Fall 2012

VISUAL SEARCH APPLICATION FOR ANDROID

Harsh Patel
San Jose State University

THE VISUAL SEARCH APPLICATION FOR ANDROID

A Project Report

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Submitted By:

Harsh Patel

June 2012

The Undersigned Project Committee Approves the Project Titled

VISUAL SEARCH APPLICATION FOR ANDROID

By

Harsh Patel

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Soon Teoh, Department of Computer Science **Date**

Dr. Robert Chun, Department of Computer Science **Date**

Rajan Vyas, Software Engineer at Healthline Networks **Date**

Associate Dean Office of Graduate Studies and Research **Date**

ABSTRACT

The Visual Search Application for Android

The Search Engine has played an important role in information society. Information in the form of text or a visual image is easily available over the internet. The Visual Search Engine aims at helping users locate and rapidly get information about their items of interest. The Visual Search Engine takes input in the form of keywords or visual images and provides information about destinations, artworks, books, wine, branded items, product's catalogs, etc.

The main goal of this project was to create a Visual Search Algorithm that can find matching images based on an input image's features or key-points. We have also focused on researching new techniques and applied them to optimize the proposed algorithm and its integration with Android mobile client.

The project's end product is an Android Mobile Visual Search Application, which will function as follows:

- User clicks a picture of his/her desired item.
- Application recognizes what that item is, using the Visual Search Algorithm.
- Application gets the user's location and based on the user's current location, it will provide a list of stores nearby him/her of the desired item with its price.

ACKNOWLEDGEMENTS

I would like to thank Dr. Teoh for guiding me through this research project and working with me to achieve this. I also thank him for his suggestions and contributions for handling some of the difficulties faced during the course of this project. Without him, this would not have been possible.

TABLE OF CONTENTS

1. Project Overview.....……01

2. Introduction …….…...02

2.1 Image Processing Search Engine……..…….02

2.1.1 Searching Images by Keywords…….……..03

2.1.2 Searching Images by Image…….……03

2.2 Video Processing Search Engine…….....03

2.3 The Mobile Visual Search Engine…….....04

2.3.1 Benefits of Visual Search Applications…….....04

2.4 Capabilities of Visual Search Applications……...…….…..05

2.5 Visual Search Applications in the Industry……...06

3. Image Matching Process……..……....07

3.1 Image Matching Techniques.....07

3.1.1 Matching Image Key-points…….……..…….07

3.1.2 Histogram Image Matching..……...…….08

3.2 Categories of Image Recognition methods based on Image Features……..09

3.2.1 Appearance - based Methods……..09

3.2.2 Feature - based Methods…….…....09

3.3 SIFT (Scale invariant feature transform) algorithm……..10

3.4 SURF (Speeded up Robust Features) algorithm…….....12

3.5 Comparison of Algorithms……....14

3.5.1 Based on Image Variations.....14

3.5.2 Based on Time.....15

3.5.3 Based on Accuracy.....16
4. Mobile Visual Search Application.....17
4.1 Application Flow.....17
4.1.1 Capture image.....18
4.1.2 Find matching image.....18
4.1.3 Fetch meta-data from matching images.....18
4.1.4 List matching items.....19
4.1.5 Find user location, stores list and item prices.....19
4.2 Server Client Architecture.....20
4.2.1 Client side.....21
4.2.2 Server side.....24
4.3 Algorithm.....27
4.4 Approach taken to make Application more efficient.....31
5. Future Improvement.....34
6. Test Case and Results.....35
6.1 Home Screen.....35
6.2 Image Gallery.....36
6.3 Search Prompt 37
6.4 Image Matching Process.....38
6.5 Result Screen.....40
6.6 Fetching Store Screen.....45
6.7 Item Store and Price List Screen.....44
7. Conclusion.....45

8. References…….……..46

LIST OF FIGURES

Figure 1: Visual Search Application…….……..02

Figure 2: Image Key-Points Extraction…….……..08

Figure 3: Time Graph v/s Random Image samples…….....15

Figure 4: Comparing the Accuracy between SURF variants and SIFT.....16

Figure 5: Application Flow Chart…….....17

Figure 6: Server Client Architecture…….....20

Figure 7: Search Time v/s Dataset Images.....32

Figure 8: Product database with JSON response.....33

Figure 9: Home Screenshot…….....35

Figure 10: Image Gallery Screenshot.....36

Figure 11: Search Screenshot.....37

Figure 12: TV Pillow Pets Image Dataset.....38

Figure 13: Image Matching Processing Screenshot…….……..39

Figure 14: Matching Images – I.....40

Figure 15: Matching Images – II.....41

Figure 16: Different Images.....41

Figure 17: Result Screenshot…….....42

Figure 18: Finding Store Process Screenshot…….....43

Figure 19: Store List and Item Price Screenshot…….....44

LIST OF TABLES

Table 1: Comparison between SIFT and SURF……..14

LIST OF CODES

Code 1: Code for binding capturing picture from Android device.....21

Code 2: Code for binding image gallery with Android device.....21

Code 3: Sending image to server.....22

Code 4: Download matching images from server.....23

Code 5: Sending user location to server.....23

Code 6: Retrieving images for dataset.....24

Code 7: Search matching cluster.....25

Code 8: Search matching images.....25

Code 9: Sample code for retrieving store location and item price.....26

Code 10: Indexing all images documents.....29

Code 11: Add image to KMeans.....30

Code 12: Form clusters.....30

1. Project Overview

In introduction, we will explore the different categories of the Visual Search Engines, their benefits, usefulness, and current applications out there in the market. Most of the current Visual Search Engines retrieve information based on an input of keywords, barcode scan or image metadata; as searching based on image is a very complex and a time consuming task. In consideration of machine learning, one notes that the front and back portions of an object are different for the machine. In addition, minor changes in the visuals like: daylight or darkness, rotation, scaling, angel, etc. changes everything.

Our project's focus is to develop a Visual Search Engine that retrieves information from an input of an image. This search application gives the information based on the image-matching algorithm. These kinds of applications are widely popular in the mobile industry, where a user can retrieve information of unknown items by clicking on a picture of it and searching through the Visual Search Engine server.

First, we have to find an optimal image-matching process that is quick and accurate. In section three of the project report, we will focus on the different image matching techniques and different image feature extraction algorithms. We will compare them with each other and select the best algorithm based on its accuracy and speed. That algorithm will then be implemented in our application.

Next, we create a Mobile Visual Search application with our own algorithm by combining it with the image-matching algorithm that we have selected, thereby optimizing the application's algorithm to make it more efficient. In section four, we will implement our algorithm and make the application more accurate, time and space efficient.

2. Introduction

About Visual Search Applications

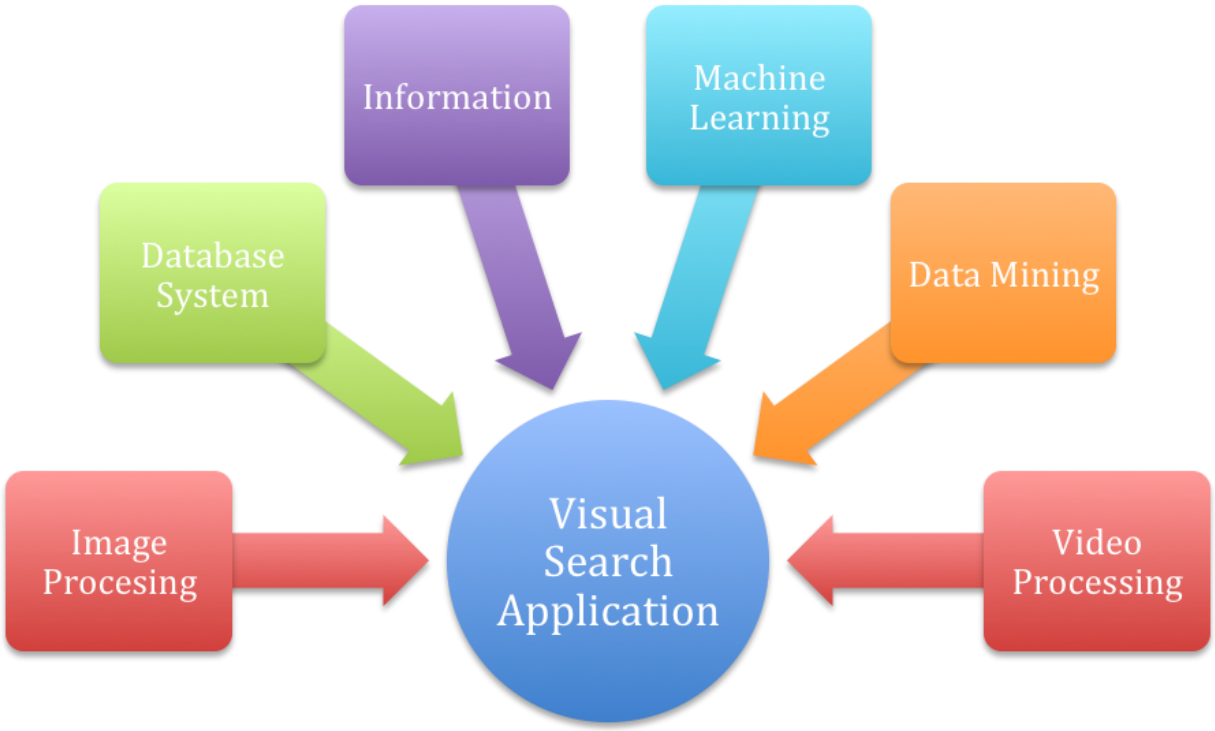


Figure 1: Visual Search Application

Visual Search applications are those applications which provide information in a visual form like an image, video, etc. Such applications can be categorized into two main groups: one is to display an image or video as a result of any query based on keywords; and the other is to recognize visual content such as an image or a video and search for it.

2.1 Image Processing Search Engine

Image Search Engine provides the search results in the form of an image. The input of the search can be either keywords or an image. The results depend on the search criterion, such as metadata, distribution of color, shape, etc. [9]

2.1.1 Searching Images by Keywords

One good example of searching images by keywords is the Google search. The user searches images by entering a keyword as input. The search is done by comparing input keywords with the image's metadata. An image's metadata can consist of an image title, color, information about the image, etc.

2.1.2 Searching Images by Image

Searching images by the Image Search Engine is done by providing input in the form of an image. The matching images are found based on the input given. This process is also known as Content-Based Image Retrieval (CBIR). In this process, the search engine compares key-points or features of the input image with the other images' features in a database. An image's color, shape, texture, etc., are considered during image matching. This approach requires complex computations for finding matching images; and yet, it is sometimes more reliable than searching through an image's metadata, since the metadata of an image can sometimes be incorrect. [9]

2.2 Video Processing Search Engine

A good example of a Video Search Engine is YouTube. It takes input in the form of keywords and searches for videos related to it. Each video has metadata information such as video title, description, type, length, size, etc. [9] Based on the input keyword, videos having matching metadata information are displayed. Some of the metadata information are generated manually while others are auto generated based on the content of a video.

2.3 The Mobile Visual Search Engine

The Mobile Visual Search Engine has a wide scope in the industry. The Mobile Visual Search technique is specially designed to work for mobile devices to retrieve information easily and quickly.

Smartphones have high-resolution cameras, high-speed internet access, fast processing hardware, GPS, etc. [9]. This makes mobile devices the best candidate for Visual Search applications. The user can use a mobile device camera to click pictures of desired items and get information about it. The GPS system also helps to get the location of the user and retrieves information based on that location.

Mobile Visual Search applications search content by providing the search input in the form of an image and process it in the following ways:

- The user clicks a picture of the desired item through a mobile camera.
- An input image is sent to the server for image processing and finding matching images.
- The server does all computations for finding matching images and sends back information to the mobile device.

2.3.1 Benefits of Mobile Visual Search Application

Mobile Visual Search applications help in retrieving information easily and quickly about unknown items. It provides information in the form of an image rather than text. Visual information is much easier to understand than reading text information. It also gives the choice of retrieving information in a graphical, text or voice format. This makes the application useful

in a wide area. Information is described in a very organized and precise way as we can provide very specific information to the user's needs. Considering the new generation smartphones, the usage of such an application with mobile devices is endless. A variety of fields and applications would be created just using mobile devices and the Visual Search algorithm. This will completely bring a new trend of applications into the upcoming market.

2.4 Capabilities of Visual Search Applications

One of the most common implementations of a visual search is the Face Recognition system. It is used by many applications like: Facebook, iPhoto, Picasa, etc. Another interesting use of a Visual Search application is that it can find information about unknown items or objects by taking pictures of them and retrieving information about those items. We can also get information about art or historic collections. This way it can be used as a guide in museums, auctions, etc. The most common and wide usage of a Visual Search can be for searching the product's catalogs (which is our project's focus), finding daily usage items by taking pictures of it and retrieving information about products such as product name, price and location where it is available. A Visual Search can also be used for medical diagnosis. Another lifesaving usefulness of a visual search is to find criminals hiding at crowded places and it can prevent criminal activities.

The Visual Search is also useful in the automobile industry for autonomous driving: detecting stops signs, red lights, pedestrians, vehicles, etc. Augmented reality is also a fresh implementation of a visual search. It combines reality with the digital world. It also helps in finding information about movies, compact disks (CDs), print media, etc.

2.5 Visual Search Applications in the Industry

A glimpse of the most common mobile visual search applications available:

Google Goggles is a Visual Search application created by Google, Inc. It does a Visual Search of famous landmarks, products like books, DVD's, wine, etc. and retrieves information about them. It also uses OCR to retrieve a text from images [13].

OMoby is a Mobile Visual Search application available for iPhone that works by image identification through a visual search. It supports features like searching information, sharing and language learning.

Collectrium uses a Visual Search method for finding artwork. It also allows the user to share art on Facebook or Twitter for managing one's art collections online. It combines a leading back-end for art sellers, art-collectors, artists, galleries, organizations and art fairs with an engaging portal for art buyers.

Shopgate uses Scan & Buy technology. It combines mobile shopping with print advertisement, catalogues, billboards and any kind of printed advertisement by using QR-codes, location based services and image recognition from LTU Technologies.

3. Image Matching Process

In the previous section, we discussed the different categories of the Visual Search Engines, several applications employing Visual Search technique and their scope in the industry. In our application we have focused on a Mobile Visual Search application that takes input as an image and gives information based on it. In this section, we have study different image matching techniques and go through some image recognition methods based on an image's features. We have study image feature extraction methods like SIFT and SURF in detail and find an optimal image matching process that is quick and accurate for our application.

3.1. Image Matching Techniques

3.1.1 Matching Image Key-points

This technique is used for intelligent image matching in which key-points from images are selected and compared with each other. This key-point matching technique is also known as Scale Invariant Feature Transform (SIFT). It selects major points on the image and then compares those points with the input image. If it finds that the points are matching then the algorithm provides images that are matching or similar. This technique is very slow as it takes a long time to compare all the points. However, this image matching technique is popular as it is robust to light, scale, rotation, etc.

This technique was modified later to produce fast key-points image matching technique, which is known as SURF. Our project uses SURF to achieve accurate and efficient results. Since this technique satisfies our need for matching the image quickly and accurately, we have done more research on this technique and have implemented it in our algorithm.

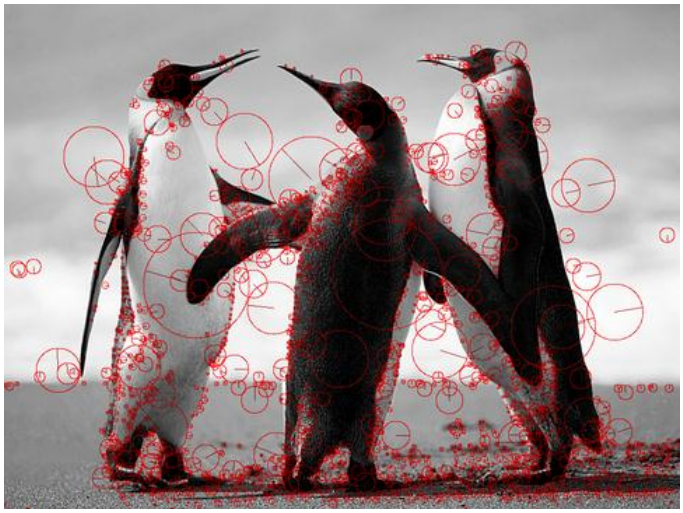


Figure 2: Image Key-Points Extraction [5]

3.1.2 Histogram Image matching

In Histogram technique, the histogram's features are extracted from the input image and then compared with database images to find matching or closest matching ones. This technique is much faster, but less reliable than the key-points matching technique. We can select the histogram's color feature of our choice, mostly RGB are selected as standard colors. Image scaling can also be selected as a histogram feature. This technique is not robust to image rotation, discolor, scale, etc. It will work if an image is cropped to a smaller size. [11]

3.2. Categories of Image Recognition method based on Image Features

Image recognition methods based on image key points or feature methods can be broadly categorized in two types:

3.2.1 Appearance-based methods

Appearance based methods use only elementary image processing. Although these methods are faster and straightforward to implement, they lack in robustness. The image matching fails if images are scaled, rotated, or discolored. Appearance based methods use templates of the objects to identify the image. Objects look different under small changes like shape, angle, color, etc.

A single template is unlikely to succeed reliably. However, it is impossible to represent all the appearances of an object. It is clear that appearance based methods are not good for matching images taken in varying conditions. Scaled, rotated and transformed images will not produce the desired results.

3.2.2 Feature-based methods

Some parts of an image have more information than other parts, those are the ones that are used for smart image matching. Matching images are searched by comparing object features and image features. The feature-based method searches the image by extracting features like surface patches, corners and linear edges from the objects to recognize similar objects.

Time to extracting feature key-points from image is O (n^2m), where n is the number of key-points in each image and m is the number of images in the database [11]. Two common algorithms that detect feature or key-point in images are SIFT and SURF.

3.3 SIFT (Scale invariant feature transform) algorithm

Scale Invariant Feature Transform is an algorithm to detect and describe local features in images. The algorithm was published by David Lowe in 1999. The algorithm extracts the interesting points out of an object in the image, and provides a feature description of an object which can later be compared with the input image. [8]

The SIFT algorithm is divided into four different steps:

Scale space detection

This is the initial stage for computational searches over all scales and image locations. In this step, the difference-of-Gaussian function is used to identify potential interest points, which are invariant to scale and orientation. Instead of the Gaussian function, the difference-of-Gaussian function (DOG) was used to improve the computation speed [1].

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$ [1]

Key-points localization

The Key-points of the objects are first extracted from a set of reference images and stored in the database. In this step, with the use of Hessian matrix, they reject the low contrast points and eliminate the edge response. Points that are more stable are likely to be selected as key-points [8].

Orientation assignment

One or more orientations are assigned to each key-point location based on the local image gradient directions [4]. In this step, in order to get the orientation assignment, the points are sampled within an area around the key-point's and they form an orientation histogram.

Key-point descriptor

The local image gradients are measured at the selected scale in the region around each key-point [4]. The key-point descriptor is 128 dimensions in general.

3.4 SURF (Speeded up Robust Features) algorithm

Speeded Up Robust Feature is a robust local feature detector. It was first presented by Herbert Bay in 2006. It is partly inspired by the SIFT descriptor, but is considered to be much faster and robust than SIFT. [7]

The SURF algorithm is divided into three different steps:

Interest Point Detection

This step computes the discrete Hessian operator on several scales using box filtering. It selects the determinant of the Hessian matrix in scale space, does refinements of the corresponding interest points and stores all the interest points [5].

Local Descriptors Construction

It estimates the orientation of each interest point and computes the vector descriptor corresponding to the scaled and oriented neighbors of all interest points [5].

Image Matching

Image matching is done by finding the matching descriptors between two given images and discarding the matches based on geometric consistency checking (ORSA) [5].

Based on descriptor types SURF has the following variants:

- SURF (DL 64)
- SURF (DL 128)
- U-SURF

SURF and SIFT are somewhat similar. The only difference in them is their ways of detecting features or key-points and describing them.

Following are some features of SURF:

- Fast interest point detection.
- Distinctive interest point description.
- Speeded-up descriptor matching.
- Scale analysis with a constant image size.

3.5 Comparison of Algorithms

Comparisons between SURF and SIFT algorithms:

3.5.1 Based on Image Variations

Comparisons are made based on different images at different time, angle, scale, etc. Both algorithms are rated on a scale of worst, average and best.

Image Variations	SURF	SIFT
Time	Best	Worst
Rotation	Average	Best
Blur	Best	Average
Scale	Average	Best
Affine	Average	Good
Illumination	Best	Worst

Table 1: Comparison between SIFT and SURF

SURF has as good of a performance as SIFT, but it is not that good for rotation and scale. SURF is several times faster than SIFT, which makes SURF a better candidate.

3.5.2 Based on Time

We will compare both algorithms, SIFT and SURF, based on time variants.

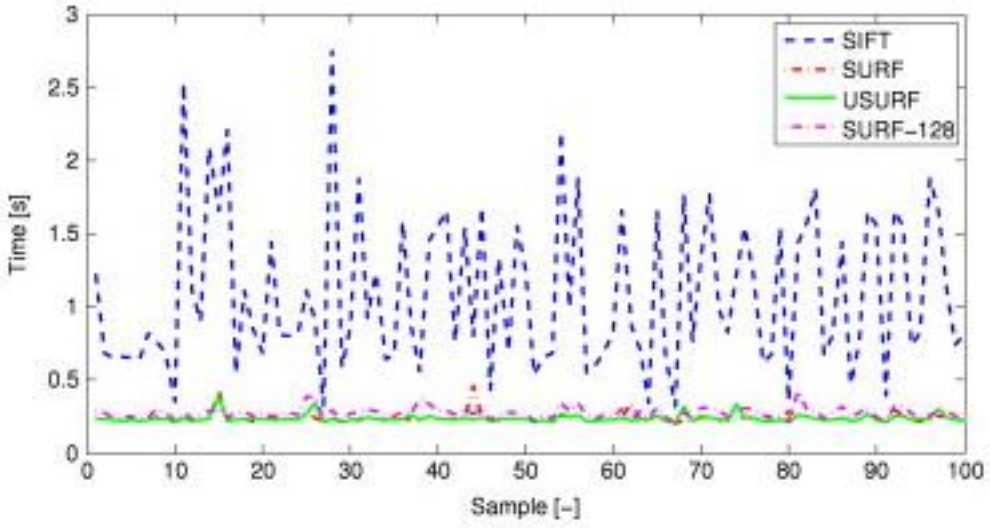


Figure 3: Time Graph v/s Random Image samples [2]

The above graph shows a comparison between different SIFT and SURF's variants (this experiment was conducted by C. Valgren in Licentiate Thesis [2]). It shows the comparison of random images and time required to complete the given task. As shown in the graph, SURF is much quicker than SIFT.

3.5.3 Based on Accuracy

Here we will compare accuracy of SIFT and SURF based on the correct matches that they find (this experiment was conducted by C. Valgren in Licentiate Thesis [2]). We are also giving different levels of threshold 0.6, 0.7 and 0.8 to each algorithm and will get the results based on it.

Algorithm	Total matches			Correct matches			Percentage correct		
	0.6	0.7	0.8	0.6	0.7	0.8	0.6	0.7	0.8
SIFT	279	1222	3558	248	824	1314	89%	67%	37%
SURF	167	598	1431	162	473	715	97%	79%	50%
U-SURF	232	760	1761	229	648	1042	99%	85%	59%
SURF-128	171	531	1191	167	452	708	98%	85%	59%

Figure 4: Comparing the Accuracy between SURF variants and SIFT [2]

Based on three comparisons above made between SURF and SIFT, the SURF algorithm is faster and more accurate for image feature or key-points detector as compared to SIFT. Therefore, we are using the SURF algorithm for detecting local features in images in our project.

4. Mobile Visual Search Application

In our project, we have made our own Visual Search algorithm that uses the SURF algorithm to find image interest points which helps in finding matching images. In this section, we study the architecture of the application, integrated with some additional techniques for enhancing the application's performance, the implied algorithm and the implementation of the application.

4.1 Application Flow

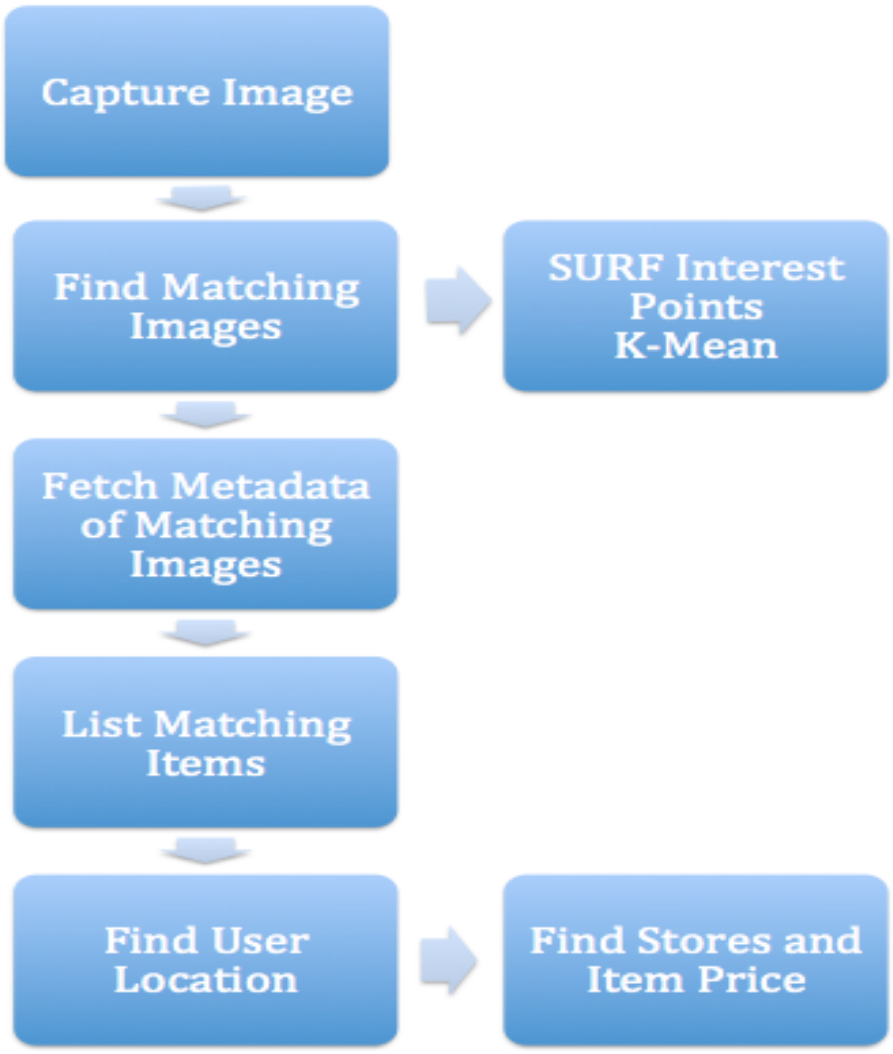


Figure 5: Application Flow Chart

4.1.1 Capture image

In order to search any item, we have to go over to a search engine and type keywords as an item description. Imagine an application that lets a user search by a taking a picture from a mobile phone's camera. It would be much quicker and easier. This application allows the user to take a picture of a desired item through a mobile device. The user clicks on the picture of the item to search for his/her desired item at stores and retrieves its price list. The picture is sent to the server by the application to do all image matching processing on the server side.

4.1.2 Find matching image

It is not necessary that the image captured is in a proper format. The image could be blurred, distorted, upside-down, sideways, etc. But, this is an additional overhead and it could slow down the searching process. So, to get faster search results we need an algorithm which is invariant to image transformations, scaling, rotations and discoloration. We use the SURF interest points and the K-Means algorithm to find the matching images. SURF is fast and robust to image scaling, rotation and blur. This whole process is done on the server side.

4.1.3 Fetch meta-data from matching images

Once we get the matching images that are similar to our input image, the next step is to get information about those images. Each image will have the metadata associated with them and the metadata will have information about those images.

For example, an image of a movie DVD can have meta-data such as the movie title, director name, category, etc. We send the matching images and their information back to the mobile device.

4.1.4 List Matching Items

The user will now have a list of 10 matching items as a result of an input image. The user can select any item from it.

4.1.5 Find user location, stores list and item prices

Now, once the user selects his desired item from 10 matching items, based on a user's current location, the application retrieves the nearby stores' information. The application will check if the item is available in those stores or not. If the item is available, it will then list the stores' addresses with the item price tag.

4.2 Server Client Architecture

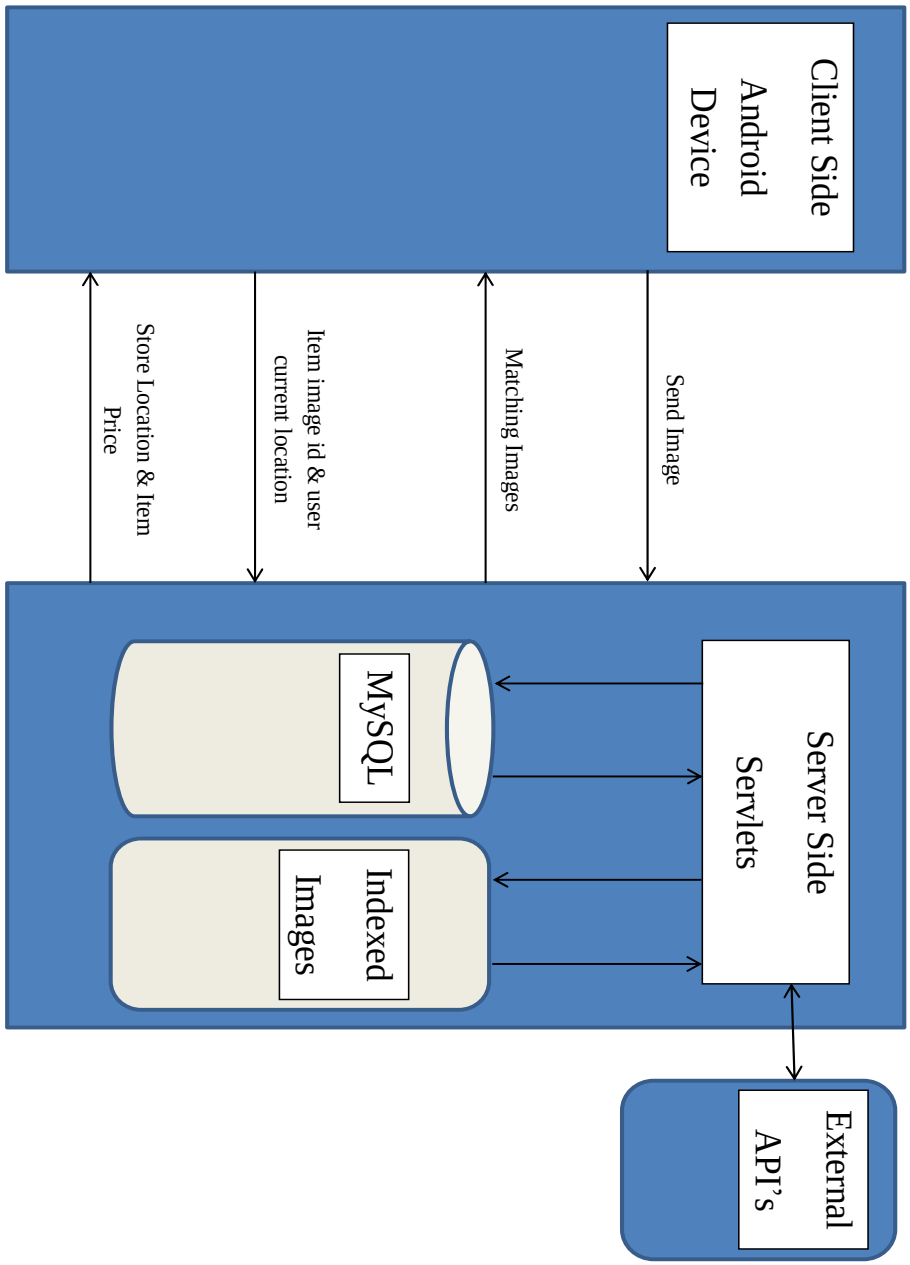


Figure 6: Server Client Architecture

4.2.1 Client Side

The Client side of the application can be any Android mobile device. The user can click the picture of his/her desired product or he/she can select a picture from an image gallery. As shown in below code snippets, we bind the browsing image gallery and capture the image to the Android client activity.

```
private void bindCaptureButton() {
    Button captureImage = (Button) findViewById(R.id.capture_image);
    captureImage.setOnClickListener(new OnClickListener() {

        public void onClick(View v) {
            Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            fileUri = getOutputMediaFileUri(MEDIA_TYPE_IMAGE);
            intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);
            startActivityForResult(intent,
                    CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
            }
    });
}
```

Code 1: Code for binding capturing picture from an Android device

```
private void bindBrowseButton() {
    Button browseImage = (Button) findViewById(R.id.image_gallery);
    browseImage.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
             Intent intent = new Intent();
             intent.setType("image/*");
             intent.setAction(Intent.ACTION_GET_CONTENT);
             startActivityForResult(Intent.createChooser(intent,
                     "Select Picture"),SELECT_IMAGE_REQUEST_CODE);
        }
    });
}
```

Code 2: Code for binding an image gallery with an Android device

Once the user selects the image he/she wants to search, the image is sent to the server. Then the server does all the computations for finding matching images and their

information. The image is sent in MIME format and the image sending process is done in the background while on the Android client UI shows the processing status to the user.

```
HttpClient httpclient = new DefaultHttpClient();
File file = new File(this.url);
HttpPost httppost = new HttpPost("http://10.0.2.2:8080/thesis-server/Process");

MultipartEntity mpEntity = new MultipartEntity();
FileEntity entity = new FileEntity(file, "multipart/form-data");

ContentBody cbFile = new FileBody(file, "application/octet-stream");
mpEntity.addPart("userfile", cbFile);
httppost.setEntity(mpEntity);

HttpResponse response = httpclient.execute(httppost);
HttpEntity resEntity = response.getEntity();

if (resEntity != null) {
    output = EntityUtils.toString(resEntity);
}
if (resEntity != null) {
    resEntity.consumeContent();
}
httpclient.getConnectionManager().shutdown();
```

Code 3: Sending the image to the server

The server sends the top 10 matching images and images' information (such as image description, image id, etc.), back to the client's device in JSON. In the download method, we check the cache in order to see if we already have the image and its information. If we don't, the bitmap value will be null and we will download the image and its information from the server. If we already have an image and its information from a previous search, then we will use it from the cache. This image caching technique helps to make the application more time efficient.

```
public void download(String url, ImageView imageView) {
    resetPurgeTimer();
    Bitmap bitmap = getBitmapFromCache(url);

    if (bitmap == null) {
        imageView.setImageResource(stub_id);
        forceDownload(url, imageView);
    } else {
        cancelPotentialDownload(url, imageView);
        imageView.setImageBitmap(bitmap);
    }
}
```

Code 4: Download matching images from server

Once we get the results from the server, the user has the choice to select an item out of the 10 results received. Once the user selects the item he wants, the item's image id and the user's current location is sent to the server. Based on the image id and the user location received from the client's side, the server finds the nearest stores to the user's current location and the price of the item that the user has selected.

```
HttpClient httpClient = new DefaultHttpClient();
HttpPost postRequest = new HttpPost(
        "http://10.0.2.2:8080/thesis-server/Stores");
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("id", this.id));
params.add(new BasicNameValuePair("longitude", this.longitude));
params.add(new BasicNameValuePair("latitude", this.latitude));
UrlEncodedFormEntity ent = new UrlEncodedFormEntity(params,HTTP.UTF_8);
postRequest.setEntity(ent);

HttpResponse response = httpClient.execute(postRequest);
```

Code 5: Sending user location to the server

4.2.2 Server Side

The Server side of the application runs Servlets and MySQL database. We use the BestBuy API to get our images for our dataset. We sent JSON request to the BestBuy API and get all the images in response. We save the image id, image URL, description, product id, page, etc.

```
if(apiResponse.has("products")){
    JSONArray products = apiResponse.getJSONArray("products");
    numberOfProducts = products.length();
    for(int i=0; i < numberOfProducts; i++){
        JSONObject product = products.getJSONObject(i);

        if(product.getInt("productId") != 0){
            Images image = new Images();
            image.setApi("bestbuy");
            image.setPage(pageNumber);
            image.setProduct_id(product.getString("productId"));
            image.setResponse(product.toString());
```

Code 6: Retrieving images for dataset

Once we have all the images, the application has to go through a machine learning process. For machine learning, we used the open source Lucene LIRE library for indexing all the images in the dataset. While indexing, we extracted the SURF features for all the images. Then we clustered the images based on the common features they have. Once we finished with the machine learning, we are ready for the image searching. The algorithm is explained in detail in section 4.3.

The server receives an input image from the client side and it retrieves the SURF features from the image received from a client. Then the server finds the matching cluster that has the same features as the input image.

```
double d = clusters[0].getDistance(f);
double tmp;
int result = 0;
for (int i = 1; i < clusters.length; i++) {
    tmp = clusters[i].getDistance(f);
    if (tmp < d) {
        d = tmp;
        result = i;
    }
}
return result;
```

Code 7: Search matching cluster

After finding the matching cluster, it searches for a matching image within that cluster. As shown below in the code portion of our search method, it returns an array-list of matching images' filenames in string format. We are using ImageSearcher, which runs a query to get all the matching images. It searches for images using input image features.

```
public ArrayList<String> search(int numHitsFiltering, Document query,
        String min_id, String max_id) throws IOException {
    System.out.println(this.indexPath);
    BooleanQuery.setMaxClauseCount(Integer.MAX_VALUE);
    IndexReader reader = IndexReader.open(FSDirectory.open(new File(this.indexPath)), true);

    String visualFeatures = query.getValues(DocumentBuilder.FIELD_NAME_SURF)[0];
    System.out.println("visualFeatures = " + visualFeatures);

    QueryParser qp = new QueryParser(Version.LUCENE_30, DocumentBuilder.FIELD_NAME_SURF, new
            WhitespaceAnalyzer(LuceneUtils.LUCENE_VERSION));
    IndexSearcher isearcher = new IndexSearcher(reader);
```

Code 8: Search matching images

The server sends 10 matching images and their metadata information to the client in JSON format. The user selects the item out of 10 items that client side has received from the server. The server receives the image id of the selected item and the user's current location. BBY-Open API is used to get the nearest stores from the user's current location. RESTY API is used to get the prices of the items in the stores. The server sends the stores' locations and the

item's prices to the client side in JSON format. Here is some sample code that shows the stores and the item's price according to location's longitude and latitude given.

```
for(Store store : storeResponse.list()) {
    out.write(
            store.getName() + " (" + store.getDistance() + " miles)"
    );
    for(Product product : store.getProducts()) {
        if(product.hasInStoreAvailability()) {
            out.write(product.getName());
            out.write(
                    "Available for $" + product.getSalePrice()
            );
        }
    }
```

Code 9: Sample code for retrieving store location and item price

4.3. Algorithm

Our algorithm forms clusters based on images' features or key-points. First, all the images in the dataset are indexed and the features or key-points of all the images are extracted. Then we divide all the images having common features or key-points in the form of clusters; this way we will have different clusters based on different features. We quickly process indexing by searching for the matching images in clusters. So when we have a new image, we get the features of the input image, find a matching cluster based on the features and then find the matching images.

Pseudo-code of Algorithm:

```
get all product P's information in the database D from BestBuy
for loop to go through all product P in database D
        get the image I for product P
        if the image I exits
                create a document D for  the image I
                get SURF features/key-points of  the image I
                add numeric field to document D
                add index field to document D
                index-writer IW add the document D to the index
Once the indexing is done, clustering will start
index-reader IR opens Index files
```

cluster C sets cluster path

start clustering

K-Means K, set numbers of clusters

Iterate through each document D

add an image file feature or key-points in K

find the distance between features or key-points

set threshold T for clustering

create and form clusters of all the documents D having similar features or key-points

index clusters-id

Once we get all the images in our dataset, we need to do machine learning for our application. As shown in below code snippet we go through each image and index all the images. We use Document Builder instance to create Documents. We create Documents by createDocument(imageFile, filename). Here we pass the image file and filename in the createDocument method. It extracts the features/key-points from the image file and stores in it in a Document with a reference to the filename; which is then used to retrieve the image information if we find matching features with the input image. We create Documents for all the images and add all documents to the index.

```
System.out.println("Started indexing");
int numberOfProducts = 0;
String [] imageKeys = keys.split(",");
ImageManager iManager = new ImageManager();
List<Images> lstImages = new ArrayList<Images>();

try{
    LireUtils lireutil = new LireUtils(indexPath, create);
    lstImages = iManager.getProducts(recordNumber, maxRecords);
    numberOfProducts = lstImages.size();

    for(int i=0; i < numberOfProducts; i++){
        Images productObj = lstImages.get(i);
        JSONObject product = new JSONObject(productObj.getResponse());

        if(product.getInt("productId") != 0){
            for(int imageKeyctr = 0; imageKeyctr < imageKeys.length; imageKeyctr++){
                String key = imageKeys[imageKeyctr];
                if(product.has(key) && product.getString(key) != "null"){
                    String imageUrl = product.getString(key);
                    if(!imageUrl.contains("/nonsku/")){

                        String filename = FilenameUtils.getName(imageUrl);
                        String filePath = dataPath + product.getString("type") + "/" + filename;
                        File imageFile =  new File(filePath);
                        if(imageFile.exists()){
                            Document doc = lireutil.createDocument(imageFile, filename);
                            doc = lireutil.addNumericField(doc, "database_id",
                                    Long.parseLong(String.valueOf(productObj.getId())));
                            doc = lireutil.addField(doc, "current_image", imageUrl,false);
                            lireutil.addDocument(doc);
```

Code 10: Indexing all images in Documents

Once we are done with the indexing of all the images, we can start clustering. We also keep track of new images added to the dataset. So after some period of time, we do re-indexing of only newly added images and add them to our index.

Now for clustering, we set the KMeans numbers of clusters to be formed when we create an object of KMeans. Then we go through each Document and get their SURF features.

We use the addImage method in which we give file and image features to KMeans. KMeans then calculates the distance between features or key-points of all images in the index.

```
HashSet<Integer> docIDs = selectVocabularyDocs();
KMeans k = new ParallelKMeans(numClusters);
// fill the KMeans object:
LinkedList<Surf> features;
int ctr = 0;
for (Iterator<Integer> iterator = docIDs.iterator(); iterator.hasNext(); ) {
    int nextDoc = iterator.next();
    if (!reader.isDeleted(nextDoc)) {
        Document d = reader.document(nextDoc);
        features = new LinkedList<Surf>();
        byte[][] binaryValues = d.getBinaryValues(localFeatureFieldName);
        String file = d.getValues(DocumentBuilder.FIELD_NAME_IDENTIFIER)[0];
        System.out.println((ctr++ )+ ": " + file);
        for (int j = 0; j < binaryValues.length; j++) {
            LireFeature f = getFeatureInstance();
            f.setByteArrayRepresentation(binaryValues[j]);
            features.add((Surf) f);
        }
        k.addImage(file, features);
    }
}
```

Code 11: Add image to K-Means

After adding all images to K-Means, it forms clusters based on images having common features/key-points.

```
clusters = k.getClusters();
Cluster.writeClusters(clusters, clusterFile);
time = System.currentTimeMillis();
int[] tmpHist = new int[numClusters];
IndexWriter iw = LuceneUtils.createIndexWriter(reader.directory(), false,
        LuceneUtils.AnalyzerType.WhitespaceAnalyzer, 256d);
if (pm !=null) {
    pm.setProgress(50);
    pm.setNote("Clustering finished");
}
```

Code 12: Form clusters

4.4. Approach taken to make application more efficient

Now, we will focus on improving the application. We have applied three techniques in order to increase the application's efficiency:

1. **Feature Clustering & Indexing**

   This approach helps the application to give accurate results. We indexed all the images to search quickly and while indexing, we also extracted image features or key-points based on SURF interest points. The extracted SURF interest points of each image were fed into the K-Means clustering algorithm to form clusters based on the SURF interest points of images. The images having similar features or key-points are clustered together. Those clusters make searching much faster compared to just indexing.

   In the indexing process, we have to go linearly to find matching images and it takes a longer time with large dataset of images. To search for matching images, we first find the cluster that has features similar to the input image and then we search within that cluster to get our top 10 matching images. The example in Section 14 shows how accurate results our application gives, as it finds all the TV pillow matches to input the image features or key-points.

2. **Multi-Threading:**

   This approach helps application to give results fast, when we get an input from an image that we are unfamiliar with, we have to search the entire cluster to find the matching images. To optimize this process of the search results we are using multi-threading. In the

multi-threading process each thread will examine a fixed set of images from the cluster. For example: Suppose if X is the total number of images in a cluster and each thread has to examine Y number of images, then the total number of threads spawned N, will be: N = X/Y.

This would produce faster results as compared to searching each image linearly. As shown in below graph, the black line of graph represents a search without multi-threading. Thereby, as the number of images in the dataset increase the time taken to search also increases significantly. Application time efficiency is around 3-5 seconds, with multi-threading on the server side.

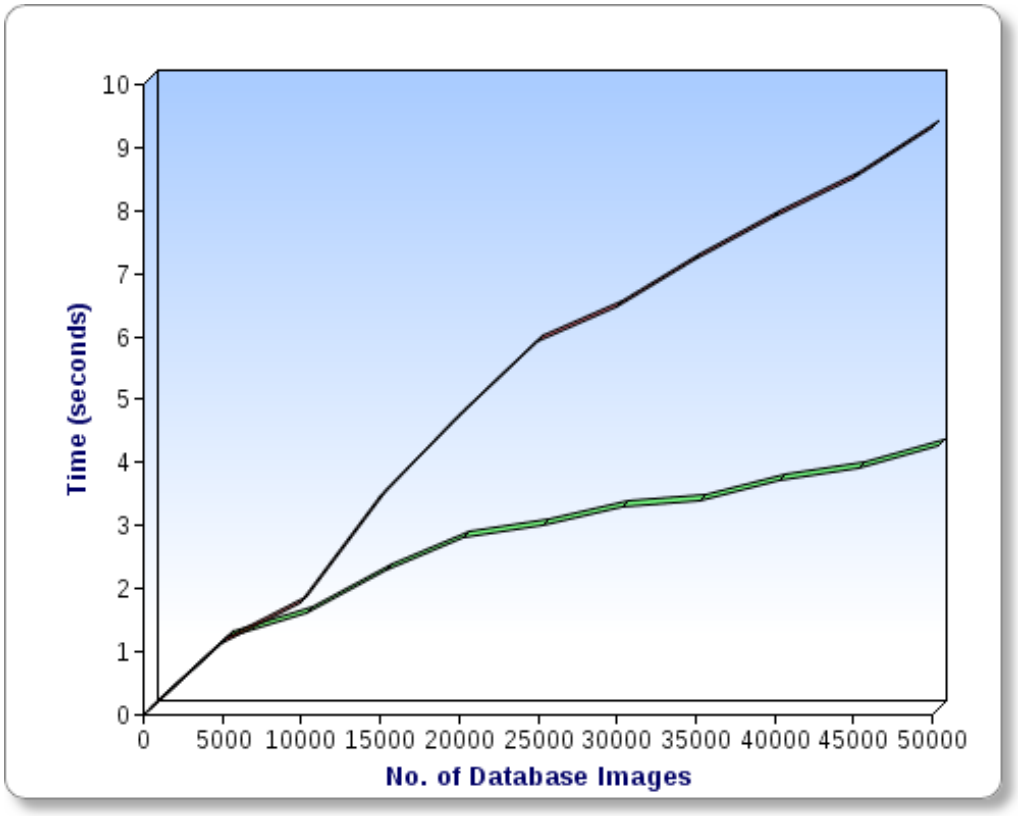


Figure 7: Search Time v/s Dataset Images

3. Disk Space Efficiency:

This approach helps the application in saving disk space and file transactions for getting images from disk. Once we extracted the features or key-points out of all the images in the dataset, the clusters were formed. We didn't need all the images in the dataset since it created extra storage needs. So, to make more storage space available on the disk, we deleted them and stored the URL of all the images and image information (like: image location link, image title, image id, etc.) into MySQL database. So that when we need to retrieve images, we will make JSON request to API through the image's URL. This will save a lot of disk space on the server. Also, it will reduce the overhead of getting an image file from the disk through code. As shown in the below screenshot of the database, we are stored JSON response on image information such as URL, product id, page no., etc.

page	product_id	response
2	1218289372187	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2011-01-09T01:32:09","addToCartUrl":"http://www.bestbuy.com/sit
2	1218168618285	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2010-03-14T01:01:07","addToCartUrl":"http://www.bestbuy.com/sit
2	1218258077278	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2010-10-31T01:34:49","addToCartUrl":"http://www.bestbuy.com/sit
2	1051384289132	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2010-07-26T19:42:51","addToCartUrl":"http://www.bestbuy.com/sit
2	1218167169065	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2010-03-14T01:01:07","addToCartUrl":"http://www.bestbuy.com/sit
2	1218072892585	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2009-06-07T01:32:43","addToCartUrl":"http://www.bestbuy.com/sit
2	1186005358126	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2007-10-21T00:31:23","addToCartUrl":"http://www.bestbuy.com/sit
2	1186005356092	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2009-10-03T13:17:26","addToCartUrl":"http://www.bestbuy.com/sit
3	1186005355962	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2009-09-19T01:55:03","addToCartUrl":"http://www.bestbuy.com/sit
3	1186005356754	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2007-10-21T00:31:23","addToCartUrl":"http://www.bestbuy.com/sit
3	1218210316674	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2010-08-22T01:33:14","addToCartUrl":"http://www.bestbuy.com/sit
3	1218193835217	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2010-09-05T01:31:00","addToCartUrl":"http://www.bestbuy.com/sit
3	1186005357079	{"accessories":[],"accessoriesImage":null,"active":true,"activeUpdateDate":"2009-02-18T01:46:02","addToCartUrl":"http://www.bestbuy.com/sit

Figure 8: Product database with JSON response

5. Future Improvement

In the future we can make the following improvements in the application:

We can implement the Optical Character Recognizing (OCR) through which we can extract a text out of the image, if the image has any text. Also we can perform keyword searches or search for that keyword with image metadata and find matching images.

We can allow the user to request more matching images from the server. If the user does not find the results useful enough, he/she can request for more results or can provide his/her own keywords to find the product. This will help in the machine learning process of the application.

We can introduce the 'User Behavior Study' feature to make the application study the behavior of the user and understand his/her preferences. Hence, each time the user initializes a search, it is of the utmost relevance to the user's preferences.

We can include the image filtering process to generate better search results. In this process, if the user uploads a blurry or improper image and it is hard to get feature or key-points in an image, then server should request another image from the user.

Results filtering can also be done by selecting the image that has the highest matching local features or key-points with the input image, and considers that image as the result. It shows content relevant to that image category only.

6. Test Case and Results

6.1 Home Screen

In the home screen we can either select an image from an image gallery or we can use a capture button to start a device camera and take a picture.

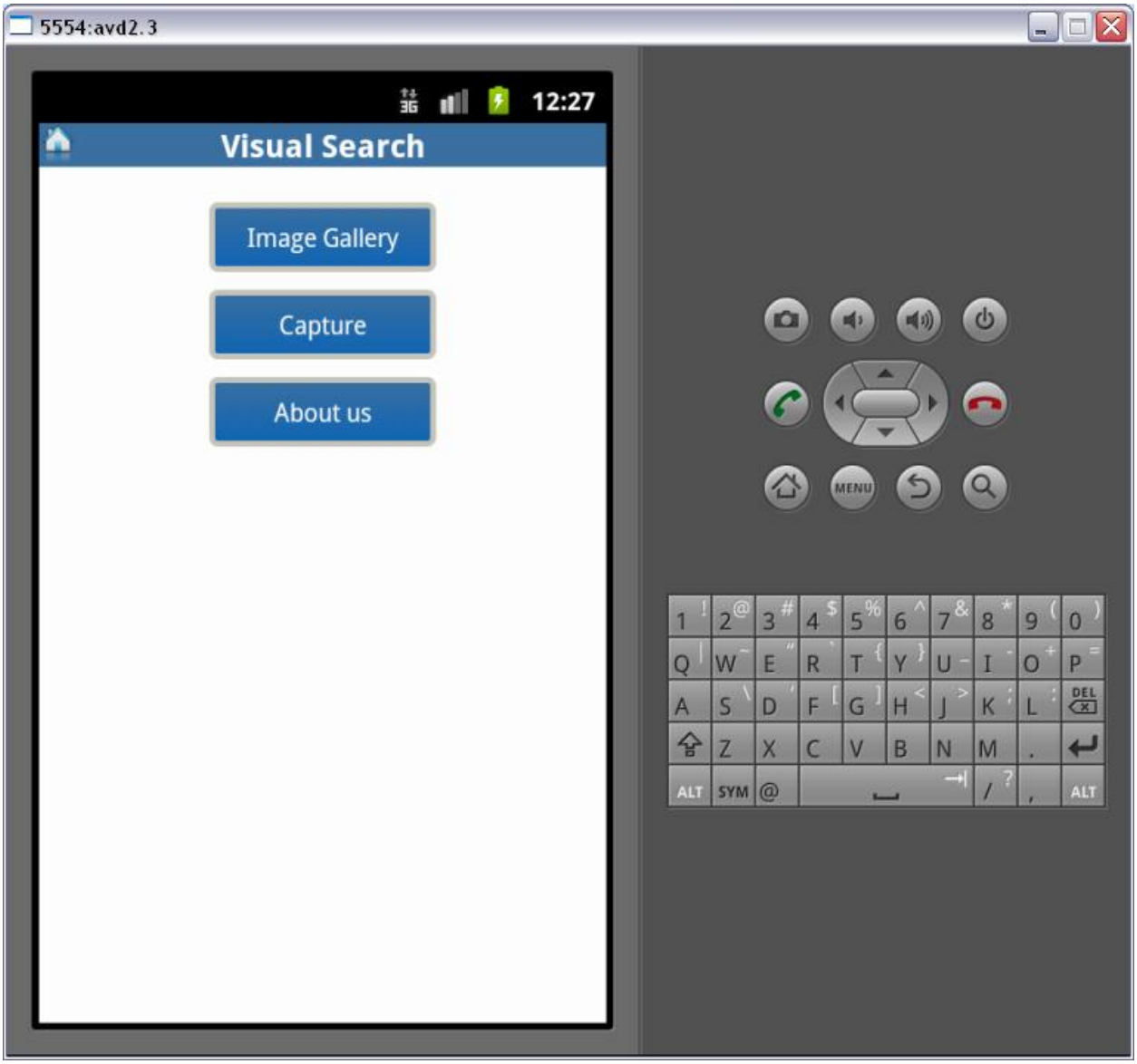
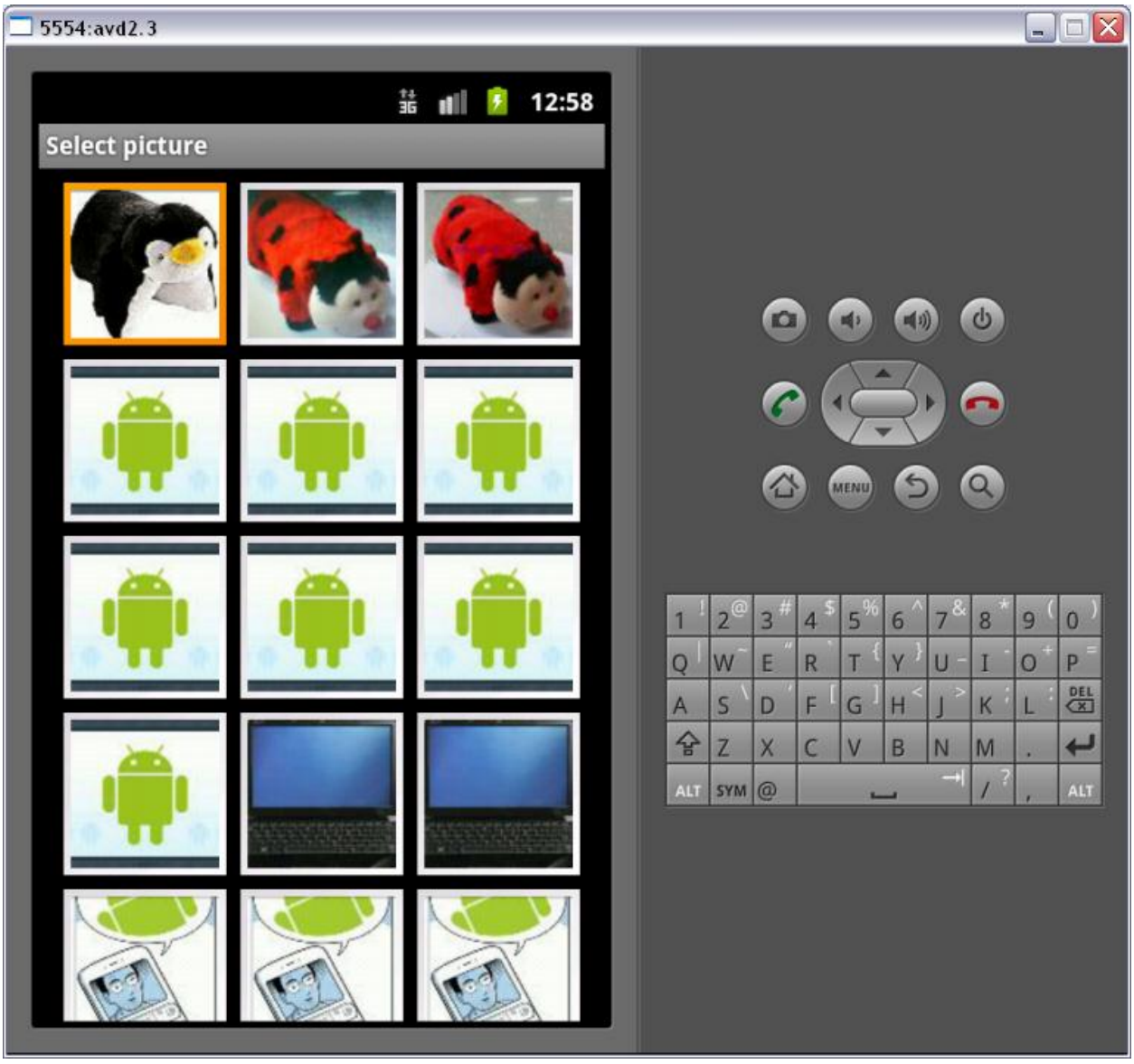


Figure 9: Home Screenshot

6.2 Image Gallery

Allows the user to select any image from the image gallery



Figur10: Image Gallery Screenshot

6.3 Search Prompt

Prompts the user for confirmation of whether he wants to search using the current image or select another image. This helps when the user uses the capture image option and the image is not captured properly. Here we are searching for TV pillow pets.

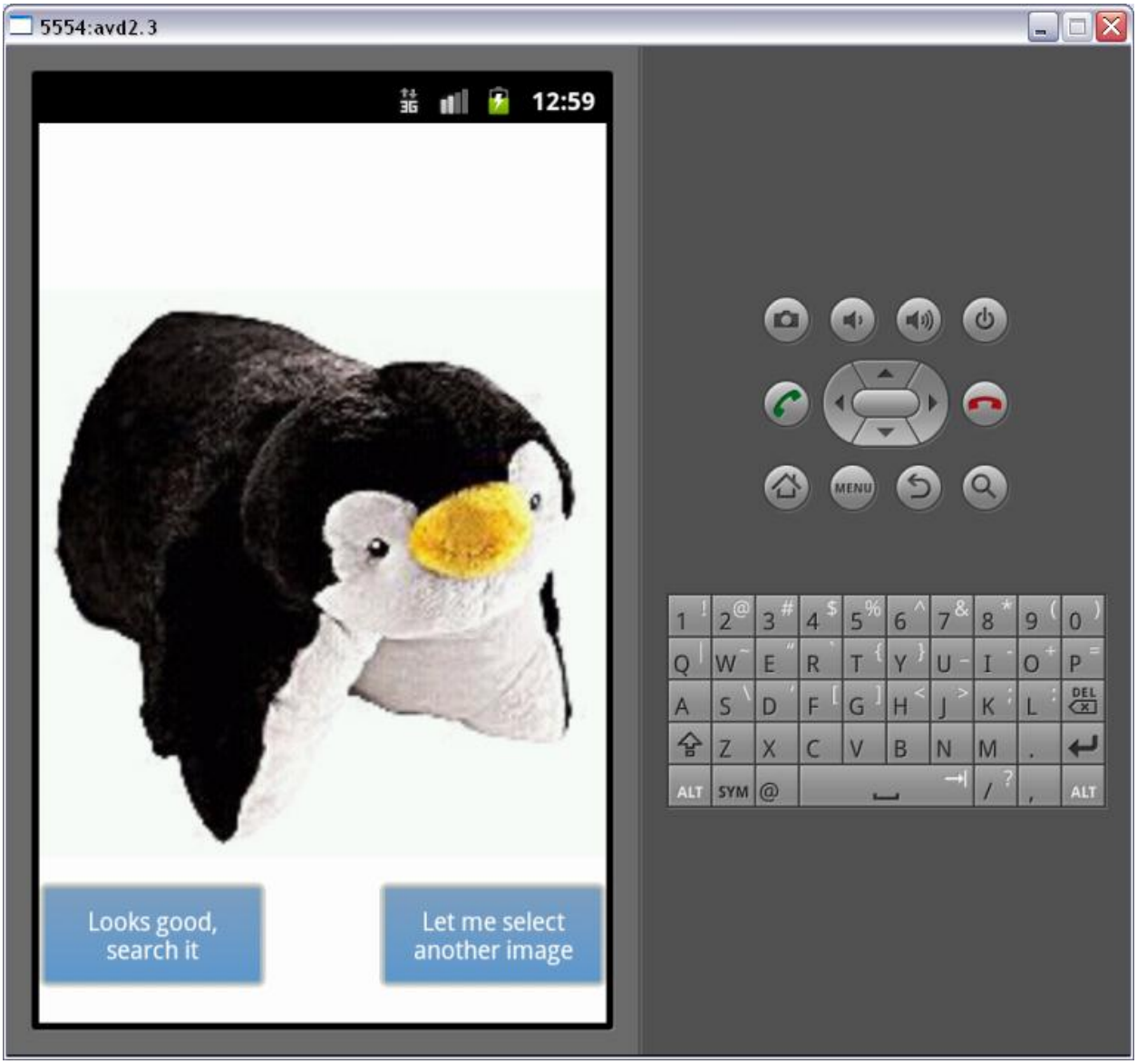


Figure 11: Search Screenshot

6.4 Image Matching Process

The Image matching process is done on the server side in which we are finding matching images based on SURF interest key-points. The images are neither tagged, nor do they have any information on them. We store the product information in the database. So, whenever we find matching images from a particular image_id, we find the product_id and we get the product information based on that.

The Image Dataset has 50,000 images currently. The Clusters and Indexes are stored on a disk. Our dataset consist of 5 TV Pillow pets set images as shown below in row two:

Figure 12: TV Pillow Pets Image Dataset

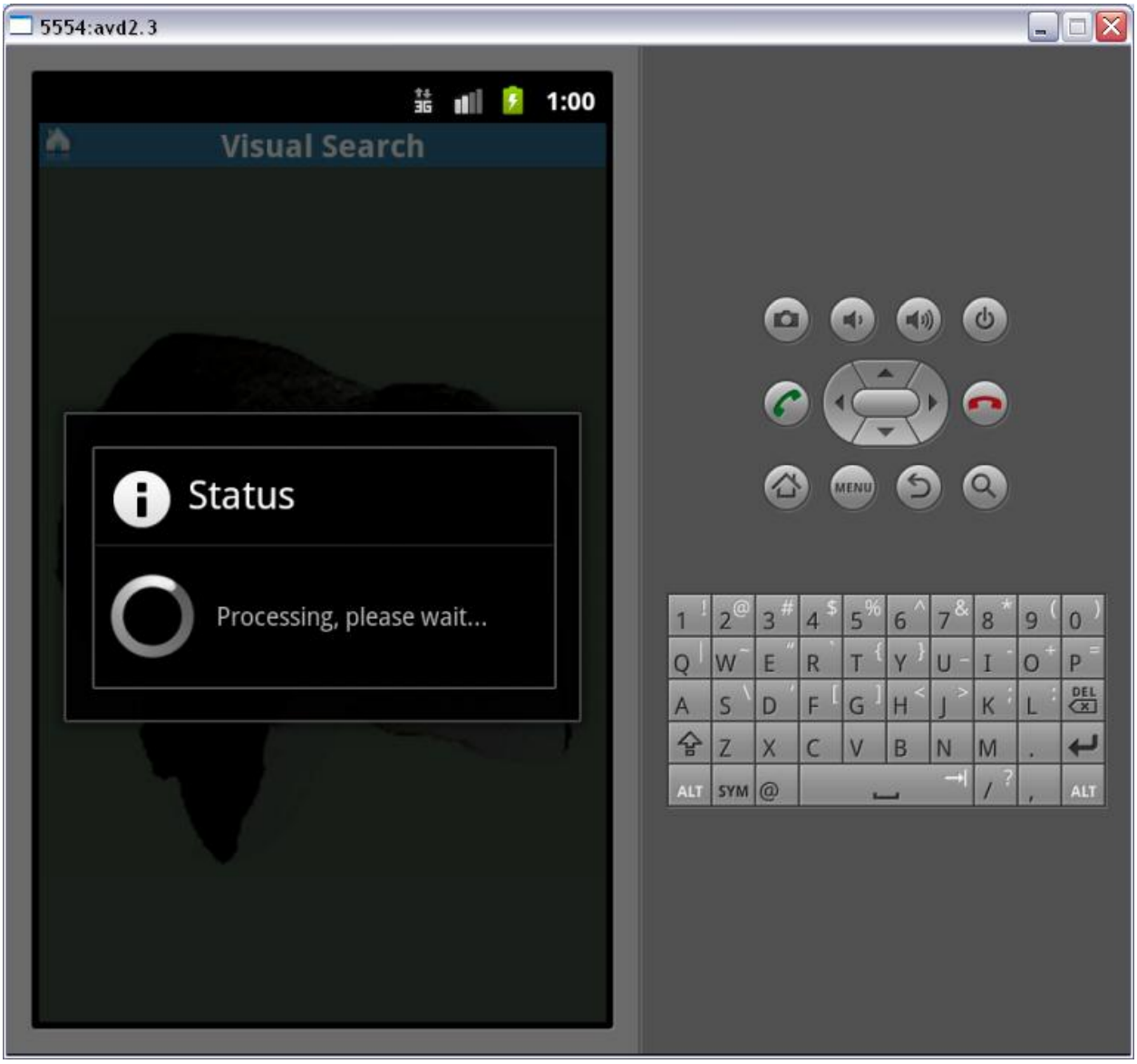


Figure 13: Image Matching Processing Screenshot

6.5 Result Screen

Our algorithm finds matching images based on the comparison of the input image's interest key-points with dataset images interest key-points. As shown in results, we got most of the TV Pillow pets except the blue dolphin, as we can see interest points of the blue dolphin does not match interest points of the input image in figure 15.

Figure 14: Matching Images - I

(As shown in above Figure 13, the left image is the input image and the right one is the matching image having similar feature interest points.)

Figure 15: Matching Images - II

(As shown in above Figure 14, the left image is the input image and the right one is a matching image having similar feature interest points.)

Figure 16: Different Images

(As shown above in Figure 15, the left image is the input image and as the right image does not have matching feature interest points it's not included in results.)

The result set includes other images since on server side we set the threshold to return minimun 10 images. So, once we have 4 TV pillow pets in results, the other 6 results are filled by other images, which match features or key-points with the input image.

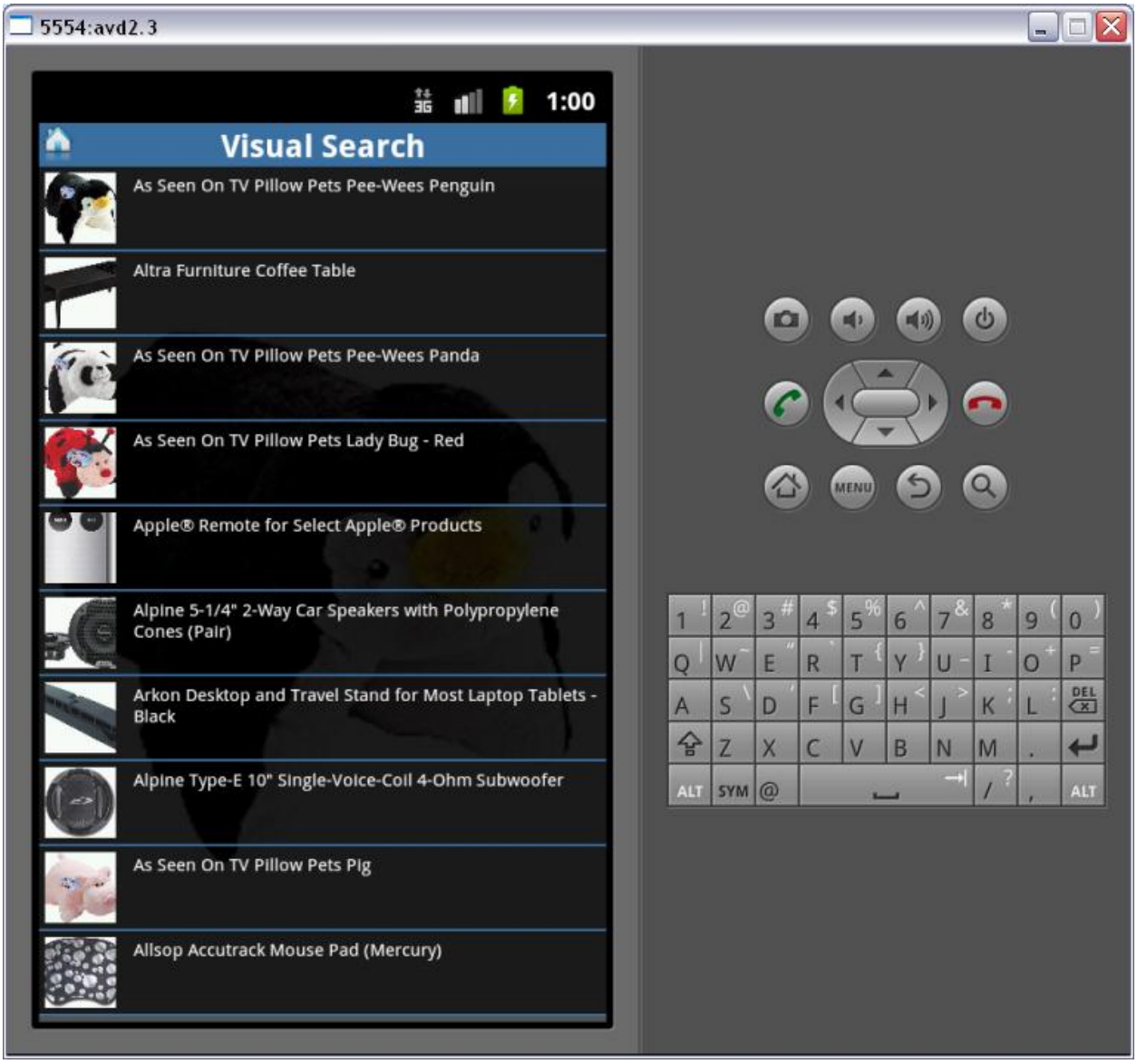


Figure 17: Result Screenshot

6.6 Fetching Store Screen

The Fetching Store Screen gets stores nearest to the user's current location and the price of the item that the user has selected from the server.

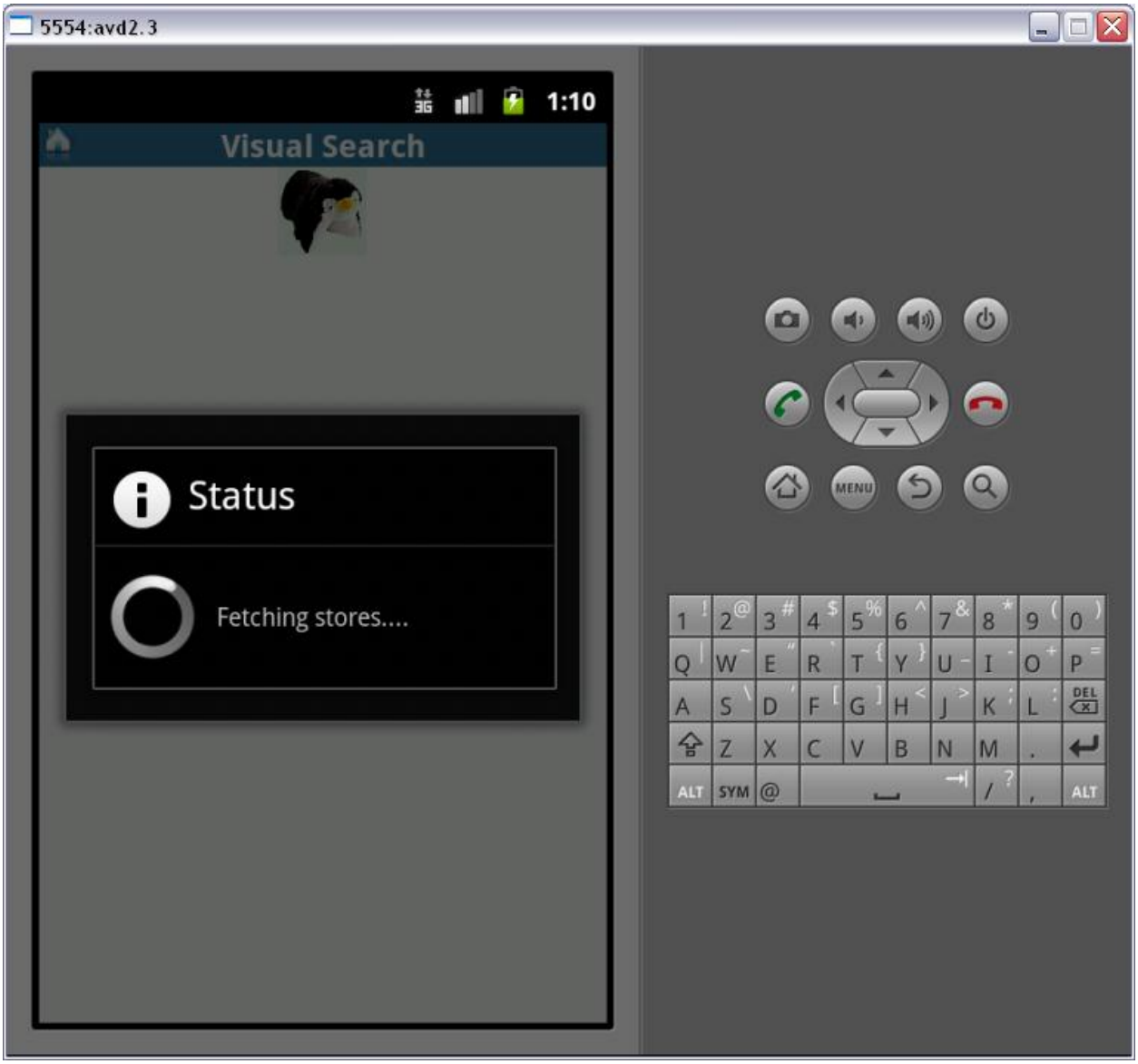


Figure 18: Finding Store Process Screenshot

6.7 Item Store and Price List Screen

It lists the stores and the item price received from the server.

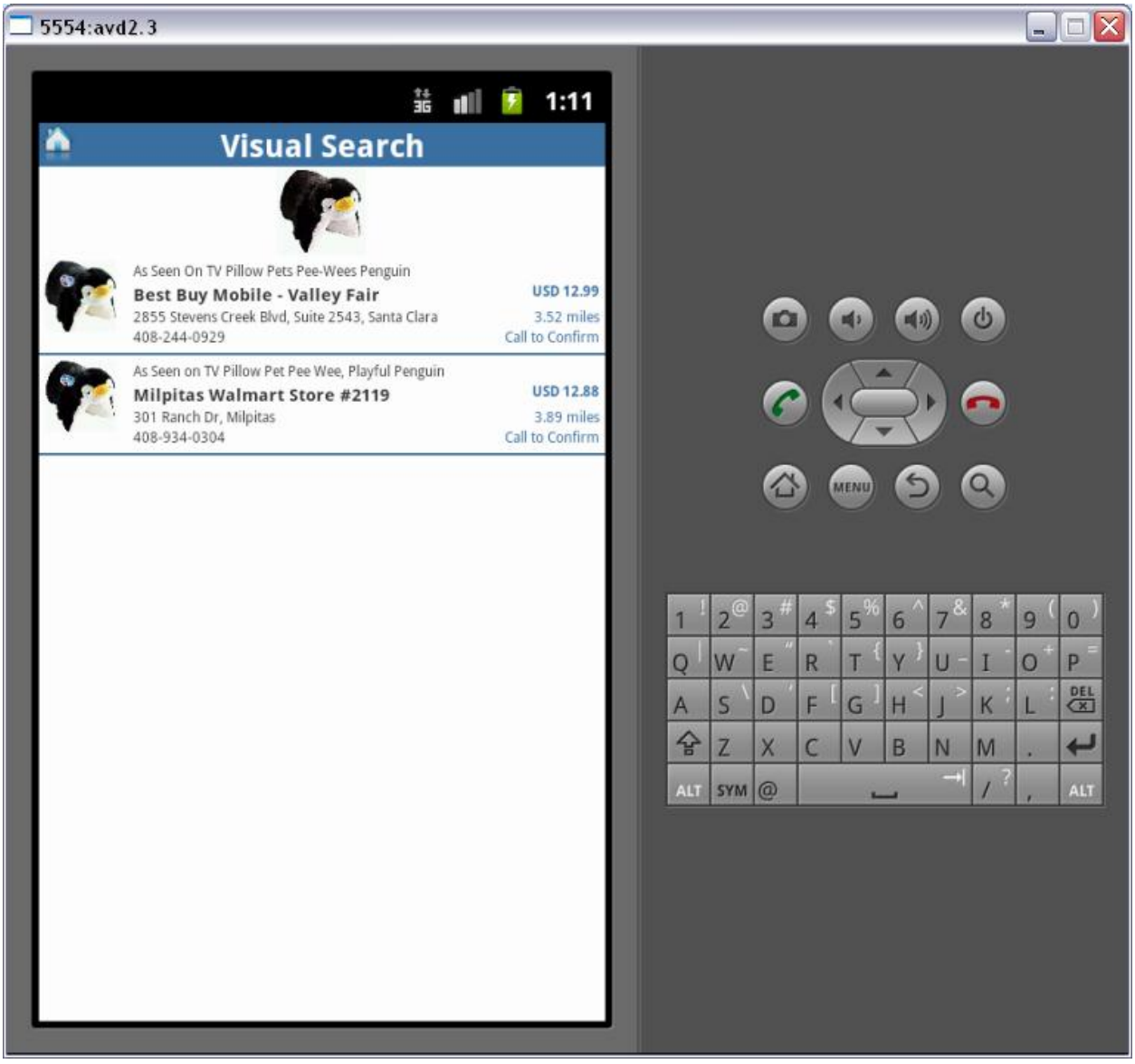


Figure 19: Store List and Item Price Screenshot

7. Conclusion

The time it takes to search for relevant data information through a visual search is greater than the time it takes to process a text search. The increase in the number of objects/distracters will increase the time of a search. For example, a search for a matching image in the database of 50000 images takes a long time; since the search has to go through the database and find the best match.

The Visual search engine does involve a learning curve at the initial stage. The search engine goes through the initial machine-learning phase. It has to be fed with thousands of images initially in order to identify the images in the future.

Image recognition algorithms are not accurate as the results may vary. Different algorithms employ different techniques to recognize the image. As image processing is a time consuming process, the SIFT algorithm takes more time to search because it goes through all the features linearly. This produces more accurate results, but requires a very long time to process; whereas SURF makes it faster and with accuracy.

Our algorithm makes use of SURF with K-Means to form clusters of image features. It applies techniques like: Indexing, Multi-Threading and Disk Space Efficiency to the project. We have developed a Mobile Visual Search application that searches for an item for the user by an image matching technique. It provides the nearby stores' addresses and the item price to the user.

8. References

1) Luo Juan and Oubong Gwun, Computer Graphics Lab, Chonbuk National University:

A Comparison of SIFT, PCA-SIFT and SURF, International Journal of Image Processing (IJIP) Volume (3), Issue (4), page 144

http://www.cscjournals.org/csc/manuscript/Journals/IJIP/volume3/Issue4/IJIP-51.pdf

2) C. Valgren, Topological mapping and localization using omnidirectional vision, Licentiate Thesis, Örebro University, Studies from the Department of Technology at Örebro University 27, Örebro 2007, page 99, 103

http://www.aass.oru.se/Research/Learning/publications/2007/Valgren_Licentiate_Thesis_Lowres.pdf

3) Christoffer Valgren and Achim J. Lilienthal, AASS Research Centre, Department of Computer Science, Örebro University: SIFT, SURF & Seasons: Appearance-based long-term localization in outdoor environments, page 6

http://www.aass.oru.se/Research/mro/publications/2007/Valgren_and_Lilienthal_2007-ECMR07-SIFT_SURF_and_Seasons.pdf

4) David G. Lowe, Computer Science Department, University of British Columbia: Distinctive Image Features from Scale-Invariant Key-points, Jan 5, 2004, page 2

http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

5) Edouard Oyallon and Julien Rabin: SURF: Speeded-Up Robust Features,

http://www.ipol.im/pub/algo/or_speeded_up_robust_features/#section_implementation

6) Mathias Lux, Institute for Information Technology and Savvas A. Chatzichristofis, Democritus University of Thrace, LIRe: Lucene Image Retrieval - An Extensible Java CBIR Library, 16th ACM international conference on Multimedia, 2008, http://delivery.acm.org/10.1145/1460000/1459577/p1085-lux.pdf?ip=173.8.133.161&acc=AUTHOR-IZED&CFID=82849508&CFTOKEN=38907829&__acm__=1337115020_ed181f18951cc8bcfe8b30d227e35f1b

7) http://en.wikipedia.org/wiki/SURF

8) http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

9) http://en.wikipedia.org/wiki/Visual_search_engine

10) http://en.wikipedia.org/wiki/Content-based_image_retrieval

11) http://stackoverflow.com/questions/843972/image-comparison-fast-algorithm

12) http://www.semanticmetadata.net/lire/

13) http://en.wikipedia.org/wiki/Google_Goggles