A Project Report

On

Medicine Name Detector

Submitted for the partial fulfilment

Of B.TECH Degree

In

Computer Science And Engineering

By

Vikas Singh
[1416510144]
Mohd. Abujar
[1416510062]
Praveen Kumar
[1416510085]

Under the supervision of

Mr. Rahul Singh

Assistant Professor (CS/IT Dept.)

Department of Computer Science And Engineering

Kanpur Institute of Technology

A-1, UPSIDC, Industrial Area Rooma, Kanpur

Dr. A.P.J Abdul Kalam Technical University,

Uttar Pradesh, Lucknow

June 2018

DECLARATION

This is to certify that Report entitled **“Medicine Name Detector”** which is submitted by me in partial fulfillment of the requirement for the award of degree B.Tech in Computer Sc. & Engineering/Information Technology to Kanpur Institute of Technology Kanpur, Uttar Pradesh Technical University, Lucknow comprises only my own work and due acknowledgement has been made in the text to all other material used.

Date:

Name & Roll No of Student:

VIKAS SINGH (1416510144)
MOHD. ABUJAR (1416510062)
PRAVEEN KUMAR (1416510085)

Approved By:

HOD

MR. PRAVEEN TRIPATHI
CS/IT Dept.
Kanpur Institute of Technology

CERTIFICATE

This is to certify that VIKAS SINGH, PRAVEEN KUMAR, MOHD. ABUJAR students of B.Tech. 8th sem in Computer Science And Engineering carried out the work present in the project titled **"Medicine Name Detector"** in the partial fulfillment of the degree of B.Tech. in Computer Science And Engineering from Kanpur Institute Of Technology, Kanpur**.**

Submitted By:

VIKAS SINGH (1416510144)

MOHD. ABUZAR (1416510062)

PRAVEEN KUMAR (1416510085)

GUIDED BY:

MR. RAHUL SINGH

(Assistant Professor)

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mentioning of the people whose constant guidance and encouragement made it possible. We take pleasure in presenting before you, our project, which is result of studied blend of both research and knowledge.

We express our earnest gratitude to our internal guide, Assistant Professor **Mr. Rahul Singh**, Department of CSE, our project guide, for his constant support, encouragement and guidance. We are grateful for his cooperation and his valuable suggestions.

Finally, we express our gratitude to all other members who are involved either directly or indirectly for the completion of this project.

(Signature)

Name :	Vikas Singh
Roll No.	1416510144
Date :	

(Signature)

Name :	Mohd. Abujar
Roll No.	1416510062
Date :	

(Signature)

Name :	Praveen Kumar
Roll No.	1416510085
Date :	

ABSTRACT

Success of **“Medicine Name Detector”** depends on two basic features that are text recognition from image and text to voice API.

Blindness makes life rather difficult for people who suffer from this health problem, but the use of technology can help in some day-to-day tasks. In this context, the present work focuses the development of a photo-to-speech application for the blind.

The project is called Camera Reading for Blind People, and its ultimate purpose is the development of a mobile application that allows a blind user to “read” text (a sheet of paper, a signal, etc.). To achieve that, a set of frameworks of Optical Character Recognition (OCR) and Text to Speech Synthesis (TTS) are integrated, which enables the user, using a smartphone, to take a picture and hear the text that exists in the picture.

TABLE OF CONTENTS

1. INTRODUCTION

Blind people are unable to perform visual tasks. For instance, text reading requires the use of a braille reading system or a digital speech synthesizer (if the text is available in digital format). The majority of published printed works does not include braille or audio versions, and digital versions are still a minority.

On the other hand, blind people are not able to read the simple warnings in walls or signals that surround us. Thus, the development of a mobile application that can perform the image to speech conversion, whether it’s a text written on a wall, a sheet of writing paper or in another support, has a great potential and utility.

The technology of optical character recognition (OCR) enables the recognition of texts from image data. This technology has been widely used in scanned or photographed documents, converting them into electronic copies, which one can edit, search, play its content and easily carry. The technology of speech synthesis (TTS) enables a text in digital format to be synthesized into human voice and played through an audio system.

The objective of the TTS is the automatic conversion of sentences, without restrictions, into spoken discourse in a natural language, resembling the spoken form of the same text, by a native speaker of the language. This technology has had significant progress over the last decade, with many systems being able to generate a synthetic speech very close to the natural voice. Research in the area of speech synthesis has grown as a result of its increasing importance in many new applications.

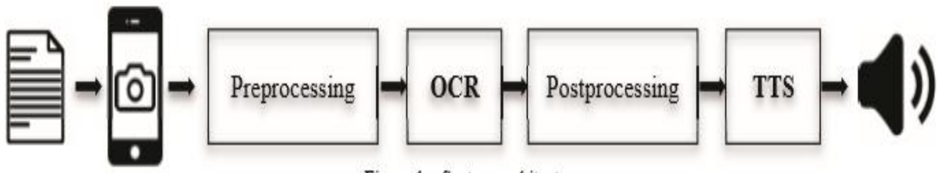


Figure 1 – System architecture

There are so many problems faced by a blind in daily routine life. This app “medicine name detector” is a small initial step taken to solving those problems for a disable and made them independent. By using the Concept of OCR and TTS this app detects all the text that could be find from an image and that results in a bunch of words.

Now to know which word is medicine name we had build a HASH MAP that contains medicine name as a key and there general use as medicine description. To detect exact name of medicine we use simple comparison between detected bunch of words and the keys / medicine names, present in Hash Map.

In this way a Blind Person will not need anyone for taking his daily medicine or during emergency need, our app will tell him that is he going to take correct medicine or not.

2. ANALYSIS

2.1 SYSTEM ANALYSIS

- ### Existing System

Today, there are already a few systems that have some promise for portable use, but they cannot handle product labeling. For example, portable bar code readers designed to help blind people identify different products in an extensive product database can enable users who are blind to access information about these products. Through Speech and Braille. But a big limitation is that it is very hard for blind users to find the position of the bar code and to correctly point the bar code reader at the bar code. Some reading assistive systems such as pen scanners might be employed in these and similar situations. Such systems integrate OCR software to offer the function of scanning and recognition of text and some have integrated voice output.

- ### Proposed System

To overcome the problems defined in problem definitions and also to assist blind persons to read text from those kinds of challenging patterns and backgrounds found on many everyday commercial products of Hand-held objects, then have to conceived of a camera-based assistive text reading framework to track the object of interest within the camera view and extract print text information from the object. Proposed algorithm used in this system can effectively handle complex background and multiple patterns, and extract text information from both hand-held objects and nearby signage.

- **Objective of the System**

This project help person to know about medicines name and description. Blind people are unable to perform visual tasks. So application help to recognize the text from image and convert into the voice. This is also help to organize his things as add medicine name and description for regular uses so on.
Make a disable an independent by using medicine name detector application. Text to voice system helps the person to know what medicine is it and why it is used.

2.2 System Specifications

- **Hardware Requirements**

- Dual Core Processor
- 2 GB Ram
- 1024 kb Cache Memory
- Hard disk 64 GB
- Microsoft Compatible 101 or more Key Board
- Camera
- Android device

- **Software Requirements**

• Operating System	:	Windows XP/7/8/8.1
• Programming language	:	Java
• Tool Kit	:	Android Studio 2.3.3
• Application Run Environment (OS)	:	Android

3. MODULES

3.1 Optical Character Recognition (OCR)

Optical character recognition, usually designated by the acronym OCR, is the process of recognition and automatic conversion of existing characters in the written-support image into the text format, which can then be used in various applications. The OCR has been widely studied and has displayed considerable advances regarding the performance and accuracy of the obtained results .

The process of optical character recognition can be summarized as a process that follows a set of steps:

- Optical image acquisition
- Location and segmentation
- Preprocessing
- Feature extraction
- Classification
- Post processing

The final text is then converted into the desired document format (rtf, txt, pdf, etc.).

3.2 Text-to-Speech (TTS) overview

Voice synthesis, defined as TTS (acronym for Text-To-Speech), is a computer system that should be able to read aloud any text, regardless of its origin . The use of TTS aims to produce human voice artificially. Voice synthesis is a complex process and complex algorithms are needed to produce an intelligible and natural result. TTS synthesis makes use of techniques of Natural Language Processing. Since the text to be synthesized is the first entry of the system, it must be the first to be processed.

There are several techniques to create a synthesized voice:

- Articulatory synthesis
- Formant synthesis
- Concatenation synthesis
- Hidden Markov models synthesis

The main synthesis techniques, presented above, are the methods used in the study and development of speech synthesis systems. However, a way to profit from the inherent advantages of each technique is to use a hybrid of the various techniques in the development of future systems speech synthesis.

The quality of a speech synthesis can be determined by its naturalness and intelligibility. Naturalness is the characteristic that describes how close to the human voice is the sound obtained by TTS; intelligibility refers to the ease with which the sound is understood in complex situations.

3.3 State of Art

There are some works being done within the image-to-speech research area. The Phone Reader thesis presents an approach to a system that allows recognizing the text in a photo using the Tesseract OCR, and the reading of the text is made through the TTS, in the selected language. The OCR processing and translation are made on an external server, which requires a data connection for communication between the server and the device.

In the paper Optical Character Recognition , an application is presented, that performs the recognition of the existing text in photos via OCR Tesseract and, after that, the speech synthesis of recognized text is made. In this application, all processing is done locally on the mobile device, and there will be no preprocessing or recognition of the limits of the images, so these will be defined as future work.

The paper Open Source OCR Framework Using Mobile Device describes a project in which the image is captured with the device camera, and is converted to a bitmap format. The multi-channel RGB image is converted into single-channel intensity image. Then it made a post processing, removing the noise generated by OCR, one based on ASCII code in the recognized text and all alphabetical and numerical characters are removed, no filter being applied. This is indicated as the future work, refining the results by using OCR cross-referencing with the dictionary, and other high level semantic techniques.

In **OCRdroid**: A Framework to Digitize Text Using Mobile Phones, the operation of a scanning system for obtaining text contained in images is discussed. This is a project whose aim is to develop an application for Android, which makes a preprocessing of the images as they are captured, correcting the existing lighting and checking if the images are properly positioned for better recognition of the text, and then sends the obtained image to an external server which processes the image recognition OCR.

Some public mobile applications also exist in this area, that are able to recognize text using OCR and reading it using the system internal TTS (like the iOS Voice Over feature):

• Say Text is an application that enables you to take photographs of texts, and after recognizing the text, you are allowed to send it by email or obtain the reading of it.

• Talking Camera Pro, for visually impaired/blind, is an application that recognizes text via OCR, synthesizes it and returns its reading.

• Prizmo - Escaneamento, OCR & Fala, is an application which allows recognizing various image and document formats, the result can be exported as PDF / Text, vCard or JPEG / PNG. The application can make the processing of images obtained and also the synthesis of the text.

Despite the existence of studies and applications that address the theme proposed in this project, it seems that existing solutions continue to present several limitations, which relate to the efficiency in recognition, particularly in situations where shooting conditions are not ideal. This research area still needs to achieve greater maturity to overcome the limitations of reallife situations where light conditions, the focus, alignment or the existence of shadows do not allow the effective recognition by OCR.

The project Camera Reading for Blind People is therefore an additional contribution to the area, since it provides additional experiencies on the subject and presents some foundations for the development of related applications for iOS devices.

4. Proposed Architecture

In this section, the methodology to pursue the project is presented, from image capture with the camera, through implementation of OCR and TTS tools.

The overall system architecture is represented by the following figure 1:

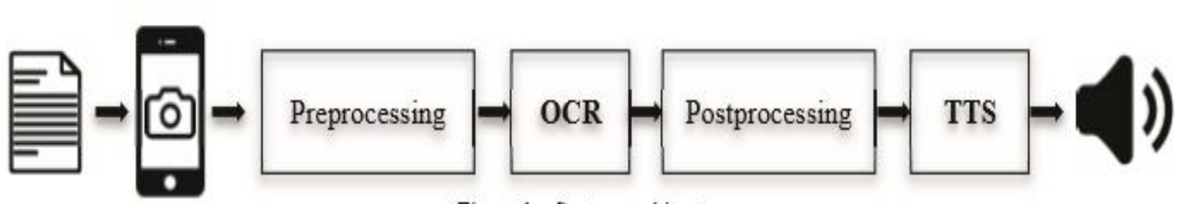


Figure 1 – System architecture

4.1. Image capture with the camera

The camera of the mobile device is critical to use the application, since it will be essential for the user to take the picture of the brackets containing text that will be recognized and synthesized.

The image capture is done inside the application itself, thus avoiding the use of additional applications such as access to the photo gallery.

The device used in the project was an Apple iPhone 5. The display screen size is 4.0' with a resolution of 640x 1136 pixels. The camera of the device has a resolution of 8 megapixels with 3264x2448 pixels, autofocus and LED Flash.

The application will run in full screen mode. The captured image will be the one displayed on the screen. The autofocus and auto flash camera options will be set as switched on.

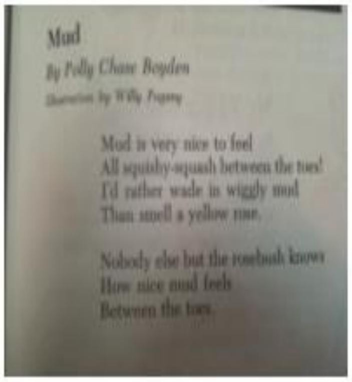
4.2. Implementation of OCR tools

After the image is obtained, the next step is the optical recognition of existing text in it, through use of an OCR framework.

Optical character recognition, usually known as OCR, is the interpretator of images that have text and, after they are scanned, the text contained in them can be recognized, edited and used.

4.3. Choice of OCR framework

To compare the efficiency of recognition of each framework, a set of 30 pictures of texts with different shapes, sizes and layouts was collected. We tried to get a set of images that represent a variety of situations as embracing as possible, with texts in different shapes, sizes and colors, different alignments for the images, different exposures to light at the time of image capture and shadows in an attempt to get the most accurate representation of real situations that will surely happen when using the application.



Mud

By Polly Chase Boyden

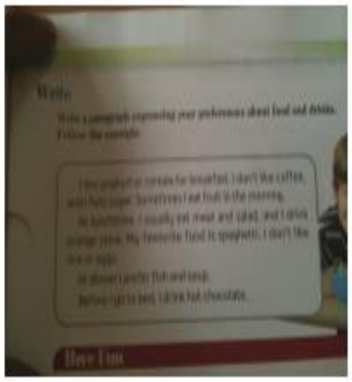
[illegible]

Mud is very nice to feel
All squishy-squash between the toes!
I'd rather wade in wiggly mud
Than smell a yellow rose.

Nobody else but the rosebush knows
How nice mud feels
Between the toes.

Simon's house

[illegible]



Write

Write a paragraph expressing your preferences about food and drinks. Follow the example.

[illegible]

At dinner I prefer fish and soup.

Before I go to bed, I drink hot chocolate.

[illegible]

Figure 2 – Example of images used in tests

4.4. Levenshtein distance

For measurement values that can be used in the comparison between the texts, the optically recognized and the original, a function that reproduces the string comparison algorithm Levenshtein distance was developed.

The Levenshtein distance algorithm was created by the Russian scientist Vladimir Levenshtein in 1965. The basic principle of the algorithm is the measurement of similarity between two strings. This is done by calculating the number of basic operations or changes needed to make two strings equal. Changes may consist of a substitution, insertion or deletion of a character. The Levenshtein distance d (p, t) between two strings (p and t represent the strings) is the minimum number of operations required to make p = t. For example, if p = "moose" and t = "moody", the Levenshtein distance is d (p, t) = 2 since the strings can be made using the same two substitutions (replacement of letter 's' by letter 'd', exchange of letter 'e' by letter 'y'.

Taking another example, the Levenshtein distance between the words "kitten" e "sitting" is 3, since with only 3 changes one can make a word turn into another, and there is no way to do it with less than three changes [17].

• kitten

• sitten (substitution of 'k' for 's')

• sittin (substitution of 'e' for 'i')

• sitting (insertion of 'g' at the end)

Tests were made with the 30 images being recognized by OCR, and the difference obtained between the recognized text and the original text was verified. The total of characters of the original texts was registered and the Levenshtein distance obtained between the original text and the recognized text was calculated, as well as the percentage value of that distance, obtained by dividing the distance obtained by the total of characters of the text.

It was then calculated the median value of the distance values obtained. Unlike the average, the median is not influenced by extreme values, since it is essentially linked to the position it occupies in the ordered set. Thus, if some value is too big or small – outliners –, it will not affect the calculation of the median, since it does not alter the order.

The results of Levenshtein distance for the three tested frameworks are presented in the following graph in figure 3:

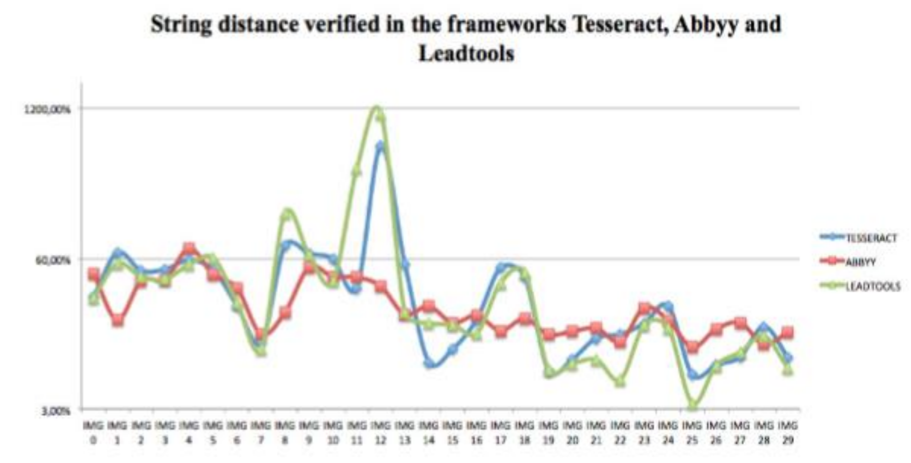


Figure 3 – Comparison of the string distance from the tested images in frameworks Tesseract, Abbyy and Leadtools (the y-axis values are on a logarithmic scale)

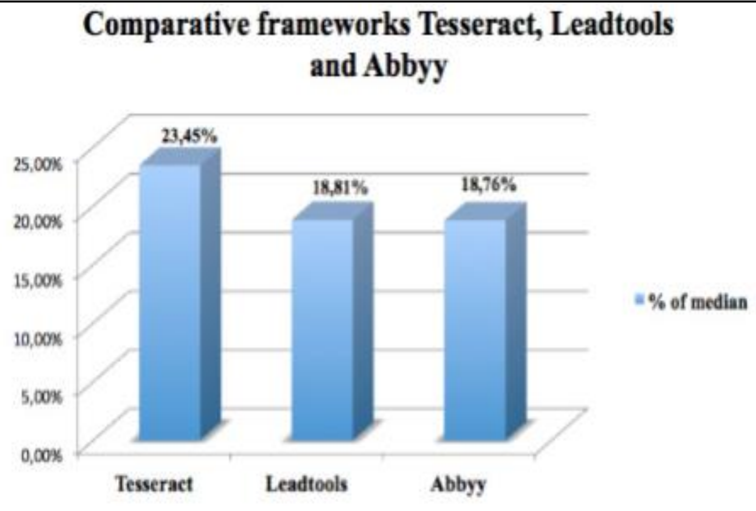


Figure 4 – Comparison of the median value of the string distance in the tested images in frameworks Tesseract, Abbyy and Leadtools

(Tesseract - 23.45%, Abbyy - 18.76% and 18.81% Leadtools), the research work was based on Tesseract framework due to budget limitations.

Set up the TextRecognizer and CameraSource

To start things out, we're going to create our TextRecognizer. This detector object processes images and determines what text appears within them. Once it's initialized, a TextRecognizer can be used to detect text in all types of images. Find the createCameraSource method and build a TextRecognizer.

OcrCaptureActivity.java

```
private void createCameraSource(boolean autoFocus, boolean useFlash) {
   Context context = getApplicationContext();

   // TODO: Create the TextRecognizer
   TextRecognizer textRecognizer = new TextRecognizer.Builder(context).build();
   // TODO: Set the TextRecognizer's Processor.

   // TODO: Check if the TextRecognizer is operational.

   // TODO: Create the mCameraSource using the TextRecognizer.
}
```

Just like that, the TextRecognizer is built. However, it might not work yet. If the device does not have enough storage, or Google Play Services can't download the OCR dependencies, the TextRecognizer object may not be operational. Before we start using it to recognize text, we should check that it's ready. We'll add this check to createCameraSource after we initialized the TextRecognizer.

OcrCaptureActivity.java

```
// TODO: Check if the TextRecognizer is operational.
if (!textRecognizer.isOperational()) {
    Log.w(TAG, "Detector dependencies are not yet available.");

    // Check for low storage.  If there is low storage, the native library will not be
    // downloaded, so detection will not become operational.
    IntentFilter                          lowstorageFilter                        =                     new
IntentFilter(Intent.ACTION_DEVICE_STORAGE_LOW);
    boolean hasLowStorage = registerReceiver(null, lowstorageFilter) != null;

    if (hasLowStorage) {
        Toast.makeText(this, R.string.low_storage_error, Toast.LENGTH_LONG).show();
        Log.w(TAG, getString(R.string.low_storage_error));
    }
}
```

Note: If you have the OCR dependency in AndroidManifest.xml (which, in the case of this lab, was done for you), the OCR dependencies will automatically download when the app is installed on the device. Because of this, most devices will have it available the first time you attempt to use your detector. But it's an important condition to check in order to account for unexpected storage or connection issues.

Now that we've checked that the TextRecognizer is operational, we could use it to detect individual frames. But we want to do something a little more interesting: read text live in the camera view. In order to do that, we'll create a CameraSource, which is a camera manager pre-configured for Vision processing. We're going to set the resolution high and turn autofocus on, because that's a good match for recognizing small text. If you knew your users would be looking at large blocks of text, like signage, you might use a lower resolution, which would be able to process frames more quickly.

OcrCaptureActivity.java

```
// TODO: Create the cameraSource using the TextRecognizer.
cameraSource =
        new CameraSource.Builder(getApplicationContext(), textRecognizer)
        .setFacing(CameraSource.CAMERA_FACING_BACK)
        .setRequestedPreviewSize(1280, 1024)
        .setRequestedFps(15.0f)
        .setFlashMode(useFlash ? Camera.Parameters.FLASH_MODE_TORCH : null)
        .setFocusMode(autoFocus                                                ?
Camera.Parameters.FOCUS_MODE_CONTINUOUS_VIDEO : null)
        .build();
```

Note: A CameraSource class, very similar to the one in this project, is also included directly in the com.google.android.gms.vision package. Using that class may be preferable to writing your own camera manager. However, if you want to integrate frame processing into an existing camera application, or have very specific needs, the CameraSource code in this sample makes an excellent reference for how to integrate a processor efficiently into your camera manager.

Here's what the complete createCameraSource should look like when you're done:

OcrCaptureActivity.java

```
private void createCameraSource(boolean autoFocus, boolean useFlash) {
    Context context = getApplicationContext();

    // Create the TextRecognizer
    TextRecognizer textRecognizer = new TextRecognizer.Builder(context).build();
    // TODO: Set the TextRecognizer's Processor.

    // Check if the TextRecognizer is operational.
    if (!textRecognizer.isOperational()) {
        Log.w(TAG, "Detector dependencies are not yet available.");

        // Check for low storage.  If there is low storage, the native library will not be
        // downloaded, so detection will not become operational.
        IntentFilter lowstorageFilter = new
IntentFilter(Intent.ACTION_DEVICE_STORAGE_LOW);
        boolean hasLowStorage = registerReceiver(null, lowstorageFilter) != null;

        if (hasLowStorage) {
            Toast.makeText(this, R.string.low_storage_error,
Toast.LENGTH_LONG).show();
            Log.w(TAG, getString(R.string.low_storage_error));
        }
    }

    // Create the cameraSource using the TextRecognizer.
    cameraSource =
            new CameraSource.Builder(getApplicationContext(), textRecognizer)
            .setFacing(CameraSource.CAMERA_FACING_BACK)
            .setRequestedPreviewSize(1280, 1024)
            .setRequestedFps(15.0f)
            .setFlashMode(useFlash ? Camera.Parameters.FLASH_MODE_TORCH : null)
            .setFocusMode(autoFocus ?
Camera.Parameters.FOCUS_MODE_CONTINUOUS_VIDEO : null)
            .build();
```

4.5. Implementation of the tool Text-To-Speech

There are several available frameworks TTS, some free, such as the case of GoogleTTS and OpenEars, or others commercial, such as the case of iSpeech or AcapelaTTS.

Several tests with free frameworks were performed, verifying very similar results in the synthesis of human voice with the provided strings. In spite of existing several frameworks TTS, iOS 7 now supports native TTS support, though the use of a new class has been included in AVFoundation framework, the AVSpeechSynthesizer class, that will be used because it allows to obtain the synthesis of voice desired for the application, without resort to external frameworks.

Create a Speech Method

To keep your Android classes well-organized, it's advisable to create dedicated methods for processes you may want to use more than once. Add the following method outline to your Activity:

```
private void speakWords(String speech) {
//implement TTS here
}
```

This is where your TTS processing will go. Back in the "onClick" listener method, call this new method, passing it the string variable your code copied from the text-field:

```
speakWords(words);
```

Using a method for the TTS process means that your code can call on it elsewhere if necessary.

Implement TTS Within the Class

To utilize the TTS facility, you need to make a few more changes to your class declaration. Add the following import statements for the TTS classes at the top of your file:

```
import android.speech.tts.TextToSpeech;
import android.speech.tts.TextToSpeech.OnInitListener;
```

You also need to implement one more interface, so alter your class declaration outline to add "OnInitListener" as in the following example:

```
public class SpeakingAndroid extends Activity implements OnClickListener, OnInitListener
```

Remember to use your own class name. Again, your IDE will alert you to the fact that you haven't yet implemented this interface but don't worry, you will soon.

Check for TTS Data

Your app needs to check that the user has the data necessary for the TTS function before you call its methods. Declare and instantiate the following instance variable at the top of your Activity class declaration, before the "onCreate" method:

```
private int MY_DATA_CHECK_CODE = 0;
```

Add the following import statement at the top of the class:

```
import android.content.Intent;
```

In the "onCreate" method, add the following:

```
Intent checkTTSIntent = new Intent();
checkTTSIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(checkTTSIntent, MY_DATA_CHECK_CODE);
```

This code creates a new Intent purely for the purposes of checking the user data. When the checking process is complete, the code will call the "onActivityResult" method.

Create a TTS Instance

Declare an instance variable for your TTS object at the top of the class declaration, also before the "onCreate" method:

```
private TextToSpeech myTTS;
```
Add the "onActivityResult" to your class as follows:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
		if (requestCode == MY_DATA_CHECK_CODE) {
			if (resultCode == TextToSpeech.Engine.CHECK_VOICE_DATA_PASS) {
				myTTS = new TextToSpeech(this, this);
			}
			else {
				Intent installTTSIntent = new Intent();

		installTTSIntent.setAction(TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
				startActivity(installTTSIntent);
			}
	}
}
```
When the data checking Intent completes, the app calls this method, passing it the "MY_DATA_CHECK_CODE" variable indicating whether or not the user has the TTS data installed. If the data is present, the code goes ahead and creates an instance of the TTS class. If the data is not present, the app will prompt the user to install it.

Provide the onInit Method

Your class declaration is implementing "OnInitListener" so you must provide an "onInit" method. In this method, you can carry out any final set-up checks you need, as well as choosing settings for your TTS instance, such as language and locale options. Add the following import statement at the top of your class:

```
import java.util.Locale;
```

Add the "onInit" method to the class as follows:

```
public void onInit(int initStatus) {
		if (initStatus == TextToSpeech.SUCCESS) {
			myTTS.setLanguage(Locale.US);
		}
}
```

This code checks that the TTS resource is successfully instantiated, then sets a US English Locale for the speech operations. You can optionally output an error message for users if the TTS does not successfully instantiate by adding the following after the "if" block:

```
else if (initStatus == TextToSpeech.ERROR) {
	Toast.makeText(this, "Sorry! Text To Speech failed...", Toast.LENGTH_LONG).show();
}
```

If you use this code you will also need to import the Toast class by adding the following statement at the top of your file:

```
import android.widget.Toast;
```

Your app can carry out checks on the user device, such as available languages, as in the following extended version of the statement creating the TTS object inside the first conditional statement:

```
if(myTTS.isLanguageAvailable(Locale.US)==TextToSpeech.LANG_AVAILABLE)
myTTS.setLanguage(Locale.US);
```

Speak!

Finally, your app is ready to speak. Inside the "speakWords" method, add the following code:

```
myTTS.speak(speech, TextToSpeech.QUEUE_FLUSH, null);
```

There are lots of options here in terms of how your app handles speech. This code instructs the app to speak the text string immediately. If you want to add consecutive speech operations, you can instruct the app to wait until any current speech operations finish by adding your new speech item to a queue, as follows:

```
myTTS.speak(speech, TextToSpeech.QUEUE_ADD, null);
```

Once your class is finished with the TTS, you can optionally shut it down as follows:

```
myTTS.shutdown();
```

Don't include this line if you want your users to be able to make the app speak more than once.

5. System Optimization

Using the framework Tesseract, we proceeded to a set of tests in order to seek better results in the recognition of images, and greater efficiency in the use of the application. In order to do this, two stages were added: a preprocessing and a postprocessing

5.1. Preprocessing

Given that the image used in the optical recognition process has a considerable impact on the results, particularly regarding image quality, brightness, contrast, focus, among other features, the project proceeded to attempt to make changes to images shot by the device, to improve the result in his reading OCR.

Thus, we used image filters in order to make a preprocessing of the images so that we could get better results.

A set of tests was run to obtain the median percentage of Levenshtein distance, resulting from the difference between the original text and the recognized text, after the application of each of these filters to the set of the 30 images collected for testing.

Various filters available for iOS (Core Image [8]) were tested, namely CIToneCurve (adjusts tone response of the R, G, and B channels of an image), CIHueAdjust (changes the overall hue, or tint, of the source pixels) CIGammaAdjust (adjusts midtone brightness), CIColorControls (adjusts saturation, brightness, and contrast values), CIColorMonochrome (remaps colors so they fall within shades of a single color), CIExposureAdjust (adjusts the exposure setting for an image similarly to the way you control exposure for a camera when you change the F-stop).

It should be noted that, in the tests for each filter, the values of its various parameters were changed in cycle, seeking the best results. The values for the parameters of each filter were calculated this way, which allowed us to obtain better results. First the filters were tested individually, and after that they were combined trying to find even better results.

After running several tests, it was shown that the combination of filters CIColorControls and CIColorMonochrome produced the best results, so this filter combination will always be applied to images captured by the device immediately before the OCR recognition process.

The CIColorControls filter allows you to adjust the saturation, brightness and contrast in an image. It receives as parameters a CIImage image, and the values for the referred attributes can be changed. The used values were 1.00 to saturation, 0.50 to brightness and 2.14 to contrast.

The CIColorMonochrome filter allows remapping the colors in an image so that they fall within shades of a single color. It receives as parameters a CIImage image, and the values for the input color and the intensity can be changed. The applied values were 0.77 for the intensity and "white" as the color for which it is intended to remap the others.

The results obtained after application of the filter combination are as follows in figure 5:

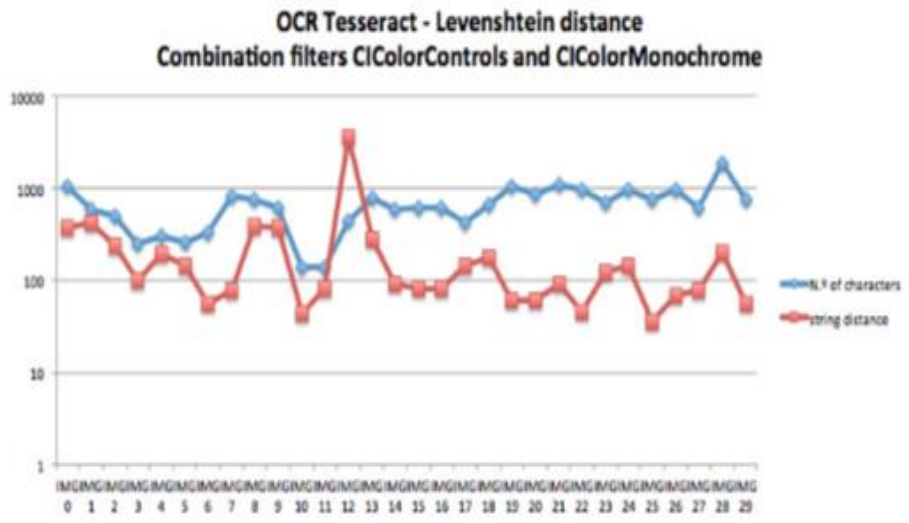


Figure 5 – Values of string distance obtained after applying the filter combination CIColorControls and CIColorMonochrome

After applying this combination of filters, there was a substantial improvement in the results, the average value of the distance between the original text and the recognized decreases from 23.45% to 17.83%.

5.2. Postprocessing

During the testing of the application, particularly when using the image captured with the camera of the device, it was found that in certain readings, the result of OCR recognition was a set of meaningless characters, that in no way reflected the text in the images. This result was caused by the poor quality of the captured image, or OCR recognition errors. As expected, the oral synthesis of these situations turned out to be misleading and unpleasant to the user.

After several tests were performed, and observing the resulting optical text recognition, it was found that the normal pattern of strings in the case of incorrect readings presented several special characters indicated below:

( ) ; * ~ » “ . =

The indicated character set was based on the observation of several samples tested, and can naturally be modified in future versions, or new characters to be included in this set, simply by including them in the function.

To avoid such a situation, regular expressions were used to test if the text returned by the OCR corresponded to a readable text or if it returned many occurrences of the listed special characters. Thus, the created function receives the text recognized by OCR, counts the total number of characters in the text, and counts the total occurrences of the above special characters, calculating a percentage value of these special characters in relation to the total characters of the text, referred to as error rate, obtained via the following formula 1:

Error rate = Occurrences of special characters / total characters of the text

If this percentage exceeds the defined value, which can be configured by the user, the application presents an error in processing, instead of performing the synthesis of the text, prompting the user to repeat the process of capturing the text.

In order to assess the initial percentage value for the application, and to set the range of values to be changed, we proceeded to a set of tests using the set of images already used in other tests. The percentage of allowed special characters was defined and, when these values were exceeded, the situations were considered as processing error.

The graphical representation of the values obtained is presented below in figure 7:

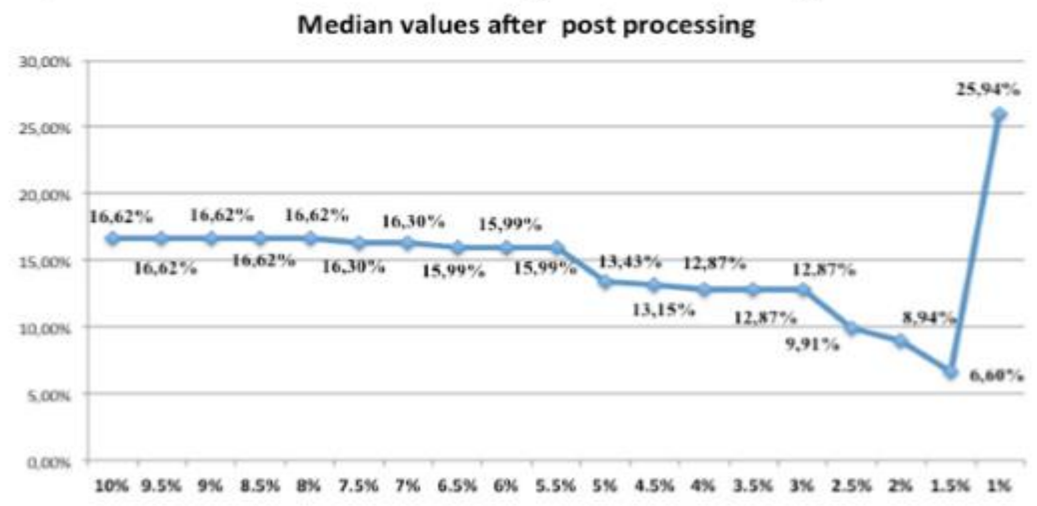


Figure 6 – Median values of the distance after post processing

By analyzing the results, it can be seen that the smaller the percentage of allowable special characters, the better the results of the obtained percentage of the median error are. This is a consequence of the fact that the images with worst OCR recognition results are discarded. The exceptional downward trend in the percentage of the median occurs in the value of 1%, since in this value only the reading of two images are regarded as valid, so this value can not be considered as a reference value for drawing some conclusions from their use.

It was found that for percentages below 3% the amount of discarded images is very high, exceeding 50% of the 30 images tested. The use of such a low rate may make the implementation little effective, often showing the error message of recognition.

The default value for the first use of the application shall be 5%, since, after the tests carried out using this value as a reference, it was found that, although the 5 discarded images of the all 30 tested, corresponding to a percentage of approximately 16,7%, the percentage value of the median of distance obtained is 13,43%, which corresponds to a substantial improvement in the figure registered without error prevention, 17,83%.

Of course the user may choose to change the set value for the percentage, which may be increased to get more incorrect results and fewer error messages, or decreased to get more effective results but a larger number of situations of recognition error.

6. TECHNOLOGY USED

6.1 JAVA

- **HISTORY:**

Java – was created in 1991 – by James Gosling et al. of Sun Microsystems. – Initially called Oak, in honor of the tree outside Gosling's window, its name was changed to Java because there was already a language called Oak.

- **The original motivation for Java**

The need for platform independent language that could be embedded in various consumer electronic products like toasters and refrigerators.

- **One of the first projects developed using Java**

  A personal hand-held remote control named Star 7.

At about the same time, the World Wide Web and the Internet were gaining popularity. Gosling et. al. realized that Java could be used for Internet programming.

- The Java technology is:

  A programming language
  A development environment
  An application environment
  A deployment environment

- **Programming Language**

  As a programming language, Java can create all kinds of applications that you could create using any conventional programming language

As a development environment, Java technology provides you with a large suite of tools:

– A compiler (javac)

– An interpreter (java)

– A documentation generator (javadoc)

– A class file packaging tool

Java technology applications are typically general-purpose programs that run on any machine where the Java runtime environment (JRE) is installed.

● There are two main deployment environments:

1. The JRE supplied by the Java 2 Software Development Kit (SDK) contains the complete set of class files for all the Java technology packages, which includes basic language classes, GUI component classes, and so on.

2. The other main deployment environment is on your web browser. Most commercial browsers supply a Java technology interpreter and runtime environment.

- **Some features of Java:**

– The Java Virtual Machine

– Garbage Collection

– Code Security

- **Java Virtual Machine (JVM)**

– an imaginary machine that is implemented by emulating software on a real machine

– provides the hardware platform specifications to which you compile all Java technology code

● Bytecode

– a special machine language that can be understood by the Java Virtual Machine (JVM) – independent of any particular computer hardware, so any computer with a Java interpreter can execute the compiled Java program, no matter what type of computer the program was compiled on

● Garbage collection thread

– responsible for freeing any memory that can be freed. This happens automatically during the lifetime of the Java program. – programmer is freed from the burden of having to deallocate that memory themselves

Code security is attained in Java through the implementation of its Java Runtime Environment (JRE).

● JRE

– runs code compiled for a JVM and performs class loading (through the class loader), code verification (through the bytecode verifier) and finally code execution

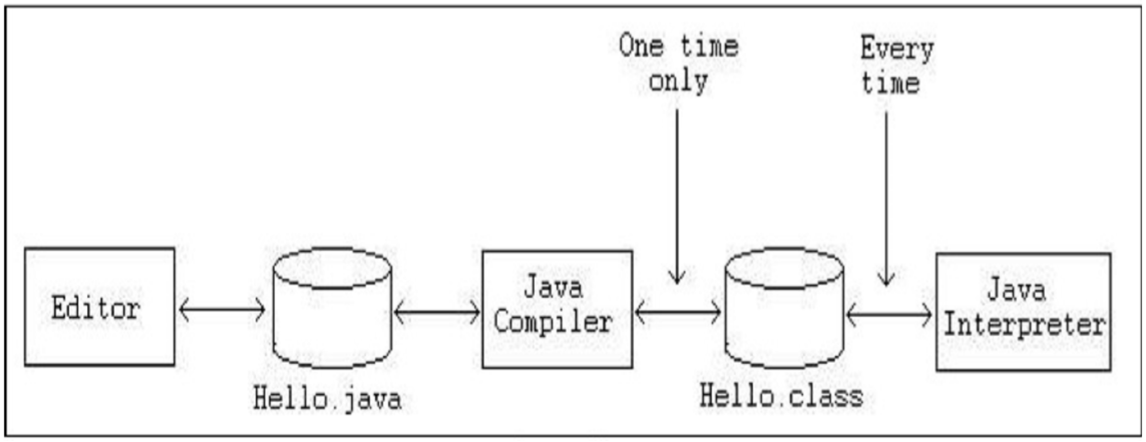
● Class Loader

– responsible for loading all classes needed for the Java program – adds security by separating the namespaces for the classes of the local file system from those that are imported from network sources – After loading all the classes, the memory layout of the executable is then determined. This adds protection against unauthorized access to restricted areas of the code since the memory layout is determined during runtime

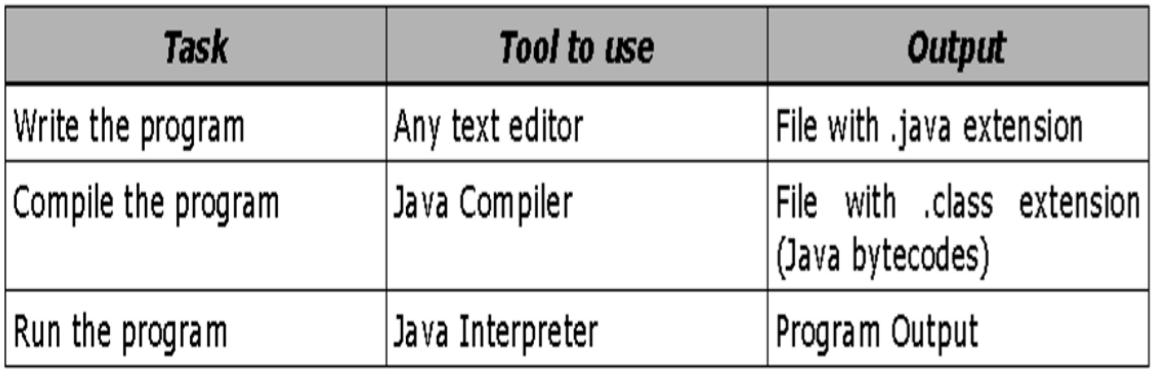
● Bytecode verifier

– tests the format of the code fragments and checks the code fragments for illegal code that can violate access rights to objects

The following figure describes the process of compiling and executing a Java program



● Phases of java



Task	Tool to use	Output
Write the program	Any text editor	File with .java extension
Compile the program	Java Compiler	File with .class extension (Java bytecodes)
Run the program	Java Interpreter	Program Output

6.2 XML TECHNOLOGIES

XML on Android

The Android platform is an open source mobile development platform. It gives you access to all aspects of the mobile device that it runs on, from low level graphics, to hardware like the camera on a phone. With so many things possible using Android, you might wonder why you need to bother with XML. It is not that working with XML is so interesting; it is working with the things that it enables.

XML is commonly used as a data format on the Internet. If you want to access data from the Internet, chances are that the data will be in the form of XML. If you want to send data to a Web service, you might also need to send XML.

In short, if your Android application will leverage the Internet, then you will probably need to work with XML. Luckily, you have a lot of options available for working with XML on Android.

Xml parsers

One of the greatest strengths of the Android platform is that it leverages the Java programming language. The Android SDK does not quite offer everything available to your standard Java Runtime Environment (JRE,) but it supports a very significant fraction of it.

The Java platform has supported many different ways to work with XML for quite some time, and most of Java's XML-related APIs are fully supported on Android. For example, Java's Simple API for XML (SAX) and the Document Object Model (DOM) are both available on Android.

Both of these APIs have been part of Java technology for many years. The newer Streaming API for XML (StAX) is not available in Android. However, Android provides a functionally equivalent library.

Finally, the Java XML Binding API is also not available in Android. This API could surely be implemented in Android. However, it tends to be a heavyweight API, with many instances of many different classes often needed to represent an XML document.

Thus, it is less than ideal for a constrained environment such as the handheld devices that Android is designed to run on. In the following sections, you will take a simple source of XML available on the Internet, and see how to parse it within an Android application using the various APIs mentioned above. First, look at the essential parts of the simple application that will use XML from the Internet.

6.3 SHAREDPREFERENCES

SharedPreferences are key-value pairs of primitive data types that are saved in a file within an apps file structure. You can then access this file from anywhere within the app to either put data into the file or take data out of the file. You can't access the file from another app so it's pretty secure from that point of view.

Use of SharedPreferences

You would for example use SharedPreferences in a game where you would save the user's name, high score, and state of the game when they logged off. Then the next time they log in, their score and game level would be retrieved from the preference file and they would continue the game from where they ended it when they logged off.

Save your data in a single or multiple SharedPreferences files

You can save your preferences data in a single file or in multiple files, depending on your needs. The process is the same except for the SharedPreferences object that you get. In the case of a single file, call getPreferences() to get a SharedPreferences object and for multiple files, call getSharedPreferences() and pass it a name for the file as a parameter.

 getPreferences() - for Activity level preferences. Each Activity will have it's own preference file

 getSharedPreferences() - for application-level preferences. You can access the preference file from anywhere in the application

Retrieving the data

Once you have the SharedPreferences object, you can use a number of get<type> methods to retrieve the values.

Saving your data

Saving the data is pretty straight forward too. Simply use the SharedPreferences.Editor class to get an Editor object by calling edit() on the relevant SharedPreferences object. Then use the put<type> methods to save the data, finalising the process by calling apply() or commit() on the Editor object.

The difference between apply() and commit()

apply()

This saves your data into memory immediately and saves the data to disk on a separate thread. So there is no chance of blocking the main thread (your app won’t hang). It is the preferred technique but has only been available since Gingerbread (API 9, Android 2.3).

commit()

Calling this will save the data to the file however, the process is carried out in the thread that called it, stopping everything else until the save is complete. It returns true on successful completion, false on failure. Use commit() if you need confirmation of the success of saving your data or if you are developing for pre-Gingerbread devices. commit() has been available since API 1

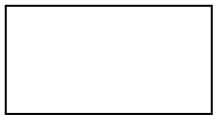
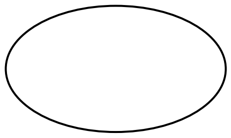
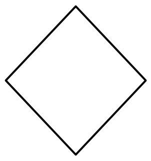
Type of data that you can save

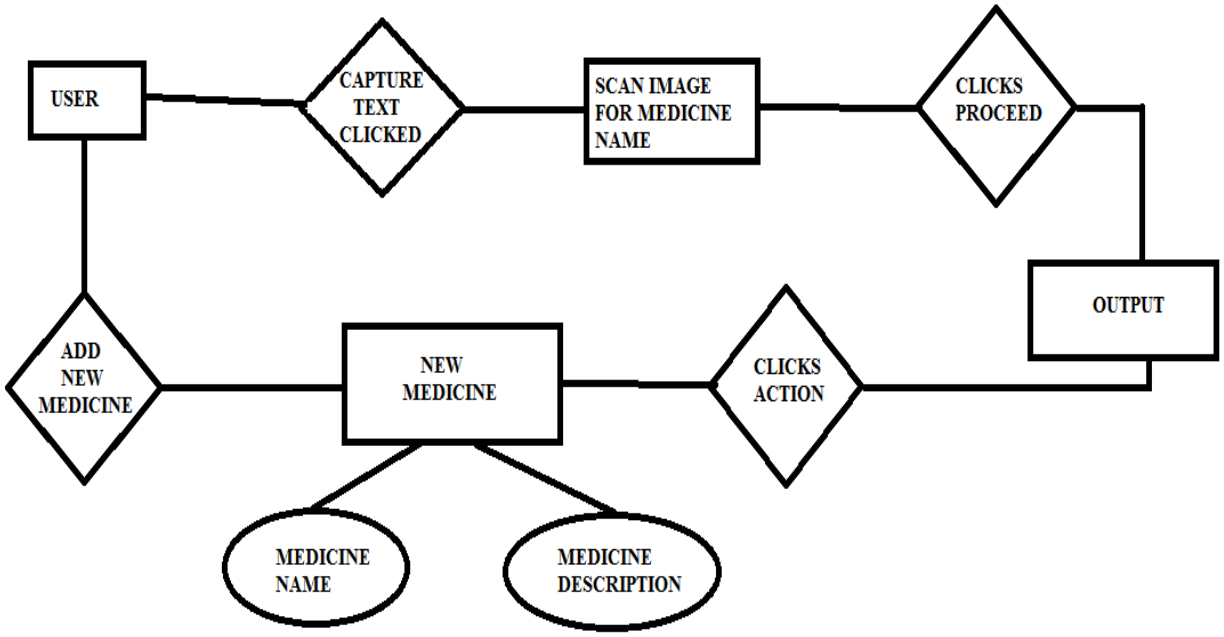
Use SharedPreferences to save and retrieve primitive data types in key-value pairs (boolean, float, int, long, strings). As of Honeycomb (API 11, Android 3.0) you can also save Set<String> sets of String values.

Entity Relationship Diagram

Entity-Relationship Diagram is a graphical representation of entities and their relationship to each other.

It describes how data is related to each other. An entity is a piece of data- an object or a concept about which data is stored. A relationship is how the data is shared between entities. In E-R Diagram, there are 3 main Components:

Symbol	Name		Description
	Entity		An entity can be any object, place, person or anything.
	Attribute		An Attribute Describes a property or characteristics of an entity.
	Relationship		A Relationship Describes relation between entities.



The Entity-Relationship (ER) model was originally proposed by Peter in 1976 [Chen76] as a way to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represents data objects. Since Chen wrote his paper the model has been extended and today it is commonly used for database design For the database designer, the utility of the ER model is:

- It maps well to the relational model. The constructs used in the ER model can easily be transformed into relational tables.
- It is simple and easy to understand with a minimum of training. Therefore, the model can be used by the database designer to communicate the design to the end user.

• Connectivity and Cardinality

The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many. A one-to-one(1:1) relationship is when at most one instance of a entity A is associated with one instance of entity B. For example, "employees in the company are each assigned their own office. For each employee there exists a unique office and for each office there exists a unique employee.

- A *one-to-many* (1:N) relationships is when for one instance of entity A, there are zero, one, or many instances of entity B, but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is a department has many employeeseach employee is assigned to one department
- A many-to-many(M:N) relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A. The connectivity of a relationship describes the mapping of associated

• ER Notation

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation. The original notation used by Chen is widely used in academics texts and journals but rarely seen in either CASE tools or publications by non-academics. Today, there are a number of notations used, among the more common are Bachman, crow's foot, and IDEFIX.

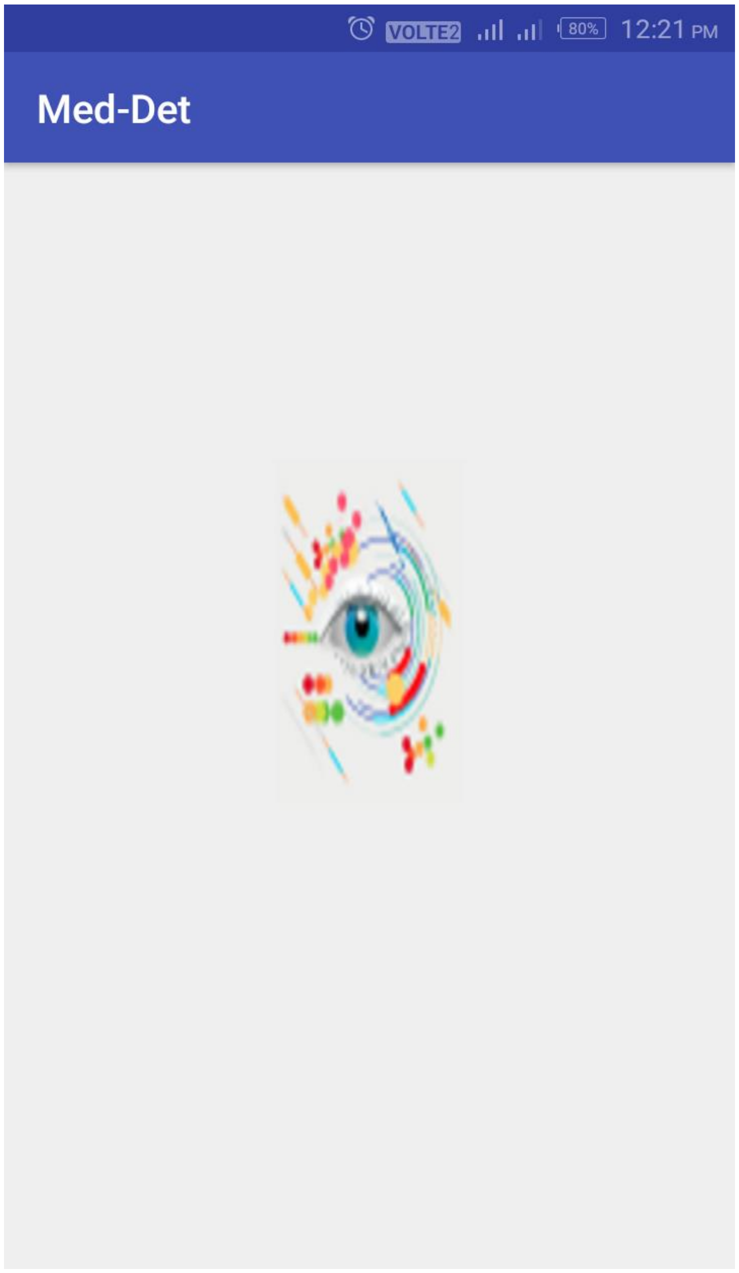
All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of a connection. The notation used in this document is from Martin. The symbols used for the basic ER constructs are:

- **Entities** are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.

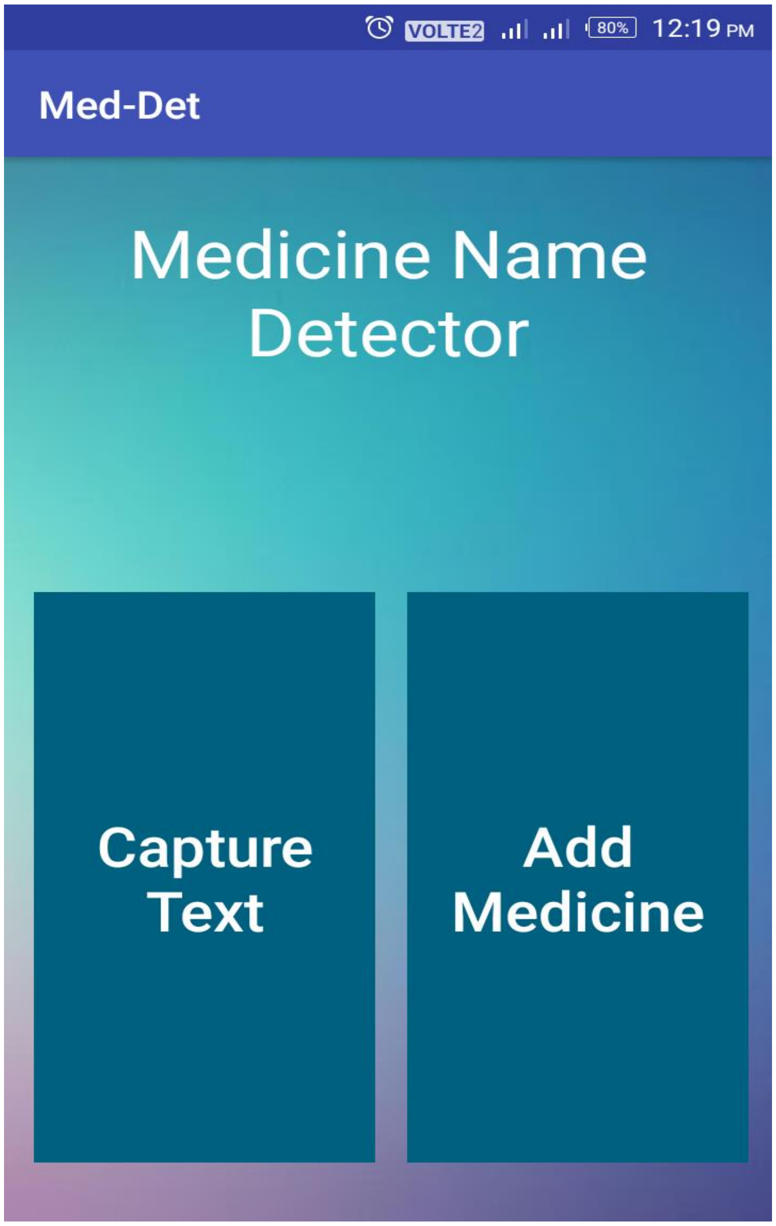
- **Relationships** are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs
- **Attributes**, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.
- **Cardinality** of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.
- **Existence** is represented by placing a circle or a perpendicular bar on the line. Mandatory existence is shown by the bar (looks like a 1) next to the entity for an instance is required. Optional existence is shown by placing a circle next to the entity that is optional

7. APP GUI

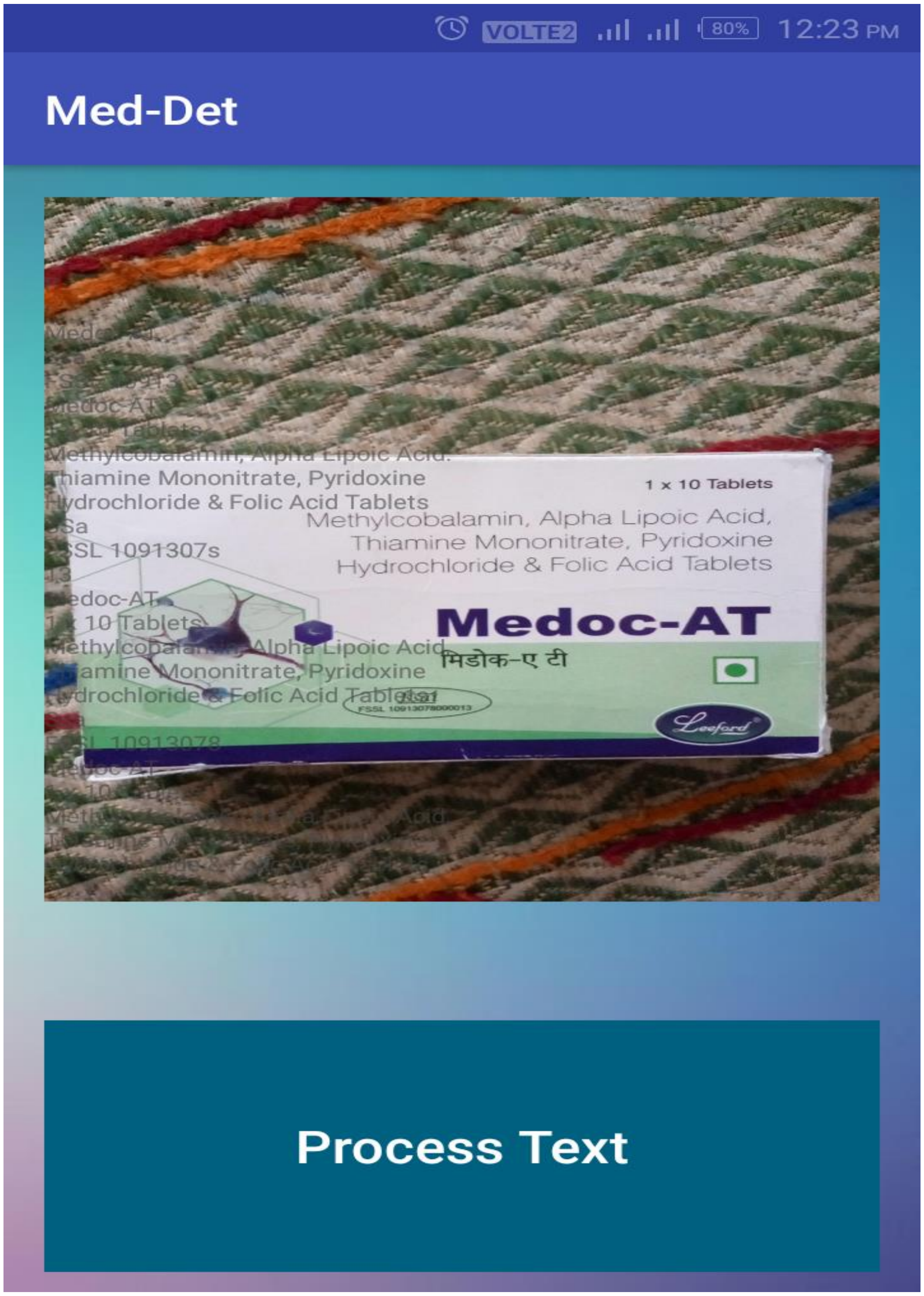
- **SPLASH SCREEN**



- **HOME SCREEN**

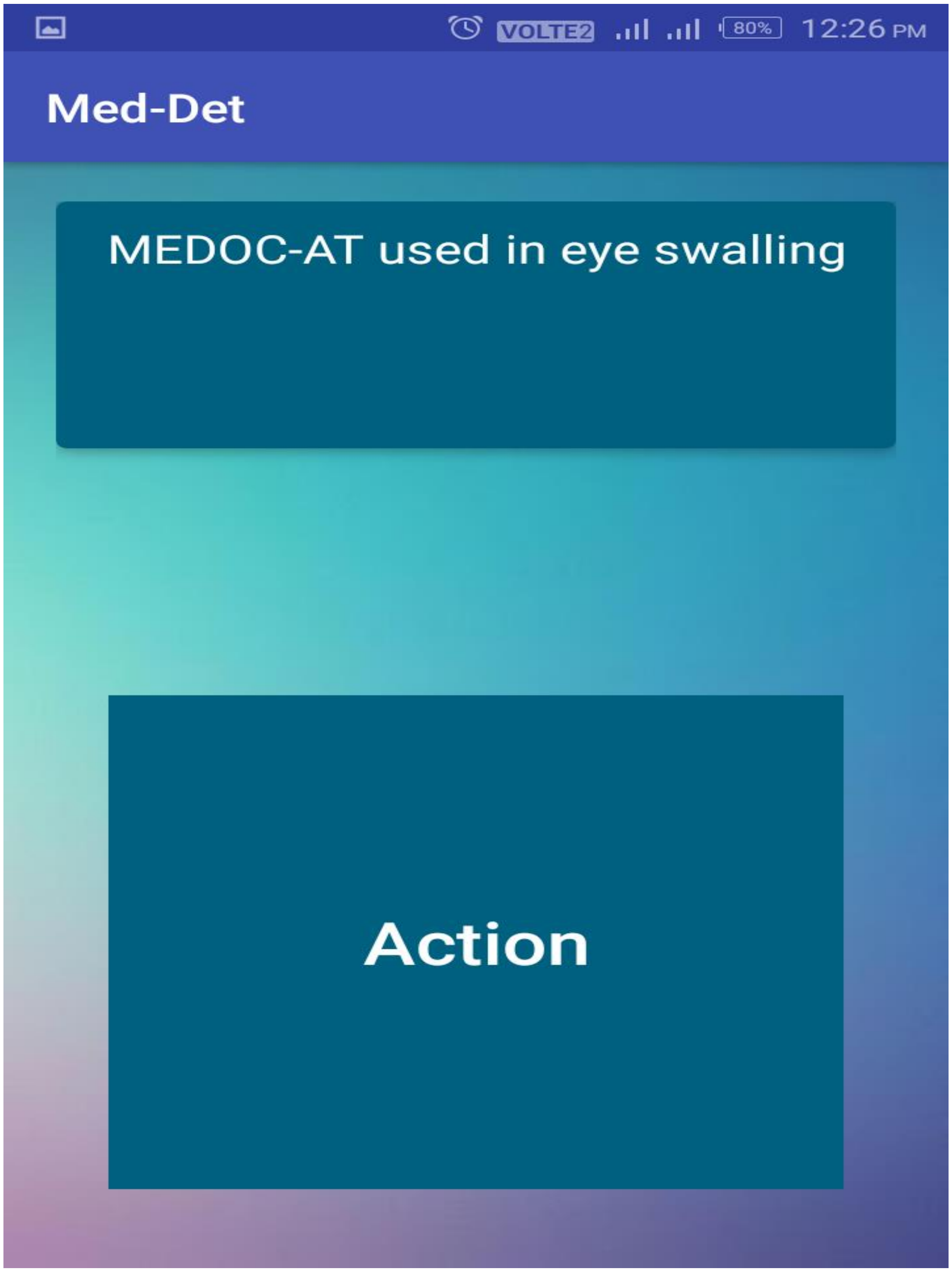

VOLTE2
80%
12:19 PM
Med-Det
Medicine Name Detector
Capture Text
Add Medicine

- **WHEN USER CLICKS CAPTURE TEXT BUTTON**



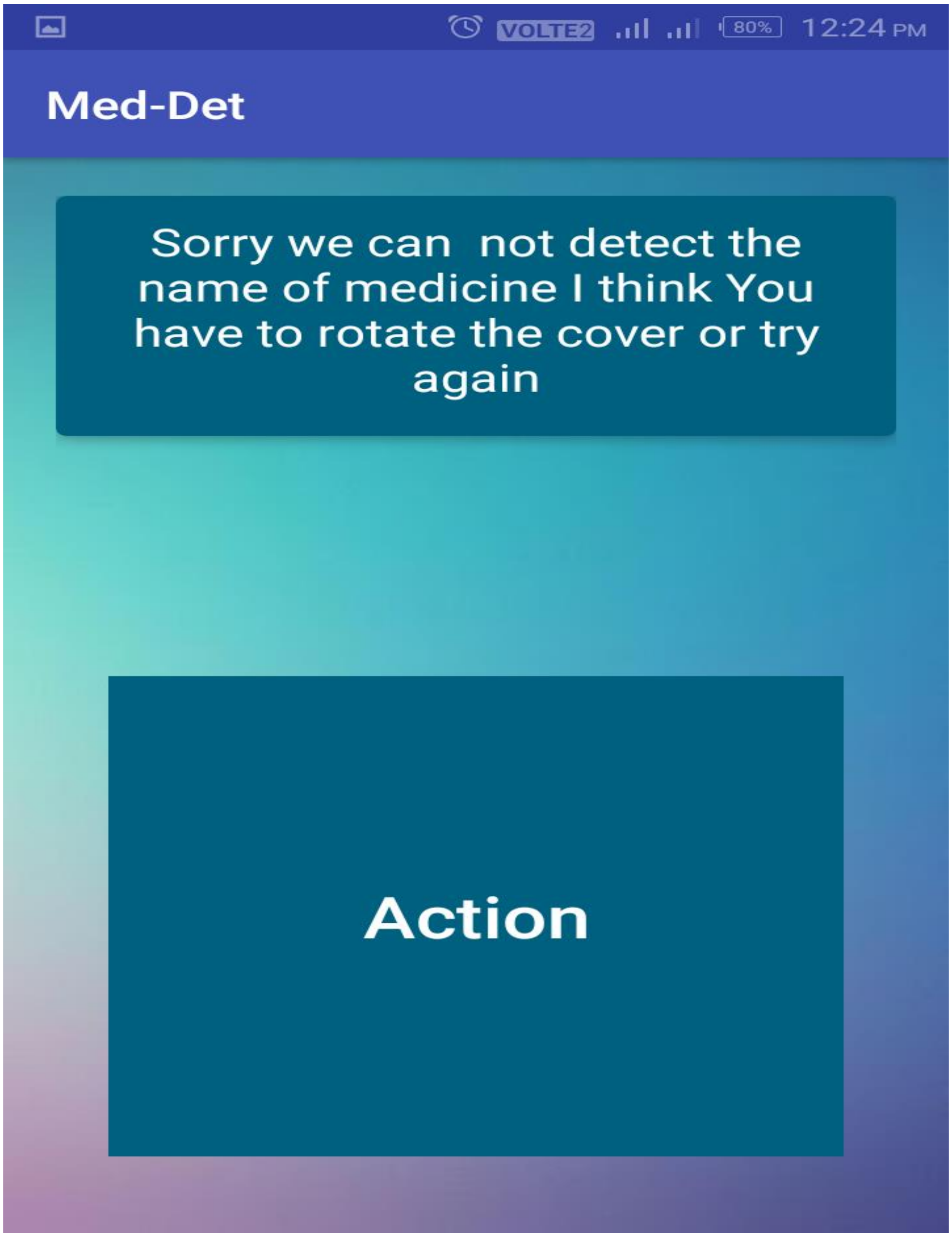
Medoc-AT
1 x 10 Tablets
Methylcobalamin, Alpha Lipoic Acid,
Thiamine Mononitrate, Pyridoxine
Hydrochloride & Folic Acid Tablets
FSSL 1091307s
Medoc-AT
1 x 10 Tablets
Methylcobalamin, Alpha Lipoic Acid
Thiamine Mononitrate, Pyridoxine
Hydrochloride & Folic Acid Tablets

- **AFTER ACCURATE DETECTION OF MEDICINE NAME**



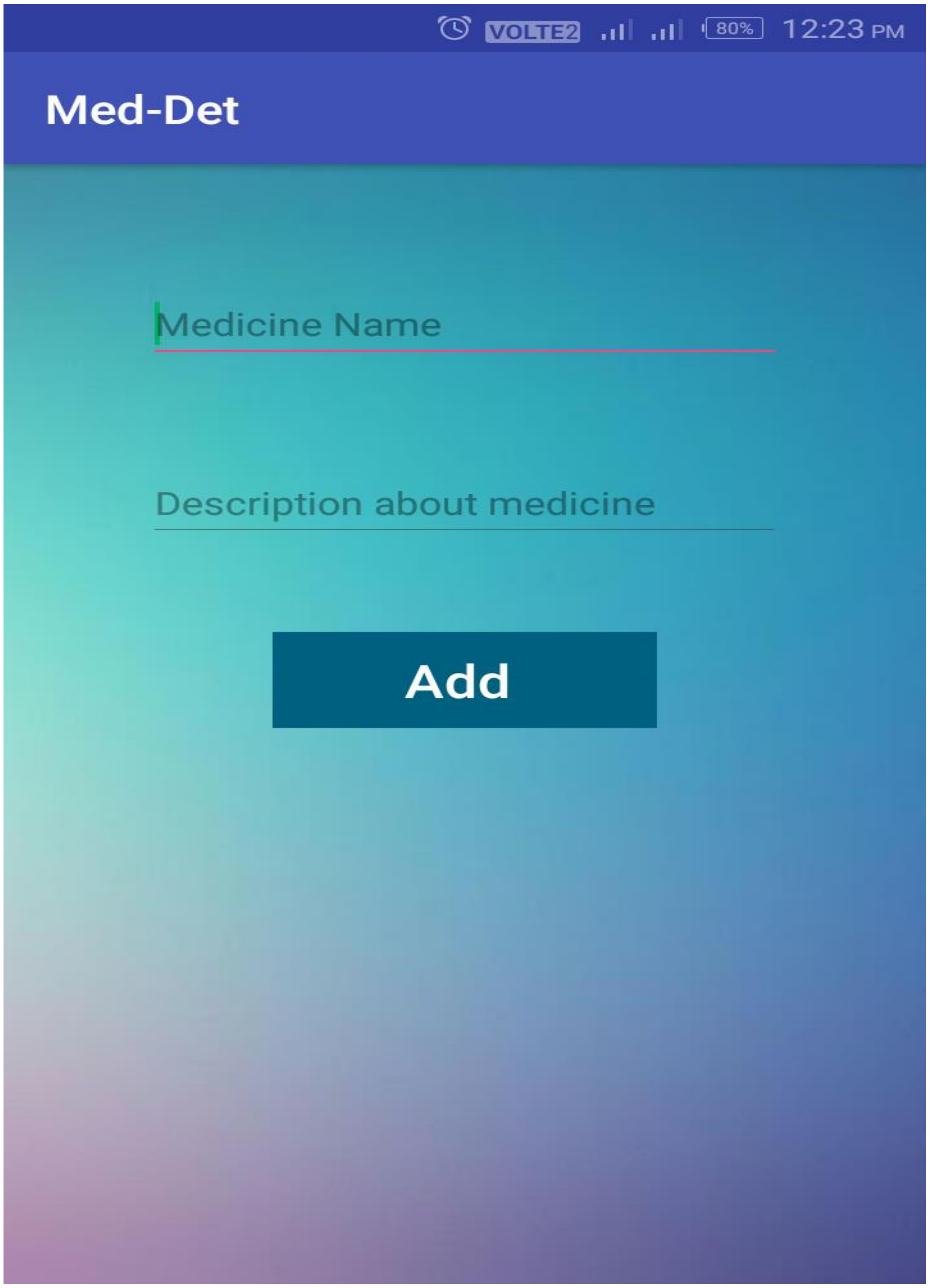
MEDOC-AT used in eye swalling

- **WHEN MEDICINE NAME IS NOT DETECTED PROPERLY**

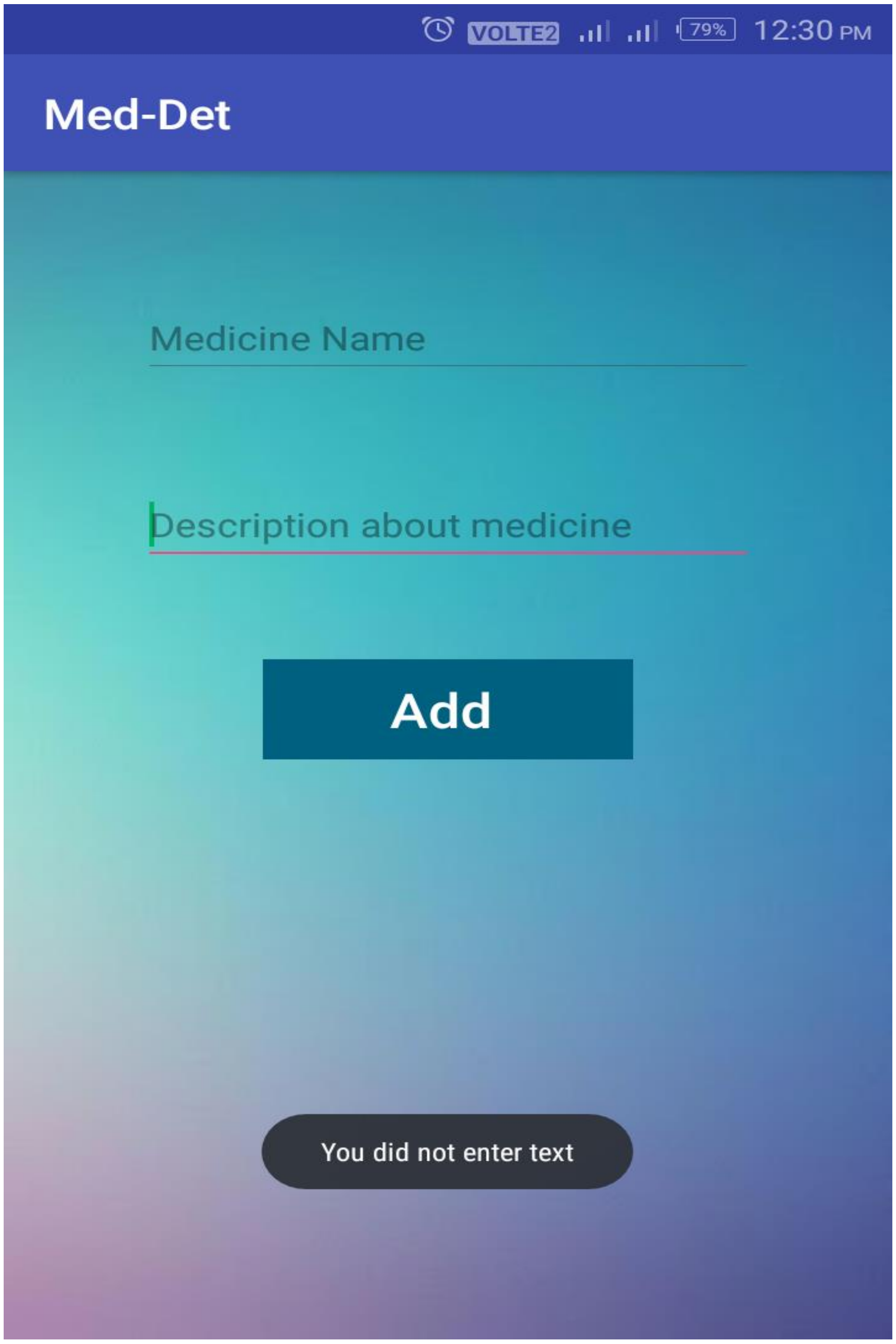


Sorry we can not detect the name of medicine I think You have to rotate the cover or try again

- **WHEN USER CLICKS ADD MEDICINE BUTTON**

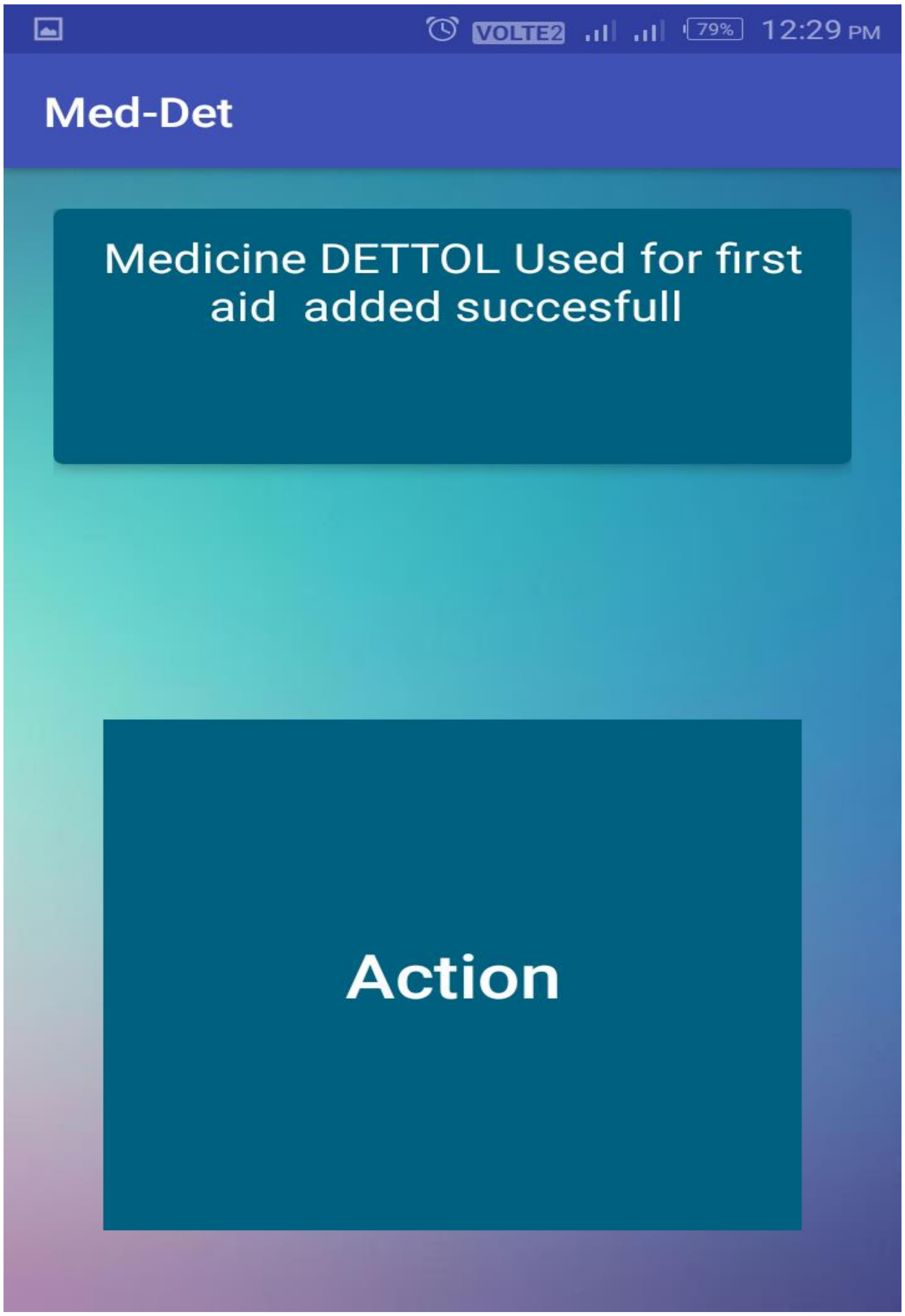


- **WHEN USER PROCEEDS WITHOUT ENTERING TEXT**



You did not enter text

- **WHEN A NEW CUSTOMIZED MEDICINE IS ADDED SUCCESSFULLY**



Medicine DETTOL Used for first aid added succesfull

8. CONCLUSION

This project report presents the development of the project with Camera Reading for Blind People , considering OCR and TTS stages, to create an application that was used for detection of name on a medicine.

An analysis was made regarding the OCR and TTS technologies that were used in the development of the application, in order to know the methods behind those, and to understand in greater detail the mechanisms that perform optical character recognition on images and speech synthesis of texts.

The project "**Medicine Name Detector**" consisted of the construction of an application composed by several parts, integrating the system of image capture by the mobile device, which is used by an OCR framework for recognition of its text, which is then synthesized through a process of TTS.

Optimizations carried out for improving outcomes resulted in a more efficient application, capable of responding to the challenge set by the theme of the project: a camera that reads texts for the blind.

To improve the quality performance of the system, a pre-processing of the image was carried out before submitting it to optical recognition. After considering various possibilities of image treatment, the choice fell on the use of a combination of filters (CIColorControls and CIColorMonochrome), which showed substantially improved results.

Another optimization applied to the project was an analysis of the optical recognition result in order to assess its usefulness as a presentable result. It was found that, in many situations, due to incorrect OCR reading, the final result represented a set of meaningless characters, but which represented text, consequently the TTS performed their synthesis, which caused an upsetting situation to the user. Thus, a function that uses regular expressions to analyze the result of OCR recognition has been developed; if the result presents a significant percentage of special characters, the system will indicate that there was a reading error and it will ask the user to repeat the process.

Although the final application does not represent a fully usable and totally reliable solution for blind people in daily life yet, the developed program and the obtained results can already be considered interesting, verifying that in some images, the ideal conditions being met, the results are quite interesting, occurring recognition and consequent reading, both very effective and consistent with the original text.

The final result achieved is not perfect, since it has shortcomings regarding the recognition of images in real time because the recognizing process of large and complex texts is sometimes slow. Also, the fact that the process of capturing the images does not provide an automatic system to aid the user to orient the image capture as correctly as possible presents itself as a limitation of this application, since the application is aimed at blind people, who may have more difficulties in accomplishing this task. However, this limitation can and should be targeted for improvement in future work.

The research, implementation and optimization developed allowed the design of a free application that is already in a state of possible use, even with the limitations referenced above, allowing the reading of texts, provided that the light conditions are the ideal for image recording and the equipment is properly directed to the text you want to listen to so that recognition and reading are as satisfactory as possible.

9. FUTURE ENHANCEMENTS

The app runs on any Android phone, and it is cheaper than those expensive item readers out there on the market. The app is simple to use and all it needs is for the user to take a picture and let the app do its magic. Scan Life Barcode and QR Reader can read UPC and QR codes. Once a code is scanned, the app reads the embedded string as a QR code. This is certainly useful for people who have a hard time shopping for items or buying stuff due to their impairment.

It affects almost every daily activity like walking, driving, reading and recognizing objects, places.

The basic techniques that are used and developed in these fields are more or less identical. Combining techniques in these fields can be applied in various applications like Artificial Intelligence (AI), Pattern/Object Recognition, Imaging, Machine Learning, Signal Processing, Neurobiology, etc. Among these applications, pattern recognition and image processing can be used to aid blind people in daily tasks by extracting information from outside environment.

To satisfy the need of blind community, it is necessary to develop device with smart algorithm to extract information from surroundings. Computer vision and mobile computing are powerful tools with great potential to enable a range of assistive technologies to aid growing population of visually impaired people.

Development in this field duplicates the abilities of human vision by electronically perceiving and understanding an image. Image processing, image analysis and machine vision are most closely related fields to computer vision

10. REFERENCES

Documentation for android api text-to-speech

https://developer.android.com/reference/android/speech/tts/TextToSpeech

Documentation for android api image to text

https://developers.google.com/vision/android/text-overview

Documentation for android api XML

https://developer.android.com/reference/android/util/Xml

Documentation for android api sharedpreferences

https://developer.android.com/training/data-storage/shared-preferences

Documentation for android api GUI

https://developer.android.com/guide/topics/ui/

Camera Reading for Blind People research paper

https://www.researchgate.net/publication/275231020_Camera_Reading_for_Blind_People

EIKVIL, LINE. OCR Optical Character Recognition, 1993

THOMAS, S. Natural Sounding Text-To-Speech Synthesis Based On Syllable-Like Units, Department Of Computer Science And Engineering Indian Institute Of Technology Madras, 2007.

Zhou, S.Z., Open Source OCR Framework Using Mobile Devices, Multimedia on Mobile Devices 2008. Edited by Creutzburg, Reiner;