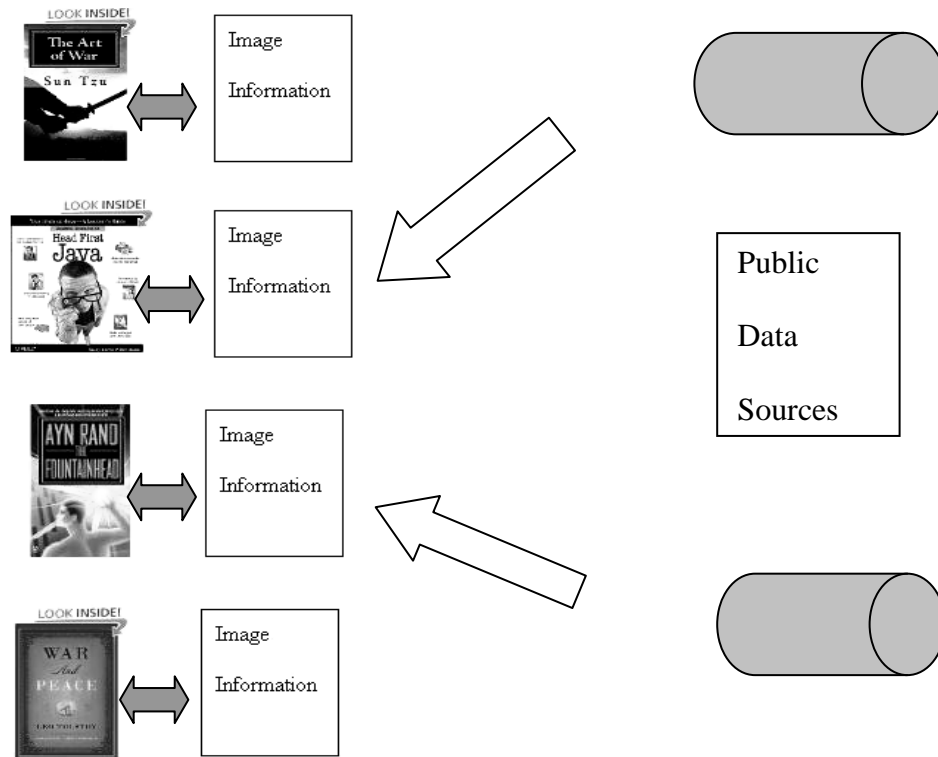
#### **4.3.4 Client Communication Protocol**

The communication Protocol is part of the object recognition application. The function of the protocol is to provide a standard mechanism for data exchange between the server and the client application. Here we use the standard http protocol for communicating with the . For our application, Android provides standard APIs to be invoked for implementing this protocol.

#### **4.4 Server Component**

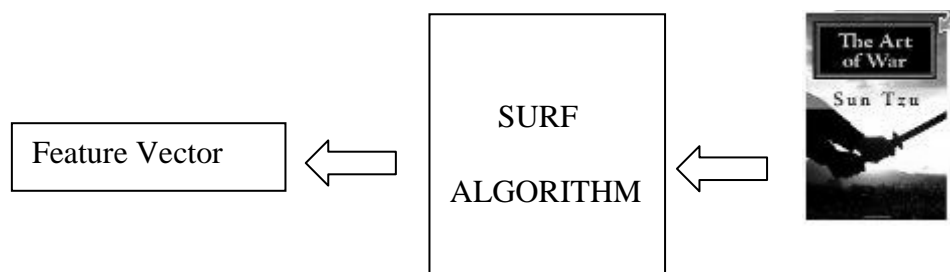
The server end of the application is responsible for implementing the classification algorithm. Before a classification algorithm can be run, we need to create a reference database. This database stores the meta data for the object of interest and the feature vector. First, we gather the images from the relevant categories to populate our reference database. For example, if we are including books and CD covers as categories, we gather images for this data set. The sources are chosen so as to also include Meta data. In Figure 4-5, we present the steps for populating a database for a “Books” category.





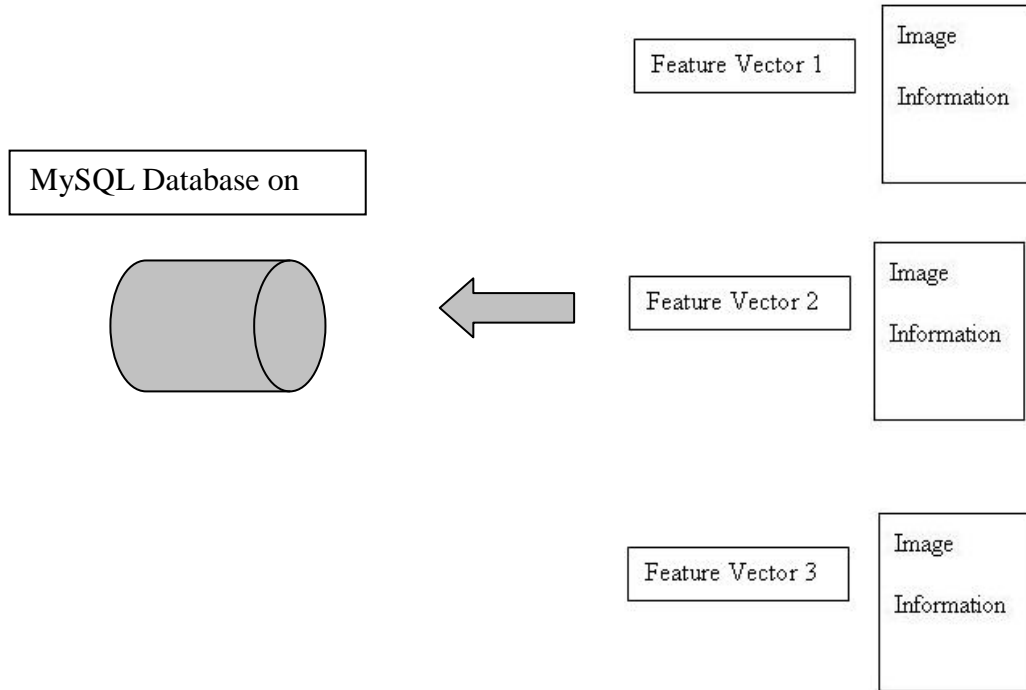
**Figure 4-5 Step One for Populating Database – Image Set Acquisition**

In the second step, we convert the image to feature vector by applying SURF algorithm, as shown in Figure 4-6.



**Figure 4-6 Step Two for Populating Database – Converting Image to Feature Vector**

This feature vector, along with the image information, is then stored in the database, shown in Figure 4-7.



**Figure 4-7 Step Three for Populating Database – Transfer of Data to MySQL**

Here the feature vector received from the client end is matched with the stored image set. The matching algorithm uses a “distance” criteria for deciding upon the best match for the given query vector. The distance criteria depend upon the choice of classification algorithm used in the process. This was discussed in detail in the previous chapter. The server side implements a database for storing the meta data associated with the image set. After the best match is selected, the server application retrieves the meta data of the matched image, which includes relevant information about the image. For example, if the image is a book, then meta data can include the title, author, summary,

price, etc. The server application then encodes this information in XML format and sends it back to the client application. The performance criteria for the server side are the classification algorithm and the database latency. We now will discuss the server side components in detail.

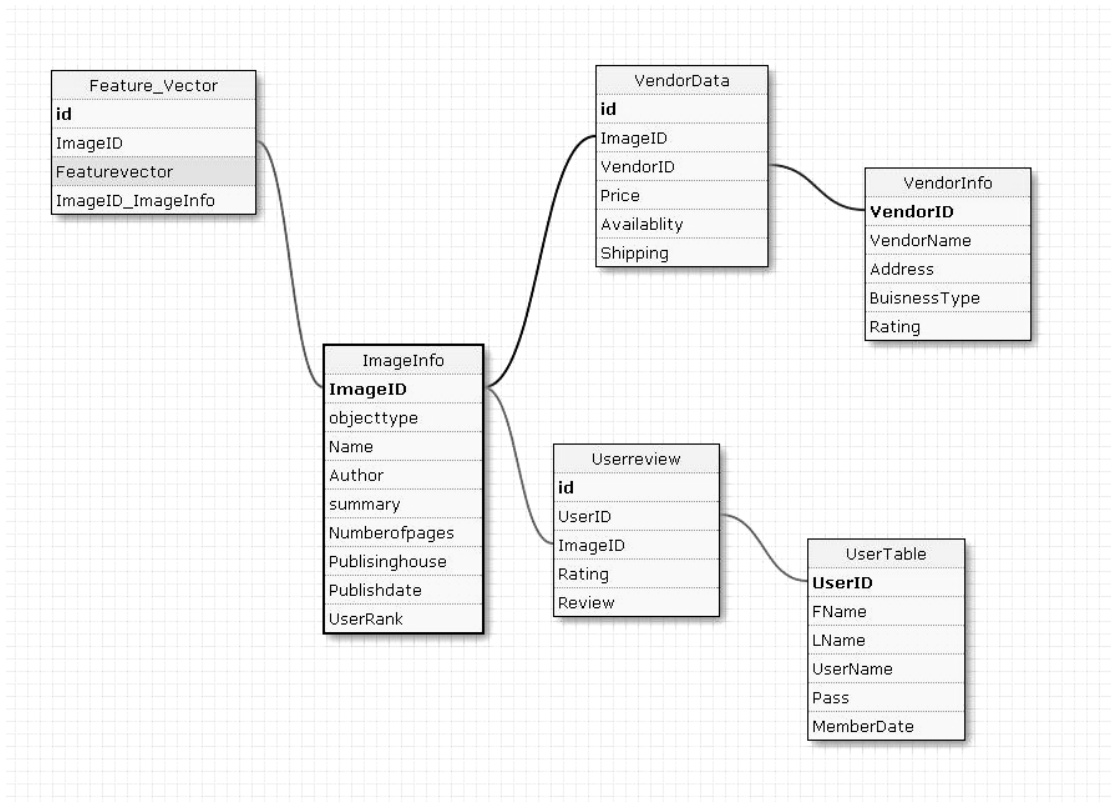
#### **4.4.1 J2EE Platform**

The thesis uses J2EE technology for implementing the server side component. The java platform Enterprise Edition includes libraries that provide functionality to deploy fault –tolerant, distributed, multi-tiered java software, based largely on modular components running on an application server. J2EE provides a component based approach to the design, development, assembly, and deployment of enterprise applications. This approach reduces cost and also enables a fast track through design and implementation. The J2EE platform provides a multi-tiered distributed application model, the ability to reuse components, a unified security model, and flexible transaction control. This enables a faster development of application and results in a scalable architecture. For the application server, the JBoss application server is chosen as it is a Java EE certified platform for developing and deploying enterprise Java applications and Web applications. The JBoss application server also provides the full range of Java EE 5 features as well as extended enterprise services including clustering, caching, and persistence

#### **4.4.2 MySQL Database**

The database on the server side carries the relevant information about the images in the form of a feature vector. The schema for the database will depend on the target application type. However one common table set would be the feature vector to ImagInfo mapping. The feature vector is used by the matching algorithm to find the best match for

the query vector. Once the match is found, the corresponding ImageID is used to find the relevant basic information about the image. As mentioned earlier, depending upon the type of application, the database can contain further information about the image. The query from the client also can contain additional parameters to further refine the information needed to be accessed from the database. In Figure 4-8 we show a sample database schema.



**Figure 4-8 Sample Database Schema for Server**

### 4.4.3 Matching Algorithm Application

As discussed in section 3.4, the first step in the server side processing is to find the most appropriate match for the query vector. The application server forwards the feature vector to the application. In the training stage, the SURF algorithm is applied to



## CHAPTER - 5 CASE STUDY

### 5.1 Overview

For implementing the framework described in chapter 3, we developed a sample application for a case study. This application provides the ability to recognize books from the image captured by the mobile device. With this application, the user will be able to gather all the information about a book just by clicking on a photo of the book with the camera in the mobile device. The user flow chart is described in detail in section 4.4. For implementing the server side, a data set for book cover images is created for this application. The information regarding these images, like the name, author, description, etc. is stored in the database. The images are converted to feature vectors as described in section 3.4. A feature vector to image id mapping is stored in a memory data structure for the matching algorithm. On the client side, the image acquisition and feature extraction application is created along the lines described in section 3.3. The purpose of this case study is to evaluate the various performance parameters of the proposed framework.

### 5.2 Client Test Platform

For the client application we used Android Dev Phone 1. This is essentially an HTC Dream (G1) phone without operator locks. The relevant hardware specifications are:

- Qualcomm MSM7201A ARM11 processor at 528MHz.
- 3.2 inch capacitive touch screen.

- 192 MB DDR SDRAM and 256 MB Flash memory.
- 3.2 megapixel camera with auto focus.
- Quad band mobile network access and 802.11 b/g wireless LAN connectivity.

Also, this phone has Android OS version 1.5

### 5.3 Android Application Development Setup

For developing application for Android OS, Google has provided Android software development Kit (SDK.) [47]The Android SDK has all the tools to build, test and deploy the application for an Android platform. The SDK also comes with a phone emulator [48]that mimics all the function of a real phone. Using the emulator, the application can be fully tested before deploying on the real device.

For developing the application we used eclipse IDE for Java developers [44] 3.5 (Galileo Version). Eclipse provides a integrated framework for writing java application, and provides a plug-in framework for targeting specific development environment. We chose eclipse because it could support both client and server side development. For android development, we used the eclipse plug-in for android [45] and the Android 1.5 SDK [46]summary of the development system is as follows.

#### 5.3.1 Development System.

- Intel Core 2 Duo CPU 2.53 GHz with 2 GB RAM.
- Windows XP professional Service pack 3.
- Sun Java Development Kit 6.
- Eclipse 3.5 Galileo.
- Android SDK 1.5.

## 5.4 Server Side Implementation

The server side of the application provides a feature matching algorithm. It also supports a database that stores the meta data for the image files. The implementation of the server side component is done using the Google App Engine[49], which provides the abstraction of the architecture similar to that proposed in section 3.4. Using this, we can develop and deploy the application at a much faster rate. The app engine also provides us with an analytics feature that helps us test the latency of the algorithm implemented.

### 5.4.1 Google App Engine Development Setup

Google App Engine provides an easy to build maintain and scale infrastructure. The framework runs on the Google infrastructure, hence providing a high quality of service and reliability. The App Engine provides an abstraction to the server side technology. Google App Engine supports apps written in several programming languages. With App Engine's Java runtime environment, we can build our app using standard Java technologies, including the JVM, Java servlets, and the Java programming language - or any other language using a JVM-based interpreter or compiler, such as JavaScript or Ruby. App Engine also features a dedicated Python runtime environment, which includes a fast Python interpreter and the Python standard library. The Java and Python runtime environments are built to ensure that an application runs quickly, securely, and without interference from other apps on the system. We chose app engine because of the many features it provides [50]

- Dynamic web serving, with full support for common web technologies.
- Persistent storage with queries, sorting and transactions.
- Automatic scaling and load balancing.



- APIs for authenticating users and sending email using Google Accounts.
- A fully featured local development environment that simulates Google App engine on your computer.
- Task queues for performing work outside of the scope of a web request.
- Scheduled tasks for triggering events at specified times and regular intervals.

App Engine also supports both Java and Python. For our implementation, we use the Java flavor. The Java implementation allows apps to interact with the environment using the Java Servlet standard, and uses common web application technologies, such as JavaServer Pages (JSPs). The Java runtime environment uses Java 6. The App Engine Java SDK supports developing apps using either Java 5 or 6.

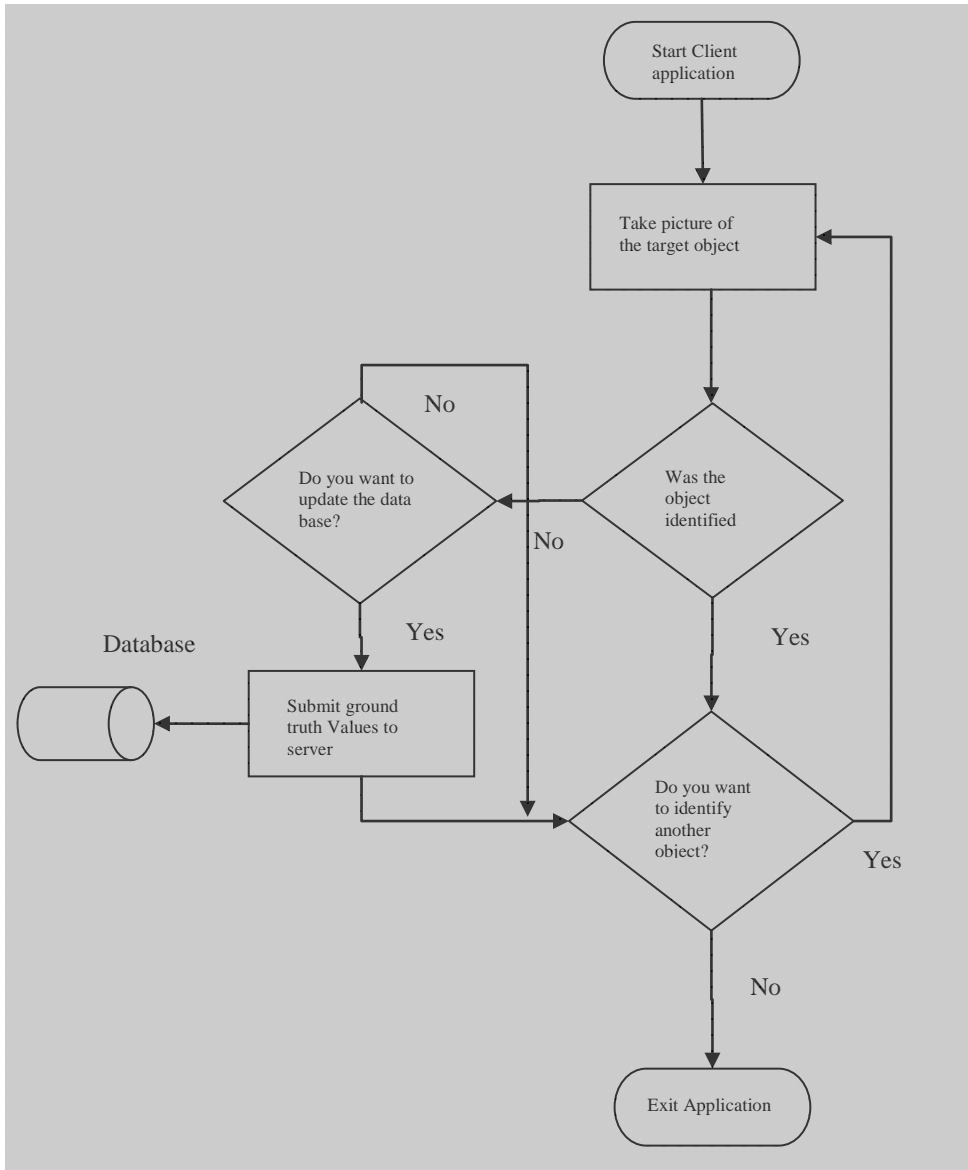
The environment includes the Java SE Runtime Environment (JRE) 6 platform and libraries. The restrictions of the sandbox environment are implemented in the JVM. An app can use any JVM bytecode or library feature, as long as it does not exceed the sandbox restrictions. For instance, bytecode that attempts to open a socket or write to a file will throw a runtime exception.

In the Google App Engine, the application accesses most App Engine services using Java standard APIs. For the App Engine data store, the Java SDK includes implementations of the Java Data Objects (JDO) and Java Persistence API (JPA) interfaces.

## 5.5 Application Flow Chart

Figure 5-1 shows the flow chart for the application use case. The user would initialize the application from the phone's control panel. On start, the application will present the user with a camera viewfinder window to take the picture of the target object.

The user then will take a picture of the book cover. The application will do internal processing to find the appropriate match for the image. When the match is found, the application will display the relevant information about the book in the phone user interface. In case an appropriate match is not found, we can investigate whether the book was in the database. If the information was not in the database, we can have an upload data feature, wherein the information regarding the book can be uploaded back to the server for future usage. The application then provides the user with the choice of either taking another picture for identification or exiting the app.

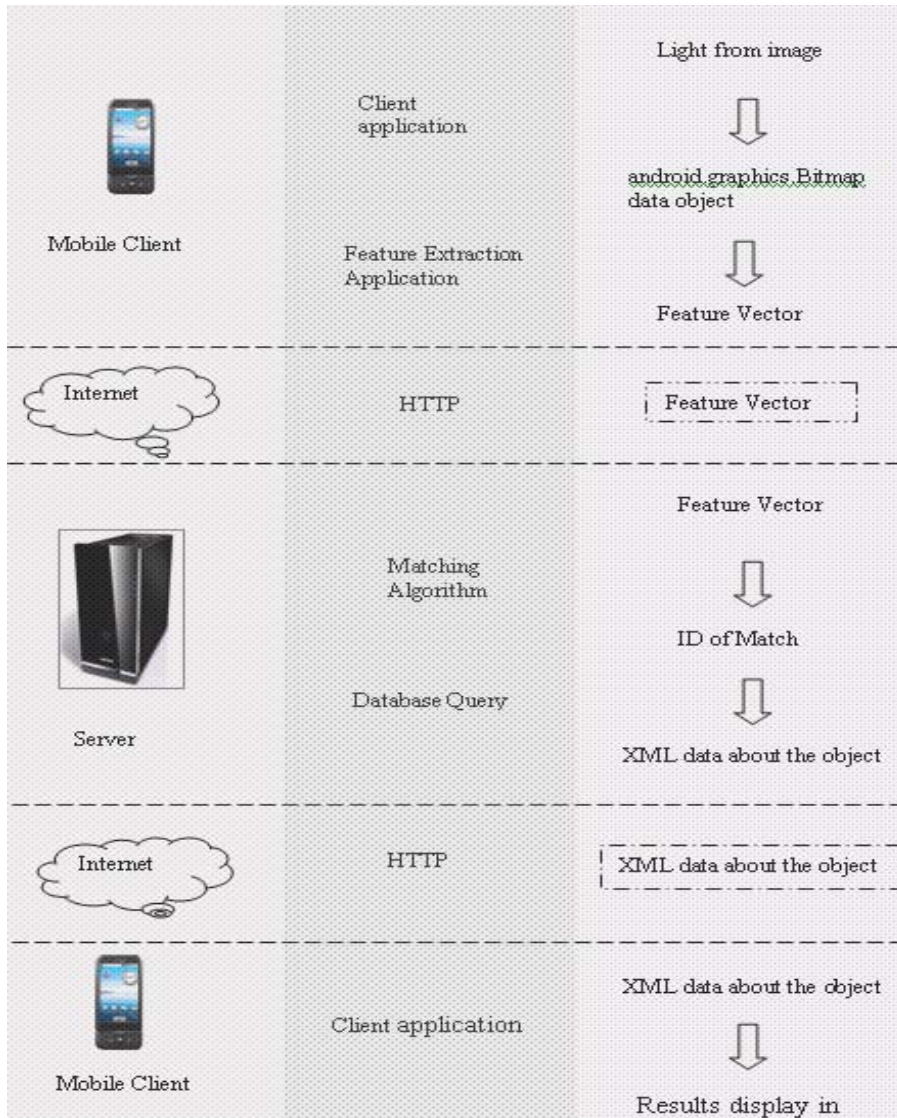


**Figure 5-1 Application Flow Chart**

## 5.6 Data Flow

During the course of execution, the application transforms the data, and in this section we give a glimpse of data flow in the application. Figure 5-2 shows the data flow during different stages, along with the components and the platform where it is happening. This presents a better understanding of the system and serves to provide insight into

optimization opportunities. The first step is the phone camera on the mobile device. The camera captures the image and presents the data in an `android.graphics.Bitmap` data object. This object has the image intensity captured in a pixel array. This data is easily available for manipulation by calling standard methods on the object. The feature extraction application takes the pixel data and calculates the feature vector. An image can have a varied number of feature vectors, depending upon the number of interest points present in the image. The size of each vector is fixed as 64. The elements in the vector represents the SURF feature attributes. The next step in the operation is the transmission of these vectors to the server application. For this, the feature vector set is wrapped in HTTP protocol and sent over to the server. The server front end unwraps the data and makes it available to the Matching algorithm application. Here the feature vector for the query image is matched with the previously stored feature vectors. Based upon a predefined criterion, a match is found. The matching feature vector is identified by the `Image_ID`, which is the primary key to the information stored in the data base about that object. From that key, the database query is executed and the relevant information about the object is extracted. In this step, the information is stored in an XML data object. This XML data object is sent over to the client as a HTTP response. At the client end, the client application parses this data and displays the result to the user.



**Figure 5-2 Data Flow Diagram**

### 5.7 Test Methodology

For testing system performance, we conduct a series of tests on the framework to identify performance and effectiveness. The goal is to present quantitative assessments of the stages of processing in the application. Before the testing phase, the system is setup with the server database. The threshold  $\Theta$  is set to 0.6 as found by similar system implementation.[17] The accuracy of system depends on the value of  $MR_{\Theta}$  (Section

4.4.2). To find the optimum  $MR_{\ominus}$  we plot the Receiver operating characteristic curve (ROC) for our classifier.

## 5.8 Latency

We define latency for the system as the time elapsed from the capture of the image to the display of the results. The latency can be subdivided further into client side latency and server side latency. For testing the latency, we used the Wi-Fi data channels for data transmissions. The software was tagged with system time function call to capture the time stamp. These time stamps were retrieved via log files to further calculate the time difference.

## 5.9 Effect of Field Condition

The image captured by the user from the mobile device will not be in a canonical view like that stored in the database. The image can have a different transformation with respect to the stored image. The transformations occur because the user in the real world will take a picture from the mobile device in varied ways and under different conditions. As this application is targeted towards field implementation, we cannot constraint the user into following a controlled procedure. Hence, we study the different transformations and field conditions that can occur in the field and their effect on the accuracy of the system. To study the effect of the external conditions in a quantitative manner we simulate the field condition by photo editing software. With this we can have better control over our test condition. We used open source software GIMP [59] for this proposes. To test the effects of these conditions, we employ a presentation methodology called response curve. A response curve is a plot of Match Ratio values for all the images in the database for a given image. The response curve shows the discriminative state of the query image with

respect to the rest of the database. An image has a higher probability of being recognized correctly if it has a single clearly defined peak in the response curve. In Figure 5-3 we can see that the query image StateA has a high discriminative state as compared to StateB. By analyzing the response curve, we can find the effect of external conditions on the discriminating power of classifier.

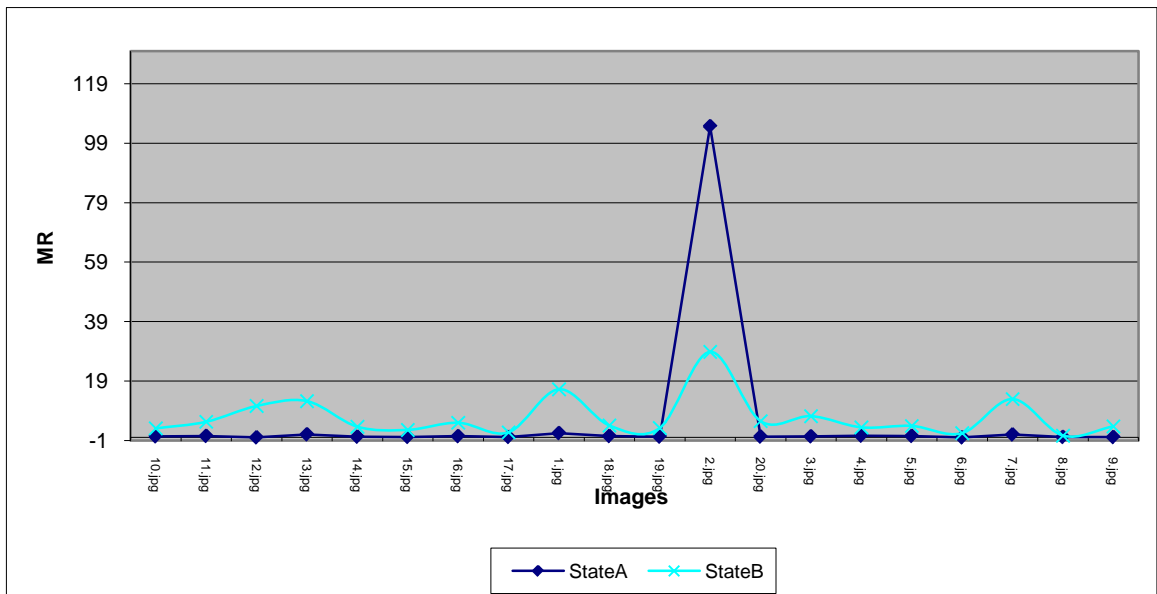


Figure 5-3 Response Curve

### 5.9.1 Rotation

Rotation is a type of affine transformation. Affine transformation is a geometrical transformation that is known to preserve the parallelism of lines but not lengths and angles. Rotation of input image with respect to the view stored in the database can occur when the user clicks a picture of the target object in an angle different from what is stored in the database. To test the effects of rotation, we calculate the response curve for the

image under varied conditions. Using GIMP, we create three different states of the original image with different angles of rotation and analyze the response curve.

### **5.9.2 Perspective View**

Perspective view refers to the image capture situation where the camera plane is not parallel to the object plane. GIMP toolbox provides methods to simulate different perspectives views on given image. We create three images from the original image, representing three different perspective views.

### **5.9.3 Light Conditions**

The images stored in the database are taken in a well-illuminated environment. This is to ensure that the finer details of the target object could be captured. However, in a real life scenario, the light condition can vary and most of the time it is not as good as the image in the database. For testing the effect of light conditions, we create three different image states with decreasing light intensity. Here the light conditions were simulated by using the GIMP tool on the original image.

### **5.9.4 Scale**

Scale refers to the change in dimension of the images while keeping the other geometric properties the same. The scale condition occurs when the user captures the image from a distance different than that present in the database. To simulate the scale condition, we use the GIMP tool to create three different versions of the input image. These are created at different scales. We then analyze the response curve to study the effect of scaling on discriminative state.



## CHAPTER - 6 RESULTS AND CONCLUSION

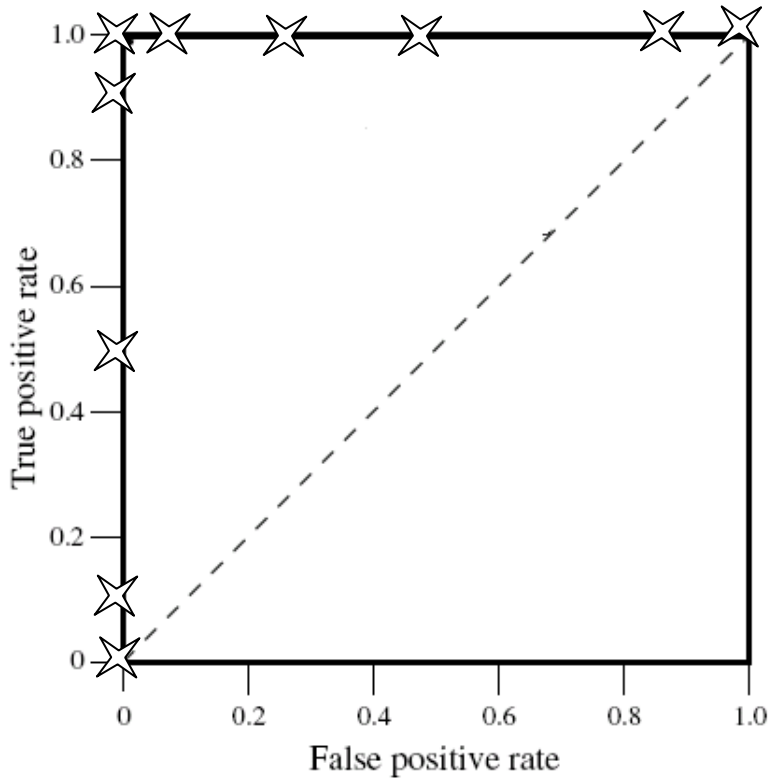
### 6.1 Threshold Selection

In this section we present the results and conclusions from our case study. We start with the selection of  $MR_{\theta}$  from ROC. We did multiple experiments with a data set of 20 images, 10 of which were present in the database (see Table 2).

$MR_{\odot}$	AP Actual Positive	Actual Negative	PP Predicted Positive	Predicted Negative	True Positive	False Positive	False Negative	TN True Negative	TPR TruePositiveRate	FPR FalsePositiveRate
100	10	10	0	20	0	0	10	10	0	0
90	10	10	0	20	0	0	10	10	0	0
80	10	10	0	20	0	0	10	10	0	0
60	10	10	0	20	0	0	10	10	0	0
50	10	10	0	20	0	0	10	10	0	0
40	10	10	0	20	0	0	10	10	0	0
30	10	10	0	20	0	0	10	10	0	0
25	10	10	1	19	1	0	9	10	0.1	0
20	10	10	5	15	5	0	5	10	0.5	0
15	10	10	9	11	9	0	1	10	0.9	0
10	10	10	10	10	10	0	0	10	1	0
2	10	10	11	9	10	1	0	9	1	0.1
1.5	10	10	13	7	10	3	0	7	1	0.3
1	10	10	15	5	10	5	0	5	1	0.5
0.5	10	10	19	1	10	9	0	1	1	0.9
0.1	10	10	20	0	10	10	0	0	1	1

**Table 2 ROC Values**

With these values, we plot our ROC curve, shown in Figure 6.1.

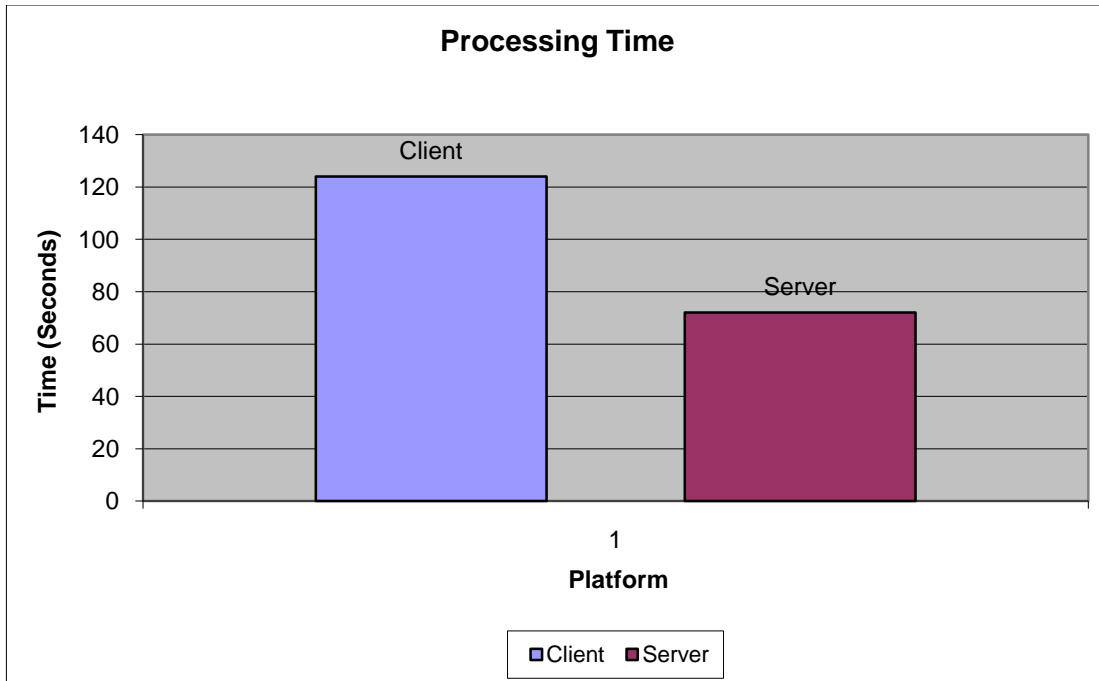


**Figure 6-1 ROC Curve for Case Study**

This curve represents a perfect classifier. The value of  $MR_{\odot} = 10$ .

## 6.2 Latency

We define latency as the time elapsed between capturing the image from the mobile device to the delivery of results. The latency can depend on many factors, such as the processing power of the devices, efficiency of the implementation, transmission time, memory usage, etc. For simplicity we studied the time delay as a function of time taken to process the data on client end and server end. Figure 6-2 shows the average time taken to process the data on different platforms.

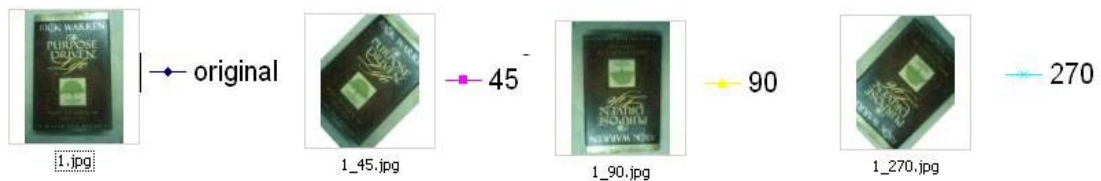


**Figure 6-2 Processing Time Distribution**

We observed that the majority of time is spent at the client end. The opportunity, therefore, lies in improving the algorithm for SURF feature vector extraction.

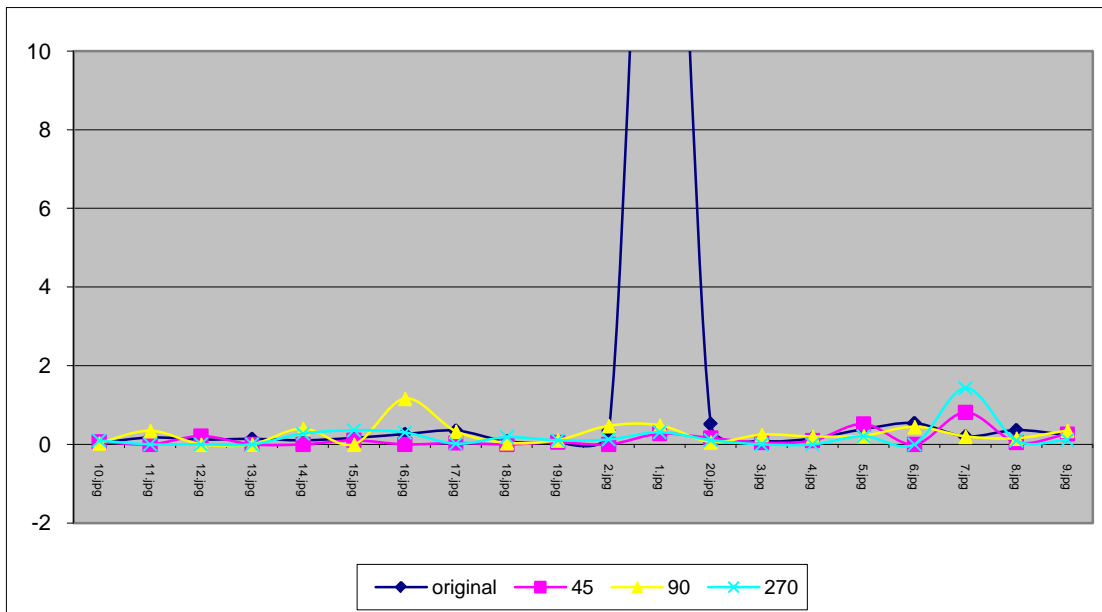
### 6.3 Rotation Effect

As discussed in section 4.9.1, we create three different images from the original image. These images are created at different angles. Figure 6-3 gives a view of this transformation.



**Figure 6-3 Affine Transformation Legend**

For finding the effect of rotation on the system, we study the response of an image (1.jpg) with the database. This image is matched with all the images in the database and its Match Ratio (MR) is recorded. The MR then is plotted on order to analyze the discriminative state of the image. An image has a higher probability of being recognized correctly if it has a single, clearly defined peak in the response curve (like original image in Figure 6-3).



**Figure 6-4 Rotation Response Curve**

We observe that as the angle of rotation increases, the image features lose their discrimination state. The original image has a clearly defined peak. The rest of the images have multiple peaks, which means it would be difficult to discriminate the correct match from the database. We can infer that the current implementation is not robust to the effect of rotation on query data.

## 6.4 Light Intensity

Figure 6-5 depicts the image sample created to represent different levels of light intensities. These images were simulated with the GIMP tool. The numbers 25,50,100 represent the level of screening effect.



Figure 6-5 Light Intensity Legend

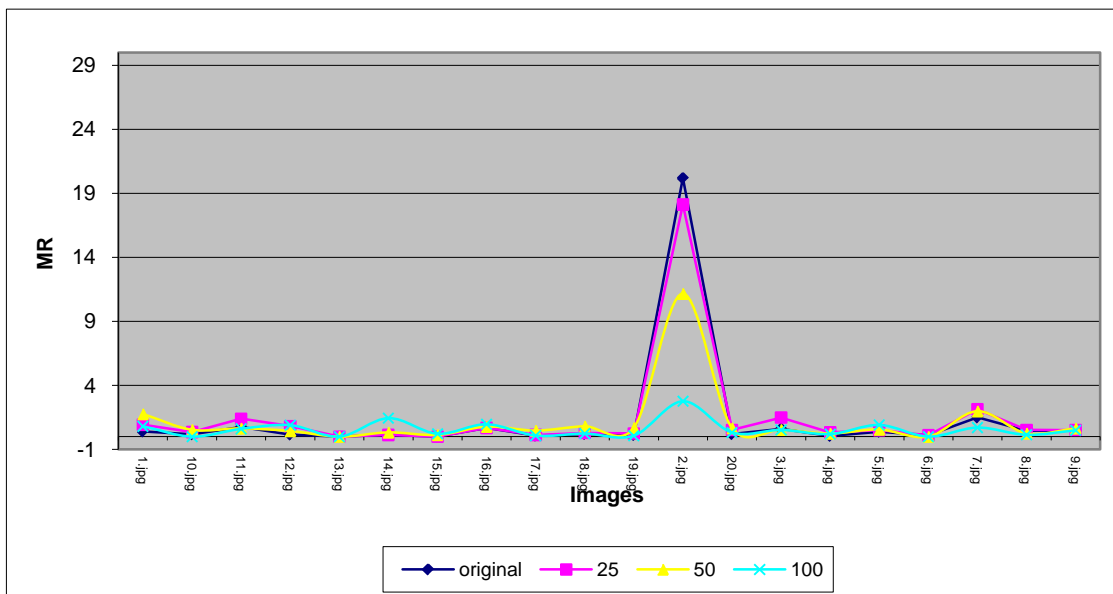


Figure 6-6 Light Intensity Response

As indicated in Figure 6-6, we observed that at lower levels of screening, the images retain their discriminatory power. However, at higher levels, multiple peaks start to emerge. The main peak also reduces in magnitude and becomes comparable to other

smaller peaks. This is indicative of loss in discriminative power. For most of the actual conditions, the system should be fairly stable and give correct results.

### 6.5 Perspective Transformation

Perspective transformation occurs when the camera plane is not parallel to the object. In this experiment, three images were created using the GIMP tool with different views, illustrated in Figure 6-7.



Figure 6-7 Perspective Transformation Legend

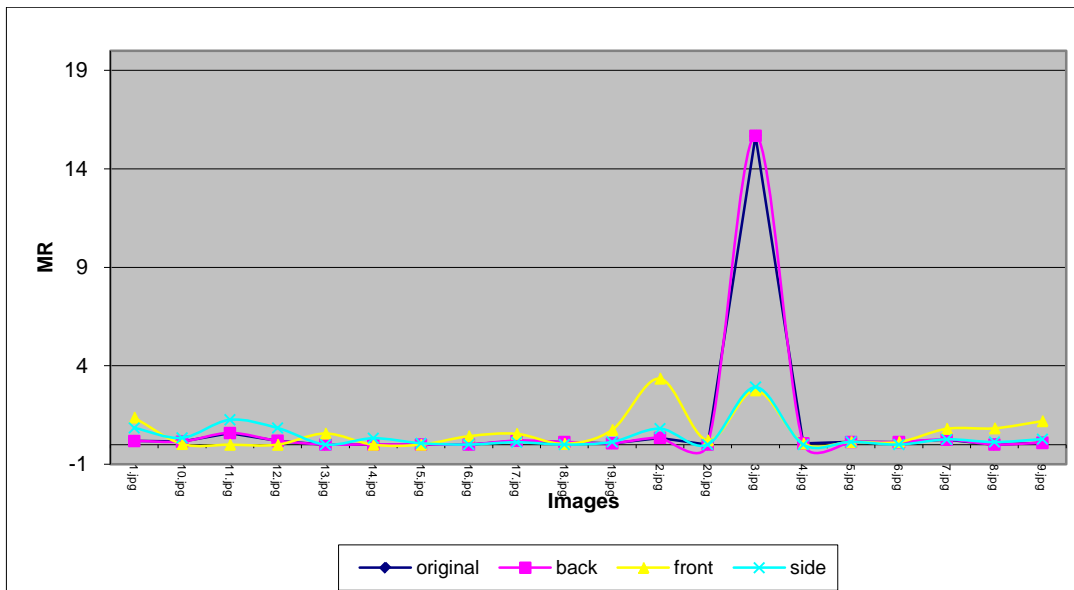


Figure 6-8 Perspective Transformation Response Curve

In Figure 6-8, the response curve shows loss of discriminatory power in the front view. The rest of the perspective views are invariant to the changes.

### 6.6 Scale

Scaling effect is indicative of the shrinking or enlargement of image dimensions, keeping the aspect ratio the same. This is shown in Figure 6-9.



Figure 6-9 Scaling Legend

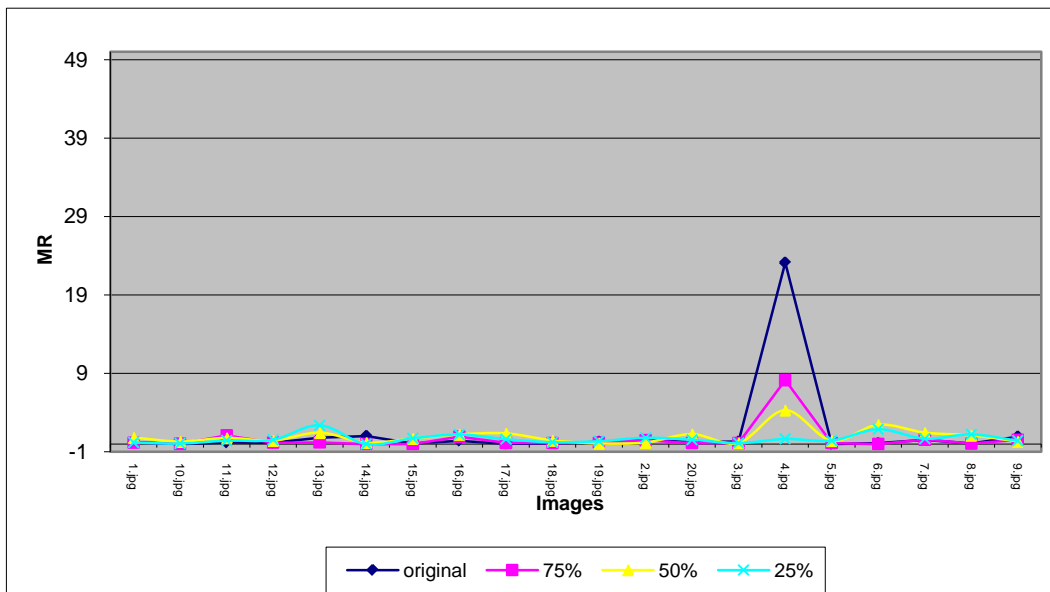


Figure 6-10 Scaling Response Curve



The response curve, illustrated in Figure 6-10, shows that there is a gradual loss of discriminative power as the image shrinks. At 25% of its size, it is fairly impossible to make a distinctive selection for the match. However, for most of the real life applications, the system is robust and effective. In Figure 6-11, we present a consolidated test report for the system testing.

Test Type	Test state 1	Test state 2	Test state 3	Test state 4
Rotation effect	Original	45 Degree	90 Degree	270 Degree
Light Intensity	Original	25 Screen	50 Screen	100 Screen
Perspectice Transformation	Original	Back	Front	Side
Scale	Original	75%	50%	25%

	Positive
	Negative

Legend

**Figure 6-11 Consolidated Test Results**

## CHAPTER - 7 CONCLUSION AND FUTURE DIRECTION

### 7.1 Conclusion

In this thesis, we proposed a scalable system architecture for providing object recognition capabilities on Android based mobile devices. The object recognition algorithm used is based on SURF. We ported a Java based implantation of SURF to the Android platform. The architecture is a client server model, which divides the effort between the mobile device and remote server for optimizing results. The client end extracts SURF features and the server end matches the query feature to the image data stored in the backend database. We proposed and implemented a simple matching algorithm based on nearest neighbor search. We provided a case study of implementing this architecture using an Android platform and the Google app engine framework .The case study consisted of a book cover matching application along with a test framework. We analyzed the matching algorithm using the ROC curve and found the best threshold for the classifier. We further identified the factors that could impact the ideal behavior of the system. Factors such as image rotation, perspective views, scale, and illuminations can cause negative effects on the accuracy of the classifier build in ideal conditions. To study the effects of these factors, we proposed a simple methodology called response curve. With response curve we can visualize the effect of the external factor on the discriminative power of classifier. We used the response curve methodology to analyze

the effects of external factors in our case study. We demonstrated that the system was robust against most of the conditions.

With this thesis we demonstrated that object recognition has come a long way from being esoteric algorithms in research labs and is ready for primetime implementation on mobile devices.

## 7.2 Future Work

The following avenues can be explored for further development of the work covered in this thesis:

- **Implementing object recognition algorithm in Native code.** Currently the SURF implementation used in this thesis is implemented in JAVA; however comparative studies have shown that it is efficient to implement the system in Native code. The feature extraction component in the system is standalone and can be replaced without affecting other components. Future implementation can use Java Native Interface (JNI) to use C or C++ implementation of the SURF algorithm.
- **Automated testing:** During the development of the system, it was found that a large percentage of time is spent on testing the system for ROC and studying the effect of external factors. Currently these tests were done manually using MS Excel to collate the data. An automated system to test these scenarios can significantly improve the time to market future implementations.
- **Feature compression:** The current implementation uses all the features extracted by SURF algorithm for a given image and sends it over the wireless network to the server. Future implementation can use a compression algorithm to quantify

this data. This will help reduce the load on the wireless network while maintaining the accuracy of the system.

APPENDIX A  
SOURCE CODE

/\*This work was derived from Chris Evan's opensurf project and re-licensed as the  
3 clause BSD license with permission of the original author. Thank you Chris!

Copyright (c) 2010, Andrew Stromberg

All rights reserved.

Redistribution and use in source and binary forms, with or without  
modification, are permitted provided that the following conditions are met:

- \* Redistributions of source code must retain the above copyright  
notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright  
notice, this list of conditions and the following disclaimer in the  
documentation and/or other materials provided with the distribution.
- \* Neither Andrew Stromberg nor the  
names of its contributors may be used to endorse or promote products  
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND  
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,  
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE  
DISCLAIMED. IN NO EVENT SHALL Andrew Stromberg BE LIABLE FOR ANY  
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,  
OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND  
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT  
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING  
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
THE POSSIBILITY OF SUCH DAMAGE. \*/

ImageTransformUtils.Java

```
package com.fau.vivek;
```

```
import java.awt.color.ColorSpace;
```

```
import java.io.File;
```

```
import javax.imageio.ImageIO;
```

```
import android.graphics.Bitmap;
```

```
import android.graphics.Color;
```

```
public class ImageTransformUtils {
```

```
    public static float[][] generateIntegralImage(Bitmap mOriginalImage){
```

```

float[][] integralImage = new
float[mOriginalImage.getWidth()][mOriginalImage.getHeight()];

int width = mOriginalImage.getWidth();
int height = mOriginalImage.getHeight();
int pix;
float sum;
for ( int y = 0; y < height; y++ ){
    sum = 0F;
    for ( int x = 0; x < width; x++ ){
//        raster.getPixel(x,y,pixel);
        pix=mOriginalImage.getPixel(x, y);

        float intensity = Math.round((0.299D*Color.red(pix) +
0.587D*Color.green(pix) + 0.114D*Color.blue(pix)))/255F;

        sum += intensity;
        if ( y == 0 ){
            integralImage[x][y] = sum;
        } else {
            integralImage[x][y] = sum + integralImage[x][y-1];
        }
    }
}
}

```

```
        return integralImage;
    }
}
```

SurfCompare.Java

```
package com.fau.vivek;

import java.io.*;

import java.util.ArrayList;

import java.util.List;

import javax.imageio.ImageIO;

import android.app.Activity;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.os.Bundle;

import android.util.Log;

public class SurfCompare extends Activity {

    static final String TAG="SurfComapare";

    private Bitmap image;

    private float mImageAXScale = 0private float mImageAYScale = 0;

    private float mImageBXScale = 0;

    private float mImageBYScale = 0;

    private int mImageAWidth = 0;
```



```

private int mImageAHeight = 0;

private int mImageBWidth = 0;

private int mImageBHeight = 0;

private Surf mSurfA;

private Surf mSurfB;

Bitmap imageA ;

Bitmap imageB ;

private java.util.List<SURFInterestPoint> mAMatchingPoints;

private List<SURFInterestPoint> mBMatchingPoints;

/** Called when the activity is first created. */

@Override

public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    List<Bitmap> imagedata = new ArrayList<Bitmap>();

        imageA =

BitmapFactory.decodeResource(getBaseContext().getResources(),

R.drawable.whitehouse1);

```

```
        imageB =  
        BitmapFactory.decodeResource(getBaseContext().getResources(),  
        R.drawable.whitehouse2);
```

```
        Log.d(TAG,"call SurfCompareloc ");  
        SurfCompareloc(imageA,imageB);  
    }
```

```
public void SurfCompareloc(Bitmap imageA2,Bitmap imageB2){  
    this.image = imageA2;  
    this.imageB = imageB2;  
    mSurfA = new Surf(imageA2);  
    mSurfB = new Surf(imageB2);  
  
    mAMatchingPoints = mSurfA.getMatchingPoints(mSurfB,true);  
    mBMatchingPoints = mSurfB.getMatchingPoints(mSurfA,true);  
    Log.d(TAG,"match poirnts found")
```

## REFERENCES

- [1] D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints,"  
International Journal of Computer Vision, vol. 60, 2004, pp. 91-110.
- [2] H. Bay, A. Ess, T. Tuytelaars, and L.V. Gool, "Surf: Speeded up robust features,"  
Computer Vision and Image Understanding (CVIU), vol. 110, 2008, pp. 346-359.
- [3] C. Evans, "Chris Evans Development OpenSURF."  
Available: <http://www.chrisevansdev.com/computer-vision-opensurf.html>  
[Accessed: June. 12, 2010].
- [4] Davis, "dlib C++ Library." Available: <http://dlib.net/> [Accessed: June. 12, 2010].
- [5] F. Zlatko and T. Reiff, "MASS/Plugin/ProcessorSURF." Available:  
<http://brain.fei.tuke.sk/wiki/index.php/MASS/Plugins/ProcessorSURF> [Accessed:  
June. 12, 2010].
- [6] E. Zatepyakin, "ASSURF in sprit SURF feature extraction library in ActionScript  
for the Adobe Flash Platform." Available: [http://code.google.com/p/in-  
spirit/wiki/ASSURF](http://code.google.com/p/in-spirit/wiki/ASSURF) [Accessed: June. 12, 2010].
- [7] E. Labun, "ImageJ SURF." Available: <http://labun.com/imagej-surf/> [Accessed:  
June. 12, 2010].
- [8] Kitanovski, "JavaSurf Java implementation on speeded up robust features SURF."  
Available: <http://code.google.com/p/javasurf/> [Accessed: June. 12, 2010].

- [9] National Institute of Health, "ImageJ Image Processing and Analysis in Java"  
Available: <http://rsbweb.nih.gov/ij/> [Accessed: June. 12, 2010].
- [10] "OpenCV wiki." Available: <http://opencv.willowgarage.com/wiki/> [Accessed:  
June. 12, 2010].
- [11] Orlinski, "Pan-o-Manic ,tool to automates the creation of control points in  
Hugin." Available: <http://aorlinsk2.free.fr/panomatic/> [Accessed: June. 12, 2010].
- [12] D. Gossow, F. Schmitt and D. Dröge, "Parallel SURF An implemntaion of SURF  
using Pan-o-manic." Available:  
[http://sourceforge.net/apps/mediawiki/parallelsurf/index.php?title=Main\\_Page](http://sourceforge.net/apps/mediawiki/parallelsurf/index.php?title=Main_Page)  
[Accessed: June. 12, 2010].
- [13] P. Strandmark, "SURFmex: A MATLAB SURF interface,," Available:  
<http://www.maths.lth.se/matematiklth/personal/petter/surfmex.php> [Accessed:  
June. 12, 2010].
- [14] P. Furgale and C. H. Tong, "Speeded Up SURF." Available:  
<http://asrl.utias.utoronto.ca/code/gpusurf/> [Accessed: June. 12, 2010].
- [15] N. Cornelis and L. Van Gool, "Fast scale invariant feature detection and matching  
on programmable graphics hardware," 2008 IEEE Computer Society Conference  
on Computer Vision and Pattern Recognition Workshops, 2008, pp. 1-8.
- [16] N. Imamura, and B. McCord, "OpenCV for Android" Available:  
<http://billmccord.github.com/OpenCV-Android/> [Accessed: June. 12, 2010].
- [17] S. Olsson and P. Akesson, "Distributed mobile computer vision and applications  
on the android platform," M.S. thesis, Lund University, Lund, Sweden, 2009.

- [18] Stromberg, "JOpenSURF The SURF descriptor." Available:  
<http://www.stromberglabs.com/jopensurf/> [Accessed: June. 12, 2010].
- [19] Biederman, "Recognition-by-components: A theory of human image understanding.," *Psychological review*, vol. M, 1987, pp. 115-147.
- [20] T. Mathew and A. Pentland, "Face Recognition using eigenfaces," *IEEE Conference on Computer Vision and Pattern Recognition*, 1991.
- [21] H.A. Rowley, S. Baluja, and T. Kanade, "Human Face Detection in Visual Scenes," *Neural Information Processing Systems (NIPS)*, 1995, pp. 875-881.
- [22] F. Fleuret and D. Geman, "Graded Learning for Object Detection," *IEEE Workshop on Statistical and Computational Theories of Vision*, 1999.
- [23] P. Viola and M.J. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, 2004.
- [24] B. Heisele, T. Serre, S. Mukherjee, and T. Poggio, "Feature reduction and hierarchy of classifiers for fast object detection in video images," *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001.
- [25] T. Tuytelaars, "Local Invariant Features: What? How? Why? When?." Available:  
<http://homes.esat.kuleuven.be/~tuytelaa/ECCV06tutorial.html> [Accessed: June. 12, 2010].
- [26] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," *Alvey Vision Conference*, 1988, pp. 147-152.

- [27] R. Wang, "Laplacian of Gaussian." Available:  
<http://fourier.eng.hmc.edu/e161/lectures/gradient/node10.html> [Accessed: June. 12, 2010].
- [28] K. Mikolajczyk and C. Schmid, "Indexing based on scale invariant interest points," International Conference on Computer Vision, 2001, pp. 525-531.
- [29] Eric W. Weisstein, "Hessian. From MathWorld--A Wolfram Web Resource." Available: <http://mathworld.wolfram.com/Hessian.html> [Accessed: June. 12, 2010].
- [30] B. Herbert and G. L. Van, "SURF:Speeded UP Robust Features." Available:  
<http://www.vision.ee.ethz.ch/~surf/index.html> [Accessed: June. 12, 2010].
- [31] K. Mikolajczyk and C. Schmid, "Scale & Affine Invariant Interest Point Detectors," International Journal of Computer Vision, vol. 60, 2004, pp. 63-86.
- [32] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of Affine region detectors. Submitted to International Journal of Computer Vision.
- [33] J. Canny, "A computational approach to edge detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986.
- [34] Y. Ke and R. Sukthankar, "PCA-SIFT : A More Distinctive Representation for Local Image Descriptors," Evaluation, pp. 2-9.
- [35] L. I. Smith, "A Tutorial on Principal Component Analysis." Available:  
[http://www.cs.otago.ac.nz/cosc453/student\\_tutorials/principal\\_components.pdf](http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf)  
[Accessed: June. 12, 2010].

- [36] Johnson and M. Hebert, "Surface Matching for Object Recognition in Complex Three-Dimensional Scenes," *Image and Vision Computing*, vol. 16, pp. 635-651, 1998
- [37] Johnson, "Spin-Images: A Representation for 3-D Surface Matching", doctoral dissertation, The Robotics Institute, Carnegie Mellon Univ, 1997.
- [38] N. Davies, K. Cheverst, A. Dix, and A. Hesse, "Understanding the Role of Image Recognition in Mobile Tour Guides," *Methodology*, 2005.
- [39] D.A. Vigo, F.S. Khan, J.V. Weijer, and T. Gevers, "The Impact of Color on Bag-of-Words based Object Recognition," *Challenge*.
- [40] M. Behrmann and D. Mapelli, "The Role of Color in Object Recognition: Evidence from Visual Agnosia," *Neurocase*, vol. Vol 3, 1997, pp. 237-247.
- [41] L. Juan, "A Comparison of SIFT , PCA-SIFT and SURF," *Image Processing*, pp. 143-152.
- [42] E.D. Michie, D. Spiegelhalter, and C.C. Taylor, "Machine Learning, Neural and Statistical Classification," 1994, p. page 6.
- [43] "Kooaba Query API." Available: <http://www.kooaba.com/developers/query-api/> [Accessed: June. 12, 2010].
- [44] "Eclipse IDE for Java Developers." Available: <http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/galileosr2>. [Accessed: June. 12, 2010].
- [45] "Android ADT Plugin for Eclipse." Available: <http://developer.android.com/sdk/eclipse-adt.html> [Accessed: June. 12, 2010].

- [46] "Android 1.5 Platform." Available: <http://developer.android.com/sdk/android-1.5.html> [Accessed: June. 12, 2010].
- [47] "Android SDK." Available: <http://developer.android.com/sdk/index.html> [Accessed: June. 12, 2010].
- [48] "Android Emulator." Available: <http://developer.android.com/guide/developing/tools/emulator.html> [Accessed: June. 12, 2010].
- [49] "Google App Engine." Available: Retrieved from <http://code.google.com/appengine/> [Accessed: June. 12, 2010].
- [50] "What Is Google App Engine?". Available: <http://code.google.com/appengine/docs/whatisgoogleappengine.html> [Accessed: June. 12, 2010].
- [51] Tomas krajnik, J. Svab, and Libor preucil, "FPGA Based Speeded Up Robust Features," IEEE Transactions on Circuits and Systems, 2009, pp. 10-12.
- [52] P. Viola and M.J. Jones, "Robust Real-Time Face Detection," International Journal of Computer Vision, 2004.
- [53] "Shopsavvy for Android." Available: <http://www.biggu.com/apps/shopsavvy-android/> [Accessed: June. 12, 2010].
- [54] "Kooaba for Android." Available: <http://www.kooaba.com/using-kooaba-on-android/> [Accessed: June. 12, 2010].
- [55] "Layar Reality Browser for Android." Available: <http://www.layar.com/> [Accessed: June. 12, 2010].



- [56] “Augemented Geo Traveller for iPhone.” Available:  
<http://www.augmentedworks.com/> [Accessed: June. 12, 2010].
- [57] “Google Goggles.” Available: <http://www.google.com/mobile/goggles/#text>  
[Accessed: June. 12, 2010].
- [58] T. Fawcett, An introduction to ROC analysis, elsevier, 2005.
- [59] “GIMP-The GNU Image Manipulation Program.” Available:  
<http://www.gimp.org/> [Accessed: June. 12, 2010].