An Efficient Bar Code Recognition Engine for Enabling Mobile Services

Author(s):
Adelmann, Robert

Publication Date:
2011

Permanent Link:

https://doi.org/10.3929/ethz-a-6665246 →

Diss. ETH Nr. 19721

An Efficient Bar Code Recognition Engine for Enabling Mobile Services

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by

Robert Adelmann
Diplom-Informatiker, Albert-Ludwigs-University Freiburg
born October 01, 1977
citizen of Germany

accepted on the recommendation of
Prof. Dr. Friedemann Mattern, examiner
Prof. Dr. Elgar Fleisch, co-examiner
Prof. Dr. Michael Rohs, co-examiner

2011

Abstract

In the area of pervasive computing, mobile phones have evolved into attractive development platforms that show considerable potential when it comes to bridging the often-cited gap between the real and virtual world. They are ubiquitous, highly mobile, provide significant computing power, and increasingly also offer an abundance of built-in sensors. With the general availability of smartphones and affordable data rates, consumers are beginning to use their mobile phones to interact with physical products found in stores in order to access product-related information and services. To support this interaction, consumer-oriented mobile applications require a fast and convenient way to identify products. Even though Near Field Communication (NFC) and Radio-Frequency Identification (RFID) technology is very promising for that purpose, the widespread use of RFID tags on retail products remains unlikely for the next years. In contrast, bar codes are ubiquitous on virtually all packaged consumer goods world-wide.

Recognizing bar codes with mobile phones induces many challenges, however. Blurry images from cameras without autofocus, but also shadows and glare, low video-image resolutions and limited computing power on many mass-market phones are just some of the inherent difficulties. Furthermore, developing applications for mobile phones still requires considerable expertise, despite the fact that mobile phones have evolved into attractive development platforms over the past years.

This thesis addresses these issues with two main contributions: First, it presents the design and implementation of a bar code recognition method for mobile phones that addresses the aforementioned challenges and, in particular, is capable of recognizing bar codes in blurry images. It provides a comparison with existing mobile bar code recognition engines on mobile phones and shows that the method presented outperforms other solutions in terms of scan speed and accuracy. The second contribution consists of a rapid prototyping environment for mobile phones that enables even non-experts to develop novel mobile services that leverage the bar code recognition in a fast and easy way.

Kurzfassung

Im Bereich des Pervasive Computing haben sich Mobiltelefone mittlerweile zu attraktiven Entwicklungsplattformen entwickelt, die das Potential besitzen, den oft zitierten „Graben zwischen der realen und der virtuellen Welt“ zu überbrücken: Sie sind allgegenwärtig, portabel, verfügen über erhebliche Rechenleistung und zunehmend auch über eine Vielzahl integrierter Sensoren. Durch die Verbreitung von Smartphones und die Verfügbarkeit von günstigen Datentarifen nutzen Konsumenten die Geräte zunehmend, um mit Produkten zu "interagieren" und Informationen sowie Dienste zu diesen abzurufen. Für solch eine Form der mobilen Interaktion ist eine Methode zur einfachen und schnellen Identifikation von Produkten unerlässlich. Obwohl sich hier Technologien wie Near Field Communication und RFID aufgrund ihrer zahlreichen Vorteile anbieten, ist ein grossflächiger Einsatz von RFID-Tags auf Produkten in den nächsten Jahren nicht zu erwarten. Im Gegensatz dazu sind Strichcodes heute auf nahezu allen Handelsgütern weltweit verbreitet.

Die automatische Erkennung von Strichcodes mit Mobiltelefonen beinhaltet allerdings zahlreiche Herausforderungen: Unscharfe Bilder aufgrund von Kameras ohne Autofokus, aber auch Schatten und Glanzpunkte, geringe Bildauflösungen oder beschränkte Ressourcen auf vielen Geräten stellen nur einige der Schwierigkeiten dar. Obwohl sich Mobiltelefone in den letzten Jahren zu attraktiven Entwicklungsplattformen entwickelt haben, erfordert auch die Anwendungsentwicklung immer noch viel Zeit und Erfahrung.

Die vorliegende Dissertation liefert zwei Beiträge, um diesen Herausforderungen zu begegnen: Der erste Beitrag besteht in der Entwicklung und Implementierung eines Verfahrens für die Erkennung von Strichcodes auf Mobiltelefonen, welches die genannten Schwierigkeiten adressiert und insbesondere in der Lage ist, Strichcodes auch in sehr unscharfen Bildern zu erkennen. Das vorgestellte Verfahren wird im Rahmen einer Nutzerstudie und einer detaillierten Analyse mit bestehenden mobilen Lösungen zur Erkennung von Strichcodes verglichen. Es zeigt sich, dass die in dieser Arbeit präsentierte Erkennungstechnologie flexibler, schneller und genauer als alternative Lösungen ist. Der zweite Beitrag besteht aus einer Software-Entwicklungsumgebung, welche es selbst mit der Mobiltelefonprogrammierung nicht vertrauten Personen ermöglicht, schnell und unkompliziert neue, auf der Strichcodeerkennung basierende Dienste zu entwickeln.

Contents

1 Introduction

1.1 Motivation

The concept of “bridging the gap” between physical and virtual worlds by means of linking real objects to virtual information and services is well known. Since Wellner presented the “DigitalDesk” project in 1993 [1], numerous other researchers have explored this concept, including Barret and Maglio [2] or Kindberg et al. in their “Cooltown” project [3].

One specific type of real-world objects for which this augmentation with virtual information and services seems very promising are the multitude of products sold world-wide. Whereas a product's packaging lists already extensive information, such as the product's ingredients or the best-before date for fresh produce, several limitations exist. The provided information is static and can therefore not be personalized for different consumers, space on the product packaging is limited, and, due to commercial considerations, the information that is provided is often biased. Retailers and product manufacturers might want to provide dynamic and more detailed information to consumers, e.g., background information about where and under which conditions a premium product was produced. In contrast, consumers might want to have access to information that is usually not present on packaging, such as details about whether a product contains genetically modified ingredients, independent product reviews, or price comparisons. Despite the availability of such information and even though such data might be highly relevant for certain user groups, having personalized, direct access to it in situations where it might be useful (e.g., while shopping) can be challenging. With the increasing proliferation of smartphones, this gap between the user and product-related data and services can be bridged [4, 5], and consumers are beginning to use their mobile phones to interact with physical products found in stores.

Figure 1.1 illustrates the prototype of a simple mobile application that we implemented; once the user defined a profile containing all substances to which she is allergic, holding the mobile phone in front of the bar code on a product gives her a simple answer to the question "Is that product fine for me?". Given that many such consumer-oriented mobile services are most useful when users are on their way somewhere (e.g., when standing in the aisle of a supermarket), a fast and simple way to identify products is essential. Manu-

ally entering a product's name or bar code number is time-consuming and likely to result in many errors [6].

There are several technologies for the automated recognition of products available. In particular the use of RFID technology for linking information to physical objects has been explored in numerous projects, e.g., by Want et al. [7] and Pohjanheimo et al. [8]. Furthermore, RFID is increasingly used in the retail industry throughout the supply chain, such as for tagging pallets of consumer goods. However, despite its benefits, today's tag costs and remaining interference issues with fluids or on metallic surfaces make it unlikely that RFID tags will appear any time soon on ordinary sales items, such as bottles or cans [9]. In contrast, bar codes are already present on most consumer items world-wide.

Figure 1.1 Screenshots of the "Allergy-Check" demo application.

In the past, recognizing standard bar codes with mobile phones required the use of special laser scanner attachments, and the optical recognition of codes with camera-equipped phones has been limited to two-dimensional (2D) codes, such as the Visual Code developed by Rohs [10]. Many 2D codes have been specifically designed to compensate for the low-resolution and poor-quality images obtained on consumer-grade phones. Whereas the use of 2D codes for product identification first requires the widespread use of such labels on sales items, the recognition of standard bar codes with camera-equipped mobile phones enables a multitude of mobile services, today.

World-wide, sales items are labeled with a bar code, camera-equipped mobile phones are ubiquitous, and there is an abundance of product-related information available. A fast and robust bar code recognition method targeted specifically at mobile phones is a central enabling technology for the large-scale deployment of such consumer-oriented mobile services.

1.2 General Challenges

While there seem to be many desirable applications possible, technical issues remain that hinder the development and adoption of mobile applications that offer information and services for retail products. In the following, we summarize the challenges inherent to the recognition of bar codes by mobile phones and the development of mobile applications.

1.2.1 The Mobile Bar Code Recognition Process

The optical recognition of bar codes on mobile phones in typical shopping environments and scenarios is challenging and differs considerably from recognizing bar codes in a controlled environment with perfect lighting. Reasons for this include:

- *Blurry images:* On mobile phones featuring no autofocus camera, almost all images taken at short distance are blurry, which considerably limits the installed base of supported mobile phones. However, even on devices with autofocus blurry bar codes are common; due to slow autofocus systems, hand movements by users or wrong focus points. The latter results for small or thin products often in a blurry image of the product itself including its bar code and a sharp background. Figure 1.2 shows some examples.
- *The variety of bar codes found on products:* Bar codes come in varying shapes, sizes and symbologies, including colored codes or codes printed on round, crumpled or otherwise distorted surfaces (see Figure 1.3).
- *Difficult lighting conditions:* Scanning bar codes in places such as stores, with various light sources, often results in uneven lighting, shadows or glare on codes.
- *The variety of mobile phone models:* When using mobile phones to recognize bar codes, one has to deal with the plentitude of different mobile phone models, including different optics, low image resolutions and limited resources like CPU and RAM on many phones.
- *User behavior:* User behavior will result in images in which bar codes are rotated or upside-down, very small, or have an angular perspective.

These inherent practical challenges complicate the recognition process on mobile devices. Section 2.1 provides more details regarding the most important effects.

Figure 1.2 Examples of blurry images: image of bar code (5410076027705) obtained from a mobile phone without autofocus camera (left image), blurry image obtained on a device with autofocus camera but a wrong focus point (upper-right image), and a slightly blurry image resulting from the fact that the phone's autofocus camera cannot focus on too close objects (lower-right image). All images can be correctly decoded with the bar code recognition method presented in this thesis.

Figure 1.3 Examples of images illustrating the variety of bar codes found on products: code on a crumpled surface with slight glare (upper-left image), thin bar code (upper-right image), bar code on round surface with transparent background (lower-left image), or bar code with low contrast (lower-right image)

1.2.2 The Mobile Application Development Process

There is an important gap between the manifold opportunities for creating novel mobile applications on one hand, and the knowledge-intensive and time-consuming mobile application development process on the other. This gap disregards an important source of creativity, as the possibilities exceed the current resources of mobile experts. A lot of users might be motivated to create novel types of mobile applications using the phones' increasing capabilities, but only few have the time and energy to handle the difficulties of mobile phone programming.

1.3 Contributions

The following section details the contributions made by this thesis to address the challenges highlighted above.

1.3.1 Bar Code Recognition Method

The main contribution of this thesis is a method for robust, real-time recognition of bar codes with mobile phones in realistic settings. The method presented outperforms existing systems in terms of recognition speed and accuracy. Furthermore, it is capable of recognizing bar codes in blurry images, such as those obtained on phones without an autofocus camera. This considerably increases the number of supported mobile devices.

In this thesis, existing challenges are examined in connection with recognizing bar codes using mobile phones, the recognition algorithm is presented, as well as proof-of-concept implementations on major mobile phone platforms. In order to compare our approach to existing bar code scanners for mobile phones, a user study and a detailed lab-analysis are presented that evaluate both the "real-world" performance as well as specific capabilities of scanners and illustrate the superiority of our approach. Based on this evaluation, general observations about users' scan behavior and guidelines for designing user interfaces for mobile bar code scanners are derived.

1.3.2 SPARK: A Rapid Prototyping Environment for Mobile Services

The second contribution of this thesis is SPARK, a rapid prototyping environment that facilitates the creation of novel mobile services. It allows non-

experts to create mobile applications that use bar code recognition in a fast and easy way.

The design and implementation of SPARK is presented. Furthermore, the results of two case studies in which SPARK has been used are shown: A graduate course on distributed systems in which more than 70 students used SPARK to develop mobile applications, as well as the development of a mobile phone-based product information platform – the "Product Advisor".

1.4 Thesis Outline

This thesis is structured as follows: Chapter 2 introduces challenges inherent to bar code recognition on mobile phones and relevant background information about the structure of bar codes. Chapter 3 presents the developed recognition algorithm and discusses related work. In Chapter 4, we present proof-of-concept implementations of our algorithm on three major mobile phone platforms (C++ Symbian, iOS and Android) and the relevant tools used to develop them, as well as to optimize the recognition method. Chapter 5 reports on our user study and scanner analysis. We also derive general observations about users' scan behaviors and present guidelines for designing user interfaces for mobile bar code scanners. Chapter 6 briefly discusses mobile services and alternative product identification technologies. Finally, we present our rapid prototyping environment for mobile services, SPARK, in Chapter 7 and conclude with a summary and discussion of future work in Chapter 8.

2 Background

This chapter covers relevant background information. Section 2.1 describes in detail the conditions and challenges when recognizing bar codes with mobile phones. The presented challenges are the basis for design decisions met in the recognition algorithm (Chapter 3). Section 2.2 continues to provide an overview of the bar code symbologies that are relevant in our context and introduces details about the structure of EAN13/UPC-A bar codes. This code symbology is used as a standard example throughout the description of the recognition algorithm.

2.1 Recognition Challenges

Recognizing bar codes that are printed on a white sheet of paper in sharp and high resolution images, with perfect lighting and a straight perspective on the code is fairly simple. However, when recognizing bar codes with mobile phones, conditions are rather different [11]. This section provides an overview of the various challenges inherent to the mobile services scenario, in which consumers use their mobile phones to identify products at various locations, e.g., stores. The challenges listed below are based on experiences gained from developing the bar code recognition algorithm and several demonstrator applications on multiple mobile phone platforms. Furthermore, we utilized feedback obtained from industry partners like the Metro-Future Store Initiative [12] or the Markant Group [13] that used the developed software in their own deployments, and from the user study presented in Chapter 5 of this thesis.

2.1.1 Blurry Images

One of the biggest challenges of mobile phone-based bar code recognition is related to the structure of most code symbologies, in which the information is encoded in bars of four (or more) different widths. Most recognition algorithms require these different bar widths to be detected reliably for successful decoding. Whereas the resolution of typical phone cameras of 640x480 pixels or higher is usually sufficient for this purpose [14], the sharpness of the obtained images is an issue. Many camera modules built into mobile phones have only fixed-focus lenses, which result in blurry images of very close ob-

jects. However, even in the case of mobile phone cameras with autofocus (AF), blurry images remain a problem. Examples of situations that result in blurry images include the following (see Figure 2.1):

- Blurry images due to phones with fixed-focus cameras. On mobile phones featuring no autofocus (AF) cameras, images of close objects will be blurry.
- Blurry images due to hand movements. This occurs especially in the case of dark environments, when long image exposure times are required because of to the small image sensors used in mobile phones.
- Blurry images because the camera cannot focus on close objects. Most AF-systems used in current phones are not able to focus on very close objects, which results in blurry images in the case of small bar codes.
- Blurry images due to a slow focusing process. AF-systems are often relatively slow, especially in case of the mechanical systems predominantly used today. This result in blurry images for some time after the user holds the mobile phone in front of a bar code or varies the distance to the code. Furthermore, AF-systems used in mobile phones are usually not able to determine the required AF-adjustments directly based on a single blurry image (like in case of reflex cameras). Focusing consists therefore not of a single, targeted correction of the camera focus, but of a complete and time consuming cycle through all focus ranges. However, the performance of AF-systems for mobile phones is expected to increase in the next years [15-18].
- Blurry images due to missing AF control. Not all devices featuring a built-in AF camera provide the software interfaces required to use or control the camera focus. For example, no AF control is possible from J2ME applications and C++ Symbian provided until recently only very limited AF control on many devices. Despite detailed functionality already defined in the according APIs on iOS, Android or QT, many features are not supported by today's devices. For example, to our knowledge it is currently not possible on any platform to set the camera focus manually on the closest focusable distance, which would be a good solution for our use case.[1]

[1] One reason for this limitation is the fact that phone manufacturers include camera modules in devices that encapsulate many features like autofocus control in the camera module itself.

- Blurry images due to the camera focusing on other objects. When users are scanning bar codes, the background will often feature other products (e.g., in stores) or be non-uniform. Especially if the product that should be recognized takes up only a small portion of the image, the phone's camera is sometimes not focusing on the product itself, but ensures the rest of the image is sharp. Only recently some devices allow applications to control focus modes and the exact point or area where to focus.
- Since today's mechanical AF-systems require energy and due to the limitations stated above, focusing is often not performed continuously, which results in a less responsive camera focus and therefore often blurry images.

Figure 2.1 Examples of blurry images: Out-of-focus blur on device without AF camera (upper-left image), blurry image due to hand movements (upper-right image), blurry image due to wrong focus point (lower-left image), and slightly blurry image because the AF camera can't focus on too close objects (lower-right image).

Figure 2.2 Examples of the variety of bar codes: Code with transparent background on round surface (upper-left image), colored code on crumpled surface (upper-right image), non-uniform image background including an additional bar code (lower-left image), and a thin bar code (lower-right image).

2.1.2 Large Variety of Bar Codes Printed on Products

Despite exiting standards and recommendations for bar codes printed on products (e.g., the ISO specification for bar codes [19] that specifies physical properties of codes printed on sales items, in order to ensure compatibility with a wide variety of scanners), in reality a vast variety of code shapes and sizes are found. On reason for this being the usually very limited and valuable space on product packages[2]. Examples for difficulties that arise from this fact include the following (see Figure 2.2):

[2] "*Some of our health and beauty care products have very small packaging, and strict regulatory requirements mean we have to put a certain amount of text on the boxes, no matter how small they are – and that means less panel space for communication with our customers.*" Bud Babcock, Procter & Gamble [106].

- On many grocery items, codes are printed on flexible and crumpled surfaces.
- Codes are colored and offer sometimes a very low optical contrast. However, due to differences in the reflection properties of the used paint, these codes are still well readable with laser scanners.
- Codes are printed on curved surfaces, complicating the recognition.
- Codes are available in a large variety of sizes ranging from very small and thin codes to very large ones.
- Finding the bar code that should be recognized in an image in the first place is not always trivial. Small codes, multiple close codes and other elements on the product packaging like text or other products visible in the background complicate this task.
- Some codes do not correspond to the standard for bar codes issued by GS1. For example, the amount of white space at the beginning or the end of codes is too short, or codes are printed too small.
- Multiple bar code symbologies are used on products, and distinguishing the symbology at hand can be challenging in case of blurry images.

2.1.3 Lighting Conditions

When recognizing bar codes in places like stores and not in a controlled environment, imperfect lighting becomes another issue. Even if the lighting differences in images on the phone's screen look minor, effects on the image can be severe. Challenges related to lighting include the following (see Figure 2.3):

- Glare on codes. This typically occurs due to lights installed on the ceiling of stores and due to codes that are printed on reflective surfaces.
- Shadows on codes. For example, when users holds the mobile phone over a product's bar code in order to scan it, in combination with spotlights on the ceiling. In such situations, the user's hand and the mobile phone often cast shadows on the code.
- Too bright images due to the automatic exposure control on mobile phones. For example, if the user moves the mobile phone from a dark spot (camera pointing at a dark floor, or the phone was in a pocket) to a brighter environment. However, after a few seconds, the exposure control is usually able to adjust image brightness, making too bright images not a major problem.
- Too dark images. In practice, this is also no major problem, except for rather dark places like bars or clubs.

Figure 2.3 Examples of a bar code with shadow (left image) and glare (right image).

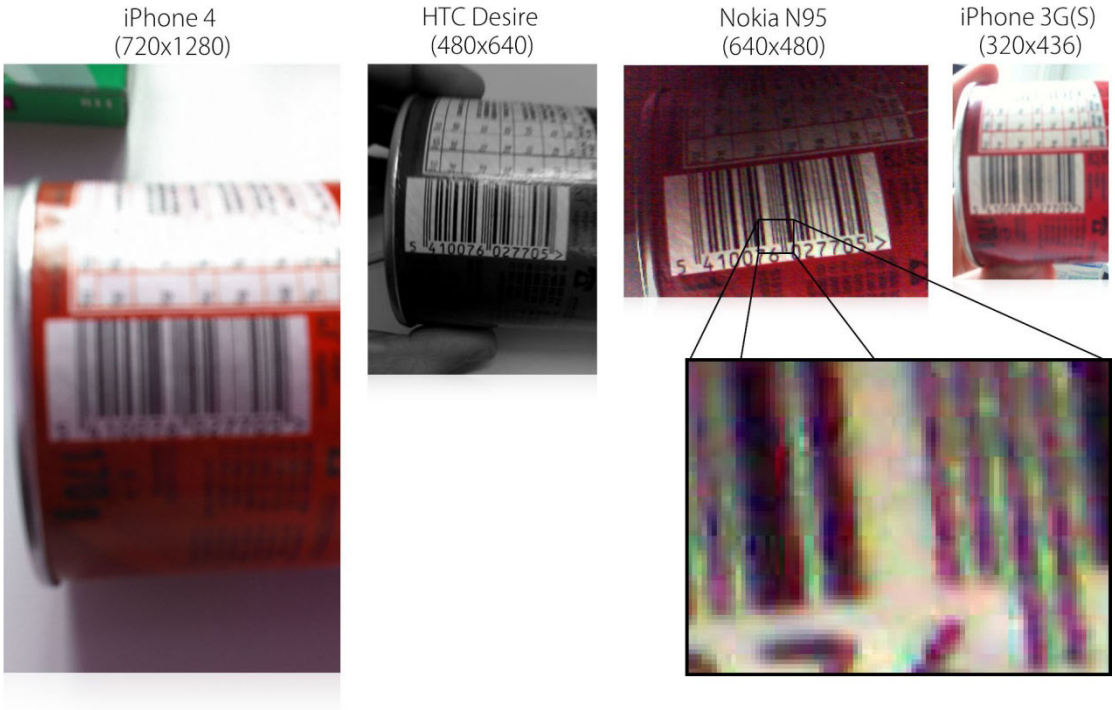


Figure 2.4 Examples of the variety of available image formats and resolutions on different mobile phones as well as image artifacts from image compression.

2.1.4 Large Variety of Mobile Phone Models

In contrast to specialized image recognition systems used for industry applications, a bar code recognition method intended for consumers has to address the large variety of different mobile phone models available. This includes different available resources, camera optics, or functionalities (APIs) for controlling the phone's cameras and accessing live video images. Challenges arising from mobile phones in general and the variety of available devices and software platforms include the following:

- Limited processing power, especially on cheaper phones, but also on many smartphones, e.g., Nokia N-Series devices.
- Limited run-time memory. Despite some devices having up to 512 MB of usable application memory, memory consumption remains a problem. For example, because of system limitations related to multitasking on the iPhone, applications are expected to require much less memory or warnings and crashes are likely to occur.
- Limited application size. The size of the complete recognition engine bundle, which includes the program code and other resources that should be distributed, is critical. Since the recognition engine is usually included in other applications, it adds to the final size of the complete distributable, and even a few megabytes count in the case people install applications directly over the mobile phone network, e.g., in a store or after seeing a commercial.
- Different camera modules and APIs result in a variety of image formats, orientations and resolutions. For example, resolutions of live images accessible for image processing include 160x120 pixels on most J2ME devices, 320x270 pixels on Nokia Smartphones like the Nokia X7 (despite its built-in 8MP camera), or up to 1280x720 pixels on the iPhone 4. On Android devices like the HTC Desire, images obtained from the camera are not encoded in the RGB, but in the YUV color format. In this case, we use the pixel-brightness (luminance) values directly and omit color information in order to save the time required for color model conversions.
- Some mobile phones automatically apply image processing steps before images can be accessed by an application. For example, edge sharpening or compression algorithms that result in image artifacts (see Figure 2.4).
- Imperfect lenses on mobile phones result in images with varying local sharpness and distortions, especially in the case of close objects.

2.1.5 User Behavior

Another factor that has to be considered and complicates bar code recognition on mobile phones is the behavior of users themselves. There are several potential challenges arising from user behavior:

- Users holding the phone too close to codes so that the code is only partially contained in the image (e.g., sometimes the case in very blurry

images obtained on phones without autofocus camera) or holding the phone too far away, resulting in very small codes in the image.
- Users holding the phone not directly above the code that should be recognized but from an angular perspective, which results in distorted codes.

2.2 Bar Code Basics

This section covers the relevant details about commonly used bar code symbologies and their usage on products, in order to establish the required background knowledge for the recognition algorithm covered in Chapter 3. There is a variety of different bar code symbologies available, but the most relevant symbologies in a retail context are the following: The European Article Number (EAN) codes EAN13 and EAN8, the Universal Product Codes (UPC) UPC-A and UPC-E, and for certain use cases Code39 and EAN128 codes. Figure 2.5 shows example codes and Table 2.1 briefly compares and presents typical use cases of these symbologies. Our recognition engine is able to recognize all of these symbologies.

The following section covers the EAN13 and UPC-A bar code symbologies in more detail. These are the most common symbologies used to tag products and will serve as a standard example when describing the recognition algorithm in Chapter 3. Other symbologies will be mentioned where relevant. Details about the information theory underlying bar codes can be found in the journal article "Fundamentals of bar code information theory" by Swartz et al. [20].

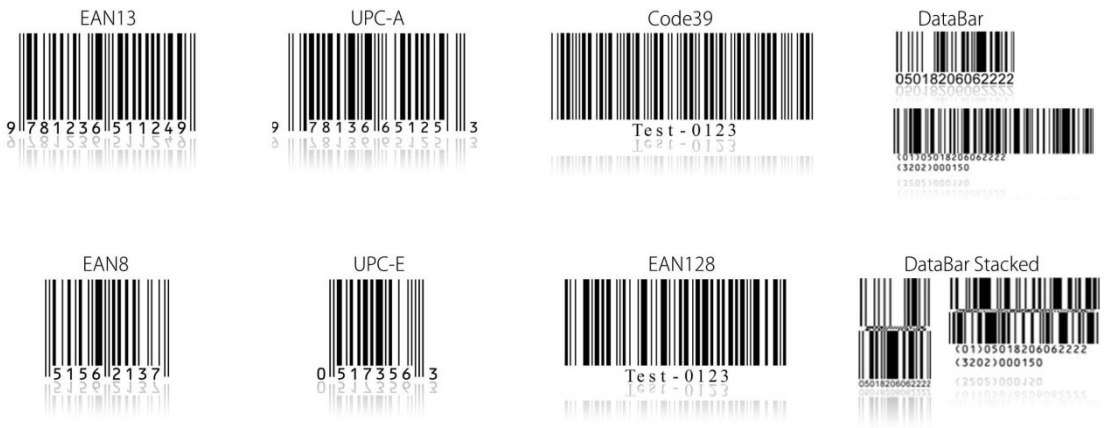


Figure 2.5 Most frequently used bar code symbologies (EAN/UPC) and future codes (DataBar).

Table 2.1 A brief comparison of relevant bar code symbologies (in their standard version without extensions) and typical use cases:

Symbology	Use Case
EAN13	Standard symbology used on most products that encodes 13 digits. The EAN13 (European Article Number) code set is a superset of the UPC-A (Universal Product Code).
UPC-A	The UPC-A code symbology is widely used in the US. Its structure corresponds to that of EAN13 codes with a leading 0.
EAN8	EAN8 codes encode 8 digits and are typically used for small products or products featuring a strong curvature (in order to minimize code distortions). Compared to UPC-E codes, they are more common in Europe. The structure is similar to EAN13 and UPC12 codes, featuring the same start-, end- and middle-pattern and the same encoding of numbers.
UPC-E	UPC-E codes also encode 8 digits, have similar use cases like EAN8 codes, but are more common in the US. However, their structure (start- and end-pattern, digit encodings and check digit calculation) differs from the other symbologies and allows the digits to be encoded in fewer bars compared to EAN8 codes.
EAN128	EAN128 codes allow not only for the encoding of numbers, but also alpha-numeric characters (Code128 variant) and are of variable length. EAN128 codes are often used for couponing, but also for various other tasks, e.g., for encoding an electronic device's serial number or for automated document or packet management. Their structure differs from the other symbologies and allows, for example, the switching between different encoding tables and alphabets inside the same code.
Code39	Code39 codes can also encode alpha-numeric characters and are of varying length. They are often used for in-house labeling of products, e.g., the re-labeling of products with regular bar codes with in-house identifiers, for special promotion offers or other use cases. Their structure differs from the other symbologies. Compared to the EAN/UPC family of codes that encode digits in bars of four different widths, Code39 encodes digits in bars of only two different widths. Compared to EAN128 codes, this usually results in much larger codes for the same information.

Figure 2.6 Structure of an EAN13 bar code.

2.2.1 Details of EAN13 Bar Codes

When mentioning the EAN13 code symbology, we will refer to both EAN13 codes and UPC-A codes, since the latter feature the same encoding. EAN13 bar codes encode 13 digits. Like all bar codes of the EAN/UPC family, each digit is encoded in four fields (two white fields = *spaces* and two black fields = *bars*)[3]. Both spaces and bars can have four different widths. They can be one, two, three or four units wide. Taken together, the two bars and two spaces encoding a digit have always a width of seven units. A bar code is structured like shown in Figure 2.6. In between a silent area of spaces (usually 5 or more units wide) and start- and end-patterns (a pattern consists of a bar, space, and a bar, each one unit wide) are twelve encoded digits (six *left bar code digits* and six *right bar code digits*) divided by a middle pattern (a pattern consisting of a total of five alternating spaces and bars, each one unit wide). The complete bar code consists of a total of 59 different bars and spaces that have together a size of 95 units. A left bar code digit can be encoded in two ways, either with *even* or with *odd* parity (see Table 2.2). The first digit is not direct-

[3] Colored bar code bars and backgrounds are common too, but the colors usually encode no additional information.

ly encoded in spaces and bars, but in the parity pattern that is contained in the encoding of the six left code numbers and shown in Table 2.3.

The encoding table for the left bar code digits is different from the encoding table for the right bar code digits (see Table 2.2). This allows us to distinguish consistently between the left and right code side, even in the case bar codes are read up-side down and there are no human readable digits printed below the code.

Table 2.2 Encoding table for the left and right bar code digits (1 corresponds to a bar and 0 to space):

Digit	Left Digit Odd Parity	Left Digit Even Parity	Right Digit
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1010000
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

Table 2.3 Encoding table for the first bar code digit that is encoded in the parity pattern of the six left code numbers:

Digit	Parity Pattern of Left Digits					
0	Odd	Odd	Odd	Odd	Odd	Odd
1	Odd	Odd	Even	Odd	Even	Even
2	Odd	Odd	Even	Even	Odd	Even
3	Odd	Odd	Even	Even	Even	Odd
4	Odd	Even	Odd	Odd	Even	Even
5	Odd	Even	Even	Odd	Odd	Even
6	Odd	Even	Even	Even	Odd	Odd
7	Odd	Even	Odd	Even	Odd	Even
8	Odd	Even	Odd	Even	Even	Odd
9	Odd	Even	Even	Odd	Even	Odd

The product identifier $d_0 \dots d_{12}$ that is encoded in the bars and spaces is also structured. Each identifier consists of a system digit d_0, followed by a 2-3 digit country code, a 4-5 digit manufacturer code, a 5 digit product code, and a single checksum digit d_{12}. The checksum digit d_{12} can be used to verify that the product identifier has been decoded correctly. It has to hold:

$$d_{12} = \left(10 - \left(\sum_{x=0}^{5} d_{x\cdot 2} + d_{x\cdot 2+1}\right) mod\ 10\right) mod\ 10$$

However, despite this structure, little reliable information can be obtained directly from the product identifier. For example, the country code of a product identifies the country of origin of the company that bought the according number range from GS1 and does not allow direct conclusions about the country of origin of a product.[4] Using such structural information for improving the code recognition is in principle possible, e.g., by reasoning that codes that start with certain country codes are more likely compared to others. The same holds for information about what kind of bar codes are on products sold in a certain store, which would allow for optimizations in combination with location-based services.[5] However, in practice this is usually not possible, since this context information is not available.

[4] An exception is the country code 974, which is used to identify books.

[5] A large supermarket like the Metro Future Store [12] features around 80000 different products, while smaller shops still have around 20000 different sales items.

3 Recognition Algorithm

In this chapter we present a novel algorithm for the recognition of bar codes on mobile phones that addresses the recognition challenges presented in Section 2.1. The main features of the presented algorithm are:

- Recognition of bar codes in images of varying sharpness, from sharp to very blurry images.
- Recognition of bar codes in images of varying sizes and formats. Typical image resolutions range from 320x480 pixels on the iPhone 3G and 3GS to 720x1280 on the iPhone 4. However, due to the capability of our method to recognize bar codes in very blurry images, even codes of only 50 pixels length are recognizable.
- A high recognition performance even in the presence of uneven lighting or difficult bar codes and a low rate of wrongly recognized codes (false-positives).
- Support for all major bar code symbologies used in the retail industry, i.e., EAN13/8, UPC-A/-E, EAN128 and Code39 codes.
- The recognition of bar codes that are arbitrarily oriented in an image.
- Fast recognition, independent of the underlying image resolution and a low memory footprint

The chapter is organized as follows. Section 3.1 will cover the general architecture of the recognition algorithm as well as relevant components not directly involved in decoding bar code numbers. This includes, for example, the detection of the presence and orientation of bar codes in images (Section 3.1.1) or the extraction of scan lines (Section 3.1.2). In order to maximize recognition accuracy, our algorithm combines two different methods to decode bar code numbers: One for codes in sharp images (Section 3.2) and a second one suitable for both sharp and blurry images (Section 3.3). Since bar codes have been in use for some time[6], previous work on optical bar code recognition and the recognition of bar codes on mobile phones exists. Section 3.4 will discuss this related work before Section 3.5 concludes with a brief summary.

[6] The retail industry introduced the European Article Number (EAN) format in 1977.

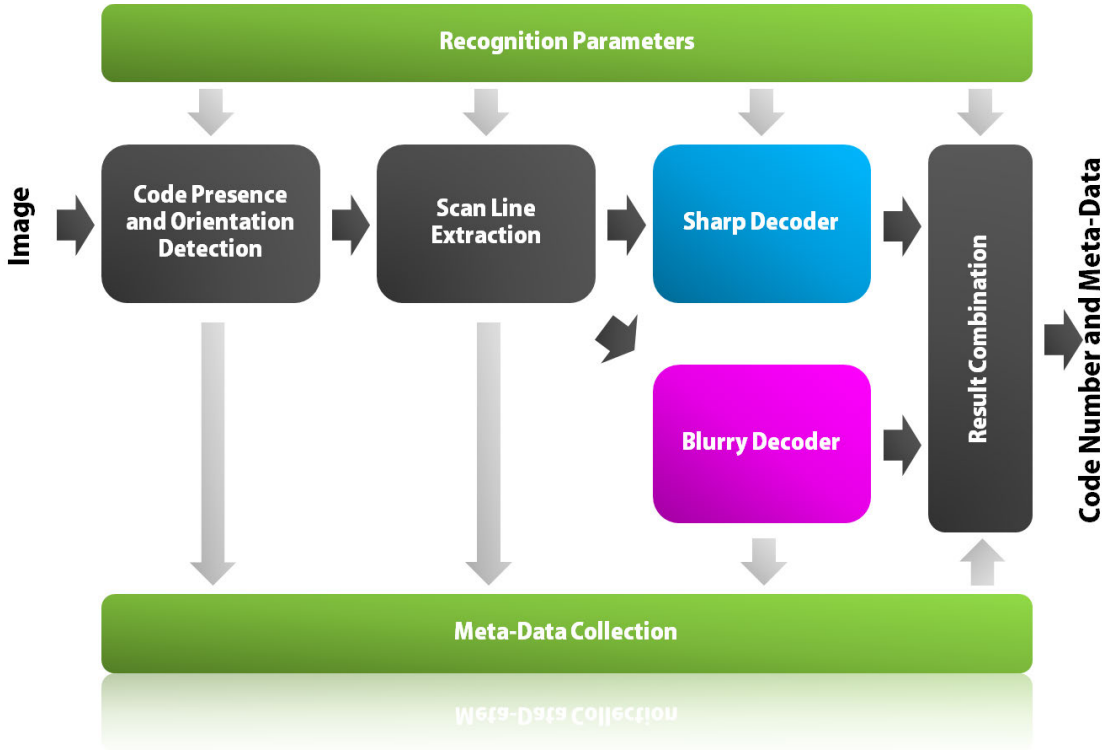


Figure 3.1 General architecture of the recognition process.

3.1 General Architecture

Figure 3.1 shows the basic components involved in obtaining the correct bar code number from a single image, or multiple frames in the case of the recognition in a video stream. First, we try to detect the presence and orientation of a bar code in a given image (Section 3.1.1). If no bar code is detected, the recognition attempt is aborted. If the likely presence of a bar code is detected, several scan lines are placed through the image at the previously determined angle of the bar code in the image, and image brightness information is extracted along these scan lines (Section 3.1.2). The resulting waveforms are then passed to a bar code decoder in order to extract the number encoded. In order to increase the overall recognition accuracy and speed, as well as address the manifold recognition challenges, we rely on two different bar code decoders:

1. The *sharp decoder* (Section 3.2) is fast and robust against distorted codes on round or crumpled surfaces, as well as perspective distortions. However, it is capable of recognizing bar codes in sharp or slightly blurry waveforms only.
2. The *blurry decoder* (Section 3.3) can recognize bar codes in sharp as well as blurry waveforms. The decoding process is in large parts invar-

iant against the waveform's resolution and sharpness, but requires more time than the sharp decoder.

Both decoders are used successively. First, the sharp decoder is used on all scan lines, since it allows for the decoding of multiple scan lines without noticeable performance drawbacks. In case the sharp decoder cannot detect a bar code, we also use the blurry decoder. Due to its higher performance requirements, the blurry decoder is used only for a single scan line. During the whole recognition process the output of each component and additional meta-data is collected. Collected information includes:

- Information if a bar code is present in the current image
- The orientation of the bar code
- The number and position of scan lines used
- The start- and end-position of bar codes detected in each scan line
- Information about how reliable decoded bar codes numbers are
- Additional information, including the image sharpness, lighting situation (glare or uneven lighting) and information at which state the recognition has been aborted in case a code could not be recognized

Results returned by the two decoders and collected meta-data from several video frames is then combined in the *result combination* component (Section 3.1.3) and used to assemble the final *recognition state information*. The recognition state information includes:

- Information if a bar code is present in the images and how close the mobile phone is to this bar code
- The last decoded bar code number, including information like its symbology and a confidence value describing how confident our algorithm is to have recognized the correct number
- The orientation and position of the bar code in the video images

The additional information, besides the bar code numbers themselves, can be used to enable applications like the ones shown in Figure 3.2. The "price-check" demonstrator augments the camera images with a price label showing the cheapest price for a specific product found online. The price label changes size, position and orientation with the bar code visible in the images. Further examples include orientation sensitive sliders or menus that allow users to select items by changing the orientation of the mobile phone in relation to the

bar code (see Figure 3.2). Such forms of interaction have the potential to ease and accelerate the interaction with products and are related to the Visual Code Widgets [21] presented by Rohs that are based on the 2D visual code.

A variety of parameters influence the recognition process, including the number of scan lines used, the orientation detection step, and the decoding of bar codes in the sharp and blurry decoder. These parameters have an influence on both the accuracy of recognition and the recognition speed. They can be used to adjust the recognition process to different mobile phones with varying capabilities and resources like CPU and RAM. Optimal parameter values for specific mobile phones are learned offline on sets of test images and stored in configuration files. Section 4.2.3 will provide more details on this optimization process. Configuration files also support simple scripting capabilities that allow the definition of more general configuration files that specify parameters to use depending on the image resolution or sharpness. Such general configuration files support multiple mobile phones without the need to create specific files for each phone type and model. The *recognition parameters* component loads the appropriate configuration file for a specific device or device class and adjusts the recognition process according to its content. Further details about the relevant parameters controlling the recognition will be provided in the specific sub sections.

Figure 3.2 Screenshots of an augmented-reality price-check application that overlays the real camera images with a virtual price tag that changes position, orientation and size according to the bar code visible in the images (left image from an iPhone 3GS). Prototypical application illustrating orientation sensitive menus (middle images from a Nokia N95 device) and sliders (right images from a Nokia N70 device).

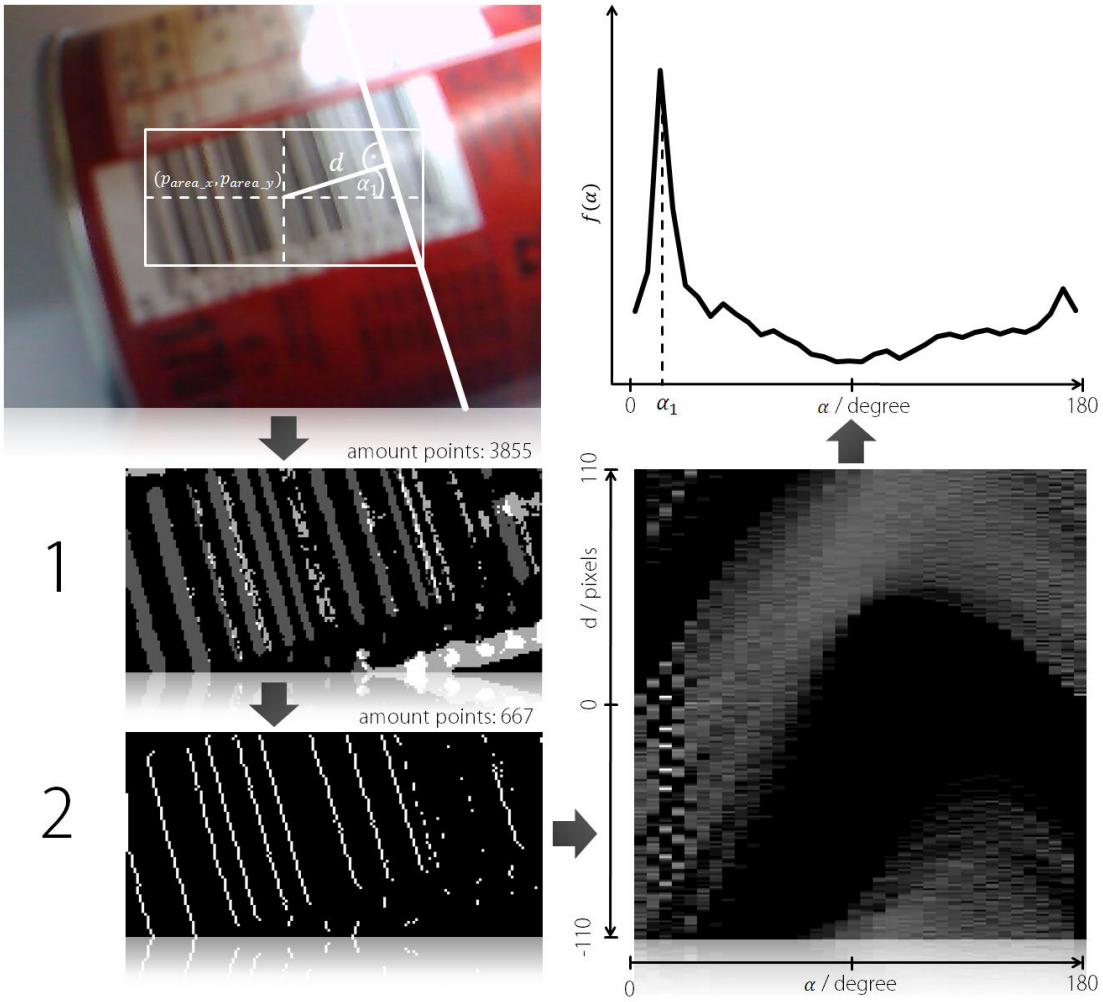


Figure 3.3 Steps performed during the orientation detection: 1. Extraction of sub-window from original image (upper-left image). 2. Detected edges colored according to their type (image 1) and the resulting edges when applying a further edge detection step to the most often occurring edge type in image 1 (image 2). 3. Result of the Hough transform (lower-right image). 4. Search for patterns that are characteristic for the bar code bars in the Hough space and determination of the bar's angle.

3.1.1 Code Presence and Orientation Detection

Detecting the orientation of bar codes in images such as the ones shown in Section 2.1 poses the following challenges:

- Since image aspect ratios can vary considerably, bar codes are not guaranteed to cover a reasonable part of the whole image. For example, this is the case on iPhone and on Android devices, on which the obtained images are in portrait mode (see Figure 4.19 on page 110).
- Since we do not know the image's sharpness beforehand, the proposed method has to work for all sharpness levels, ranging from sharp to very blurry images.

- Noise complicates the process and includes image errors, compression artifacts, lighting effects or local code defects. Such defects rule out approaches that try to track the bar code's bars based on the color of pixels, and requires for a more robust solution.
- The algorithm has to be fast, even in case of high resolution images.

Our method for orientation detection has already been published in [22] and can be broken down into three steps that are illustrated in Figure 3.3. First, we perform an edge detection on a pre-defined area of the original image. On the resulting image that contains the detected edges, we apply a speed-optimized Hough transform [23, 24], and finally search for patterns characteristic for parallel lines in the Hough space. The Hough transform is a standard method in image recognition that allows for the detection of analytically representable features in images, such as straight lines, circles or ellipses. The recognition of these global patterns in the image space is based on the identification of local patterns in a transformed parameter space. The following sections provide further details regarding these steps.

3.1.1.1 Edge Detection

In order to address the fact that multiple bar codes can be visible in an image, we focus on a certain area in the original image $I(x, y)$. This area is defined by its center point (p_{area_x}, p_{area_y}) and window size $(p_{area\_width}, p_{area\_height})$. In a first step, a very fast edge detection algorithm that is based on simple threshold values is applied on the image data. A pixel in the resulting edge image is set, if the difference of its brightness value to the brightness value of its preceding pixel in x- or y-direction is higher than a threshold value $p_{threshold}$. The brightness value of a pixel at position (x, y) is calculated based on the red, green and blue values of a pixel[7]:

$$brightness(x, y) = \frac{1}{4}\left[red\big(I(x, y)\big) + 2 \cdot green\big(I(x, y)\big) + blue\big(I(x, y)\big)\right]$$

[7] In many image sensors, more die space is dedicated to the detection of light waveforms corresponding to the color green, since humans can distinguish the brightness of green tones much better compared to red or blue ones. When calculating the brightness value, we consider the green value therefore above average, since a larger sensor area usually results in less noise, especially in low-light situations.

Instead of considering every pixel in the original window, we consider only every $p_{distance}th$ pixel in x- and y-direction, in order to limit the number of pixels that have to be accessed and therefore speed-up the process. The value for $p_{distance}$ depends on the image resolution and is chosen in such a way that the same number of pixels have to be accessed, independent of the underlying image resolution. Image 1 in Figure 3.3 shows the resulting edges. However, this edge image is not in all cases well suited for a direct application of the Hough transform, due to three issues:

1. The Hough transform is computationally expensive and the required time increases linearly with the number of detected edge pixels. This problem of many edge pixels increases with the requirement to detect lines both in sharp and very blurry images without knowing the image sharpness beforehand. If we adjust the threshold value $p_{threshold}$ to detect only sharp edges, the number of edge pixels decreases, but no edges are detected any more in very blurry images. If we lower the threshold value so that edges in very blurry images are detected, we end up with a large number of pixels in case of sharp images.[8]
2. Usually, not only edges resulting from the bar code's lines are detected, but also from other features, e.g., columns of text or the upper and lower vertical borders of the bar code pattern, which complicates the orientation detection. Since we cannot assume bar codes to be horizontal or nearly horizontal in an image, considering only edges in a certain direction is no solution to this problem.
3. Due to the low resolution of the edge window, detected edges in the case of codes that are neither vertical nor horizontal in the image tend to merge into each other, making it hard to detect parallel lines.

We address these challenges with a simple measure that reduces the number of edge pixels that have to be considered for the following Hough transform and the number of edges not belonging to bar code lines. When constructing the first edge image shown in Figure 3.3 (image 1), we distinguish for each pixel what kind of edge caused it: A vertical, horizontal or diagonal

[8] Detecting the image's sharpness before the orientation detection is not always reliable, since at this stage, we do not know if there is a bar code in the image or where and how large it is. A fast method observing arbitrary image parts may also be inaccurate due to the unknown background (e.g., a uniform area on the product packaging) or areas of differing sharpness in the image (e.g., a sharp code on a small product in the foreground and a blurry background).

edge in the original images. Each edge type corresponds to a different color. Furthermore, we count how often each edge type occurs. Based on this information, we keep only the edge type that occurs most often and discard the other two types. The underlying assumption here is that most edges will be caused by the bar code's bars. This step clears the edge image, especially in the case of diagonal codes. In order to further reduce the number of pixels, we apply an additional edge detection step, which produces the result shown in image 2 in Figure 3.3. While this second edge detection step shifts the position of edges, it preserves the edge orientation, which is relevant for our use case. Using such a simple edge detection approach produces not the best possible result, but it can be performed fast and the following Hough transform is robust enough to handle the remaining imperfections.

3.1.1.2 Hough Transform

In a second step, the Hough transform is used in order to detect straight lines in the edge image 2 shown in Figure 3.3. We use this method for the detection of the bar code's lines, since it is very robust regarding noise and partially hidden or incomplete lines, even if the image resolutions are low (in our case 160x120 pixels).

For the detection of lines, we use the line equation in its normal form, where d is the distance from the origin and α the angle with the normal: (The upper-left image in Figure 3.3 illustrates this association.)

$$d = x \cdot cos(\alpha) + y \cdot sin(\alpha)$$

Using this formula, the Hough transform will in principle map all collinear points lying on the line described by the above equation in the image space, to sinusoidal curves in the parameter space that intersect at the point (α, d). The algorithm used to calculate the parameter space is fairly simple: All the pixels in the original edge image are processed one by one. For each pixel at position (x, y), we are going to obtain a set of (α, d)-pairs by using different values for α in the above line equation. For each resulting point (α, d), the corresponding entry at that position in the parameter space is incremented by one unit. When all pixels are processed, the parameter space contains the number of collinear pixels for all the lines found in the image. The resulting parameter space for our example is displayed in the lower-right image in Figure 3.3.

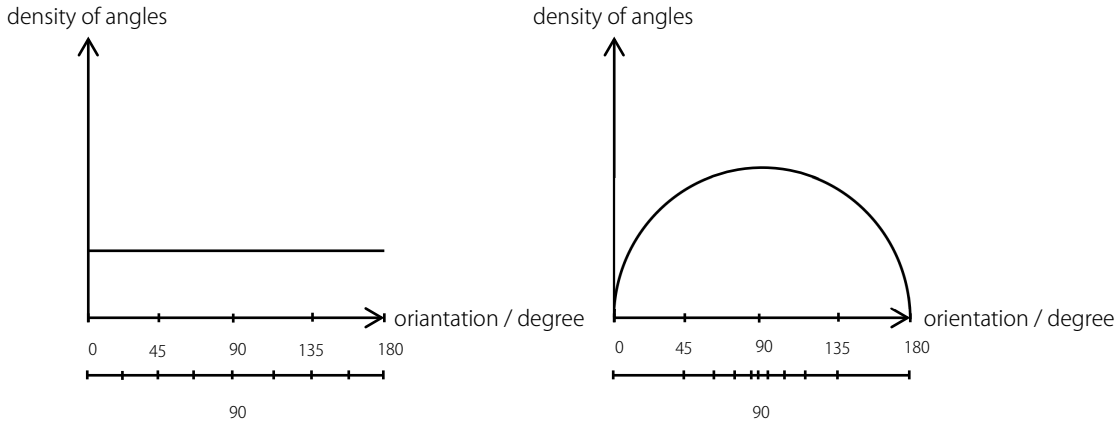


Figure 3.4 Adjustment of the angle distribution after an orientation has been found, in order to increase the algorithm's robustness and accuracy while maintaining overall speed.

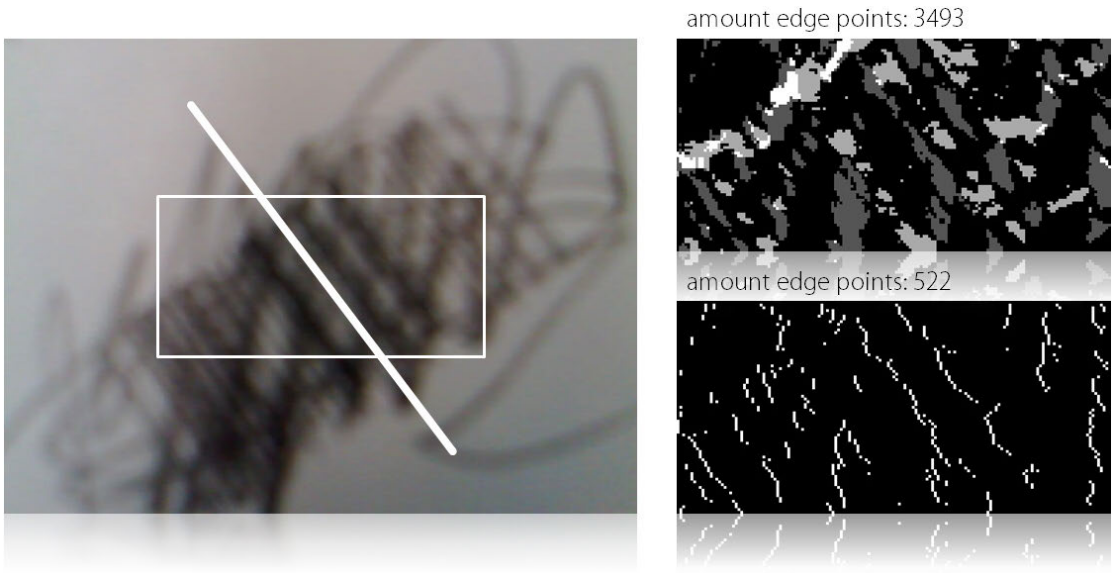


Figure 3.5 Example of the robustness of the described Hough-transform-based approach, which can determine the correct bar code orientation even in difficult situations like the one shown.

In order to consider all possible lines in the image space, the values for α have to range from 0 to 360 degrees. We can apply a simple trick and save half the necessary computations by allowing negative values for d. This way we have to evaluate only values from 0 to 180 degree. The number of angles evaluated (the entered values for α) for each pixel is variable and corresponds with the width of our parameter space. The more angles we use, the more accurate and robust the orientation detection becomes, but also the longer the computations take. Increasing the number of considered angles considerably increases the time needed to calculate the parameter space, since the calculations are executed for each pixel in the image space. We use 36 different angle values, evenly distributed from 0 to 180 at 5 degree steps (the left

diagram in Figure 3.4 shows this uniform angle distribution). In order to increase the accuracy as well as the robustness of the orientation detection, we change the distribution of angles if a bar code's orientation has already been recognized in previous images in such a way, that there is a higher angle density around the last detected orientation. For example, the right diagram in Figure 3.4 shows the new angle distribution if the last detected orientation angle was 90 degree. This way, we can achieve a higher accuracy and robustness while maintaining the fast execution speed of the algorithm, since the total number of considered angles (36) remains the same.

In order to accelerate the calculation of the parameter space further and allow for an execution of the algorithm in real-time, several other means have been taken, including the following major one: We use an optimized integer-based version of the Hough transform that abstains from using time consuming floating point operations and heavily relies on lookup tables. Such an integer-based implementation accelerates the algorithm considerably and Magli et al. [25] showed that the negative effects on the accuracy, compared to the floating point based version, are negligible.

3.1.1.3 Pattern Search

After the parameter space has been calculated, it is searched for a set of parallel lines belonging to the bar code. In order to accelerate this process, we can use the fact that all maxima belonging to parallel lines in the image space will show up in the same column in the parameter space, since all have the same angle parameter α. For each column of the parameter space, a value $f(\alpha)$ is calculated by adding up the differences of adjacent entries in that column. Figure 3.3 shows the resulting shape of $f(\alpha)$ for the different columns in our example. Determining the bar code's orientation is now reduced to the determination of the column a_1 that contains the highest value $f(a_1)$ – in our example 11.25 degree. However, a combination of low resolution edge windows and sharp images result often in a second, lower peak at an angle α_2 at a distance of 90 degree to the first one. This corresponds to the fact that many of the pixels in the edge window are lying not only on the lines of the bar code, but also on imaginary lines that are at a 90 degree angle to them along the bar code. We consider the determined orientation angle $\alpha = a_1$ therefore only then to be reliable if the following two conditions hold:

$$f(\alpha) > f(\alpha_2) \cdot 1.5$$

$$f(\alpha) > \overline{f} \cdot 2$$

$$\overline{f} \stackrel{\text{def}}{=} \frac{1}{length(f)} \sum_{x=0}^{length(f)-1} f(x)$$

$$length(f) \stackrel{\text{def}}{=} \textit{the length of the waveform } f \textit{ in pixels}$$

Currently, we use the information in the Hough space only to detect the orientation of bar codes, since this can be determined very robust even under difficult conditions (see Figure 3.5 for an example). However, the information contained in the Hough transform could also be used to detect, for example, the presence of 2D codes in the image. Instead of one prominent peak in $f(\alpha)$, we would expect in this case two equally prominent peaks at a distance of 90 degrees for all 2D code symbologies featuring block-typed patterns. The two peaks correspond to the prominent lines at the pattern edges that have a relative angle of 90 degree between each other. This information can be used to integrate our recognition of bar codes with software recognizing 2D codes and avoid the overhead of a naive solution, which consists of executing both recognition algorithms in parallel. Results of the code presence and orientation detection component include information if a bar code is likely present in the current image and the orientation α of the bar code in the image.

3.1.2 Scan Line Extraction and Adaptation

Based on information about the bar code's orientation α and the point (p_{area_x}, p_{area_y}) where the recognition should be performed, we place a set of $p_{num.of\ scanlines}$ scan lines through the image using the well-known Bresenham algorithm [26]. Each scan line s is characterized by its distance $d(s)$ to the central scan line going through the point (p_{area_x}, p_{area_y}) and its thickness in pixels $t(s)$.

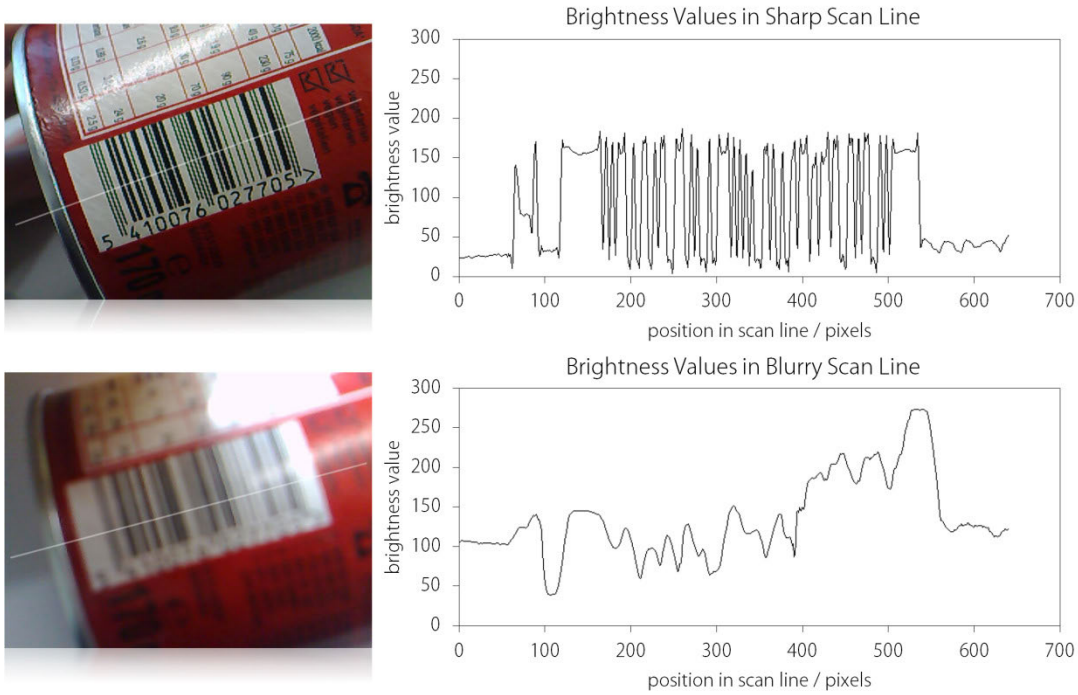


Figure 3.6 Position of central scan lines (left images) as well as extracted waveforms along these scans line in the case of a sharp image (upper-right diagram) and a blurry image as obtained on devices without autofocus camera (lower-right diagram).

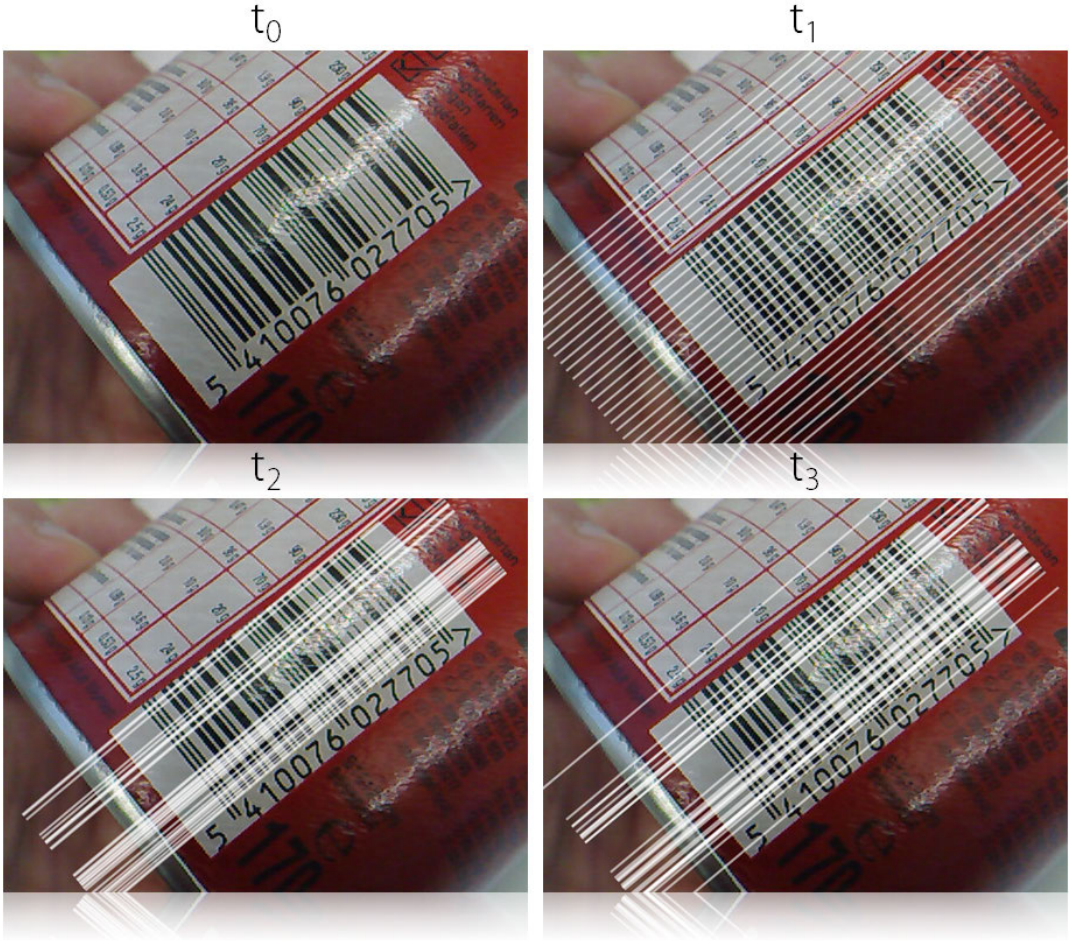


Figure 3.7 Dynamic adaptation of scan line positions over several frames.

3.1.2.1 Scan Line Extraction

Along each scan line, we extract pixel brightness information $f(x)$. Pixel brightness is calculated as already described in Section 3.1.1. In the case a scan line's thickness t exceeds one pixel, we combine the information obtained by the several parallel sub-lines $f_{nr}(x)$:

$$f(x) = \frac{1}{t}\sum_{nr=0}^{t-1} f_{nr}(x)$$

Thin scan lines have the advantage that less pixels of the original image have to be accessed, while thicker scan lines have the advantage that the effects of noise and local pixel errors are reduced. Figure 3.6 shows examples of typical extracted waveforms from a sharp as well as blurry image.

3.1.2.2 Scan Line Adaptation

When recognizing bar codes in video images with typically 10-15 frames/s, the meta-data collected in the previous video frames is used to adjust the position and thickness of scan lines in the current image, in order to maximize the likelihood of recognition. The adjustment is based on the "recognition success" of each scan line in the last frame. According to its recognition success, a scan line is categorized into one of the following five classes:

1. A valid bar code has been recognized along the scan line with a high confidence.
2. A bar code has been recognized along the scan line, but the result is not reliable.
3. The start- and end-position of a bar code have been recognized along the scan line but no code has been recognized.
4. It is likely that a bar code is present in the scan line waveform, e.g., due to the number of extreme points detected in the waveform $f(x)$.
5. The scan line contains most likely no bar code.

Based on this classification, the position and thickness of scan lines is adjusted according to the following rules:

- Scan lines of category 1 remain at their current position and with their previous thickness.
- The thickness of scan lines of category 2 is increased up to a certain threshold, in order to become more robust against local defects.

- If scan lines of category 1 or 2 are present, scan lines of category 3, 4 and 5 are moved towards the closest such scan line; else, scan lines of category 4 and 5 are moved towards potentially present scan lines of category 3.
- With a probability ρ (typically 0.1), scan lines of category 3, 4 and 5 are reset to their original position and width, in order to ensure an adjustment to changing conditions over time, and to avoid being caught in local maxima.

Figure 3.7 shows an example of the scan line adjustment process over a series of three frames $t_1 \ldots t_3$. The original image is shown at t_0.

3.1.3 Result Combination

In the case of video images, the recognition algorithm combines the results obtained from several images in a *recognition state*. The recognition state consists of the following:

- Information if a bar code is likely present in the video images and if the mobile phone is currently close to a bar code
- The last decoded bar code, including information about its symbology and a confidence value describing how sure our algorithm is to have recognized the correct number
- The bar code's orientation and position in the image(s)

As already mentioned before, this type of information is relevant for augmented-reality type of applications like the allergy assistant or price label application that display information as long as the user is close to a specific product.

3.1.3.1 Information about Bar Code Presence

A bar code is considered to be present in the images if the orientation detection component detected a prominent set of parallel lines in at least 50% of the last $p_{num\_of\_frames}$ image frames. We do not require all frames to result in a valid orientation, since certain frames can be severely distorted and blurred due to hand movements. The number of considered frames depends on the frame rate available on a specific mobile phone and is typically 5 frames. The bar code presence indicator reacts not only to bar codes, but all prominent sets of parallel lines, e.g., also patterns caused by text columns. We therefore

also provide information if the phone is currently in the proximity of a "real" code. If a bar code number has been recognized in a frame, we assume to be close to this bar code until the presence of parallel lines in the video images cannot be detected for $p_{num\_of\_frames}$ frames.

3.1.3.2 Information about Decoded Bar Code

In the case any of the two decoders detected a bar code with a confidence value above a threshold value $p_{final\_confidence\_threshold}$, this code is considered. All bar codes detected in the last $p_{num\_of\_frames}$ video frames are recorded. In principle, this information can be used to increase the confidence value in a bar code that has been recognized in several successive video frames. Nevertheless, since the decoders internally perform already several checks in order to avoid the detection of wrong code numbers, we do not rely on the aggregation of results from different frames in order to determine the final bar code. Especially in the case of very blurry codes, the chance of detecting a wrong code number is too high when automatically increasing the confidence value of a bar code with a low confidence value over several video frames.

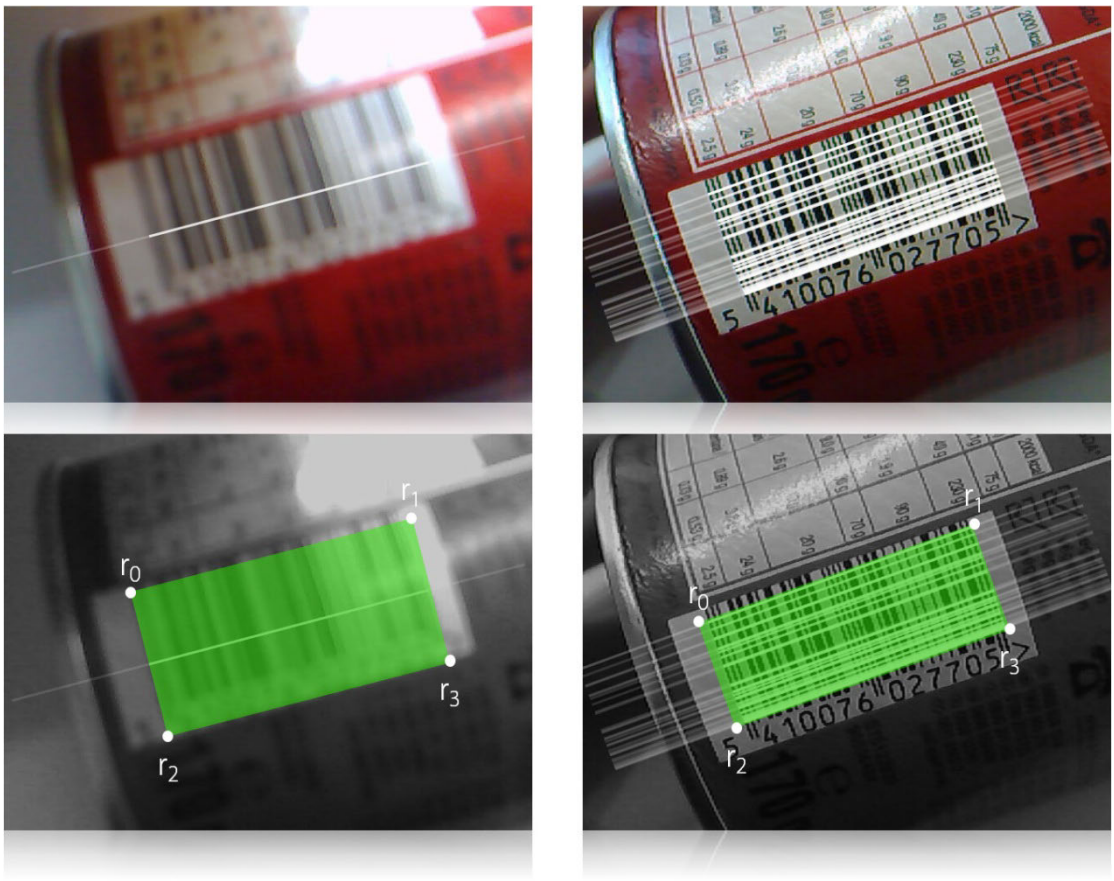


Figure 3.8 Determination of the approximate code position based on the information returned by the blurry decoder (left images) as well as sharp decoder (right images).

3.1.3.3 Information about Bar Code Position

Information about the bar code position in an image includes the code orientation α in degrees and a rectangular area, in which the code is located. The latter is defined by four corner points $r_0 \dots r_3$. Since the bar code presence and orientation detection can be incorrect for some frames, e.g., due to hand movements, a code's orientation value returned as part of the recognition state is also filtered over the last frames:

$$\alpha = \frac{1}{v} \sum_{t=0}^{p_{num\_of\_frames}} \alpha(t)$$

$$v = number\ of\ valid\ orientations\ in\ the\ last\ p_{amount\_frames}\ frames$$

$$\alpha(t) = detected\ orientation\ at\ the\ t_{th}\ lastframe\ and$$
$$\alpha(t) = 0\ if\ no\ valid\ orientation\ could\ be\ detected$$

The approximate code position is calculated based on the information recorded, e.g., the positions of scan lines and the start- and end-position of potentially detected bar codes in each scan line. If the sharp decoder recognized a bar code, we use the outermost two scan lines that recognized a bar code to calculate $r_0 \dots r_3$ in image coordinates. In the case the blurry decoder recognized a code with a high enough confidence, we construct the points $r_0 \dots r_3$ by assuming a rectangle with a pre-defined width-to-height ratio and the blurry scan line that detected the bar code at its center. Furthermore, the approximate code position reported as part of the recognition state is also filtered over several video frames, in order to avoid flickering. Figure 3.8 illustrates the detected approximate code position in the case of a sharp and blurry image.

While this method always results in a rectangular area that perfectly corresponds with the code's orientation and start- as well as end-position, the height of the rectangular area and its y-position in relation to the code can vary – in particular, in the case of blurry images when the approximate code position is based on one scan line. For perfect images, a simple and fast algorithm detecting the upper and lower vertical borders of the bar code could be implemented. However, in the case of blurry images, and the many imperfections found in reality, developing a fast and robust method requires more effort.

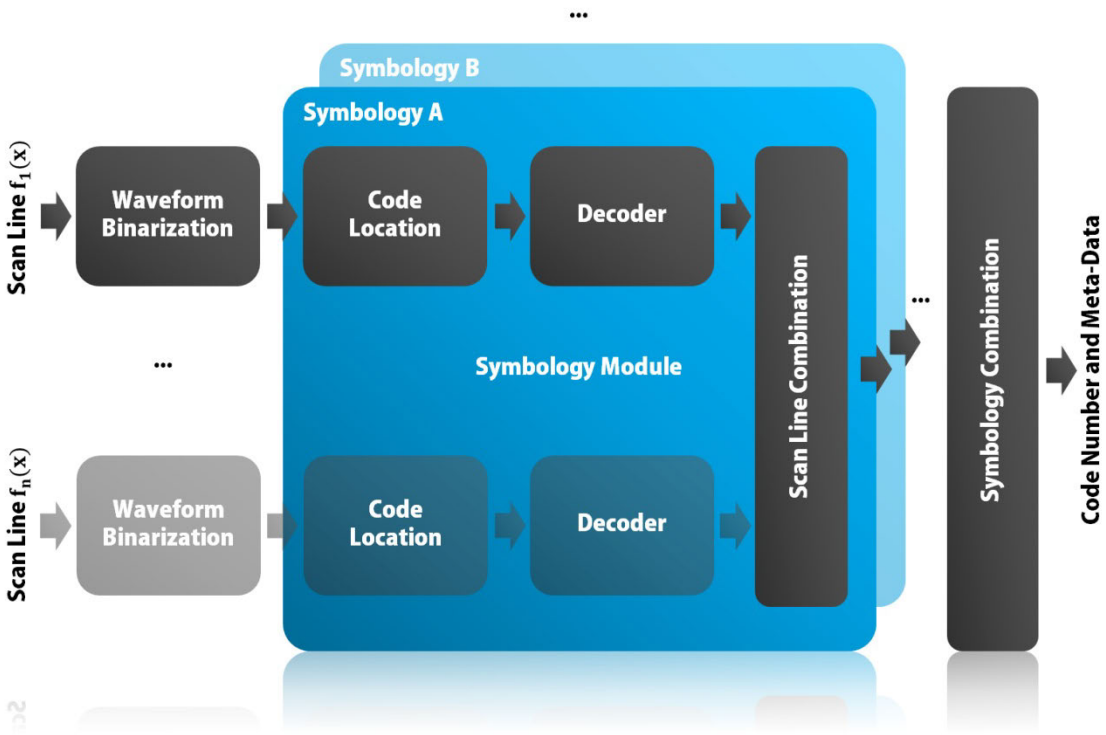


Figure 3.9 Structure of the sharp bar code decoder.

3.2 Sharp Decoder

The decoder for sharp bar code images had four specific design goals:

1. It should be as fast as possible in order to result in a low performance overhead when run in parallel with the blurry decoder
2. It should be robust against perspective distortions
3. It should be robust against local defects like glare or damaged codes
4. It should allow for the easy addition of new code symbologies

In order to achieve these goals, the sharp decoder uses fast, distortion invariant operations for decoding bar codes for each scan line and relies heavily on the use of multiple scan lines in order to increase robustness. Due to the scan line-based approach, we do not have to perform time intensive operations like the binarization process on the whole image but only along certain scan lines. This drastically increases the recognition speed and reduces the overall memory consumption in the case of high resolution images. The functionality required to decode a bar code of a certain symbology is encapsulated in a *symbology module* (see Figure 3.9). Each symbology module has to implement three components, corresponding to the three main tasks it has to fulfill:

1. The *Location* component, which provides the functionality to detect the presence, as well as start- and end-position of a bar code in a given set of alternating black and white fields
2. The *Decoder* component, which provides the functionality to decode a bar code from a given set of alternating black and white fields that belong to the code pattern
3. The *Scan Line Combination* component, which provides the functionality to merge decoder results obtained from different scan lines and uses the combined information to construct a final bar code

Modules for bar code symbologies can be dynamically added, activated or de-activated. This can be done, for example, based on the requirements of the application using the bar code recognition. Figure 3.9 shows the structure of the sharp bar code decoder that takes a set of scan line waveforms as input and produces a final bar code as well as meta-data about the recognition process as output. We published the general recognition method of the sharp decoder described in the following already in [22], and a related, but very limited first algorithm for recognizing sharp bar codes in [14].

First, the original waveforms that are extracted from the scan lines and contain brightness values ranging from 0...255 are binarized (converted to only two values). On the resulting set of alternative black and white fields, each active symbology module performs a fast check in order to clarify if a valid code is contained in the pattern. In the case a symbology passes this test, the code's start position in the pattern is searched for. If a valid start position was found, this information is used to decode the bar code in the pattern. This process is repeated for each scan line. The decoding result of each scan line is then passed on to the *result combination* component. This component combines the information obtained from different scan lines and returns the most likely bar code and a quality factor indicating the confidence in the result. This information is reported by each active symbology module and passed on to the symbology combination component. Based on the obtained results and additional meta-information, e.g., how many scan lines recognized what kind of code symbology, this component selects and returns the final bar code. In most cases, only one code symbology will be detected in an image. However, in images with multiple bar codes present, several symbology modules will detect a bar code. The following sub-sections discuss relevant components in more detail: the waveform binarization (Section 3.2.1), and symbology-specific code location detection, decoding, and result combination steps (Section 3.2.2).

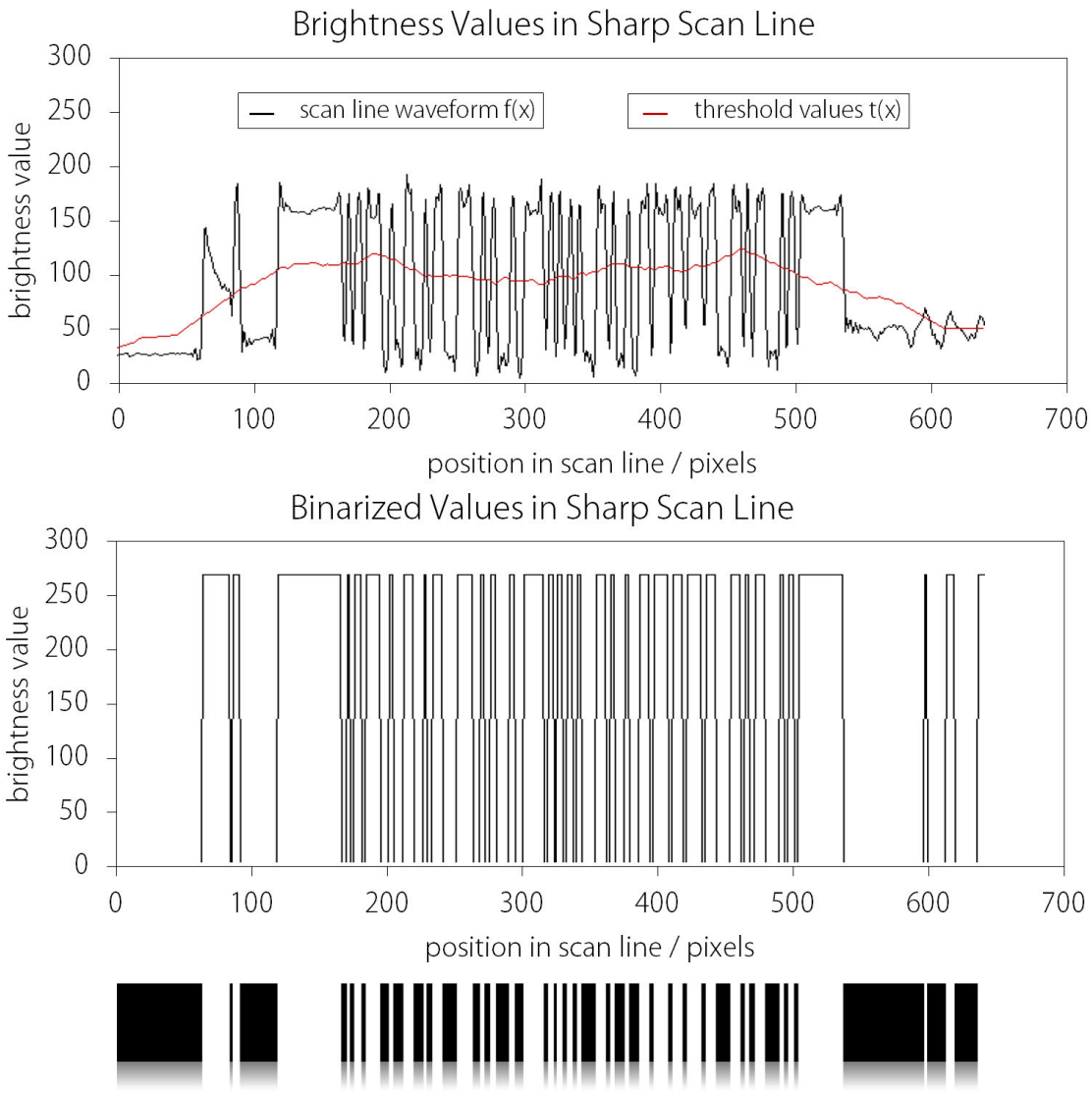


Figure 3.10 Sharp bar code waveform from a Nokia N95 phone that exhibits sharpening artifacts at the signal's edges and the threshold values, determined with a window size of 70 pixels, used in binarization algorithm 1 to classify pixels as black or white (upper diagram). Binarized waveform (middle diagram) that corresponds to the final set of black and white fields (lower image).

3.2.1 Waveform Binarization

First, the waveform data along a scan line is transformed into a set of alternating black and white fields, using one of two different binarization algorithms. Which algorithm is the most appropriate, depends on the image resolution and potentially present artifacts from image compression or sharpening. The outcome is a set of alternating black and white fields of varying lengths. The drawback of performing a binarization at such an early stage is the accumulation of errors, as subsequent processing steps might be based on faulty binarization information (i.e., a grey pixel being wrongly judged as being black or

white), especially in the case of slightly blurry or low-resolution images. However, it has the advantage that the following operation steps can be performed very fast. By considering only the relative sizes of consecutive black and white fields, a high robustness against distortions, e.g., perspective distortions or distortions due to codes printed on round surfaces, is achieved.

3.2.1.1 Algorithm 1

The first binarization algorithm is based on a simple dynamic threshold approach, similar to work presented in [27]. It is very fast and robust against waveform artifacts resulting from image compression or edge sharpening (see Figure 3.10 for an example). The threshold value $t(x_1)$ used to decide if a pixel at position x_1 is converted to black or white is based on the average brightness along the whole scan line $f(x)$, as well as the local illumination $local(x_1)$ around the pixel at position x_1:

$$t(x) = p_{ratio} \cdot \bar{f} + (1 - p_{ratio}) * local(x) \quad \forall x \in \{0 \dots length(f)\}, 0 \leq p_{ratio} \leq 1$$

$$local(x) = \frac{1}{p_{window\ size}} \sum_{i=x-p_{window\ size}/2}^{x+\frac{p_{window\ size}}{2}-1} f(i)$$

A pixel at position x_1 is converted to "black" if the following condition holds, else it will be interpreted as "white":

$$f(x_1) < t(x_1) + p_{bin\ threshold} \quad \forall x \in \{0 \dots length(f)\}, p_{bin\ threshold} \in \mathbb{R}$$

The parameters p_{ratio}, $p_{bin\ threshold}$, and the window size $p_{window\ size}$ are set dynamically based on the illumination situation found in the original waveform. By analyzing the brightness values of pixels along the scan line and their distribution, the illumination of a scan line waveform is categorized into six categories, along the two dimensions $image\ brightness \in \{dark, normal, bright\}$ and $uniformity\ of\ lighting \in \{normal, uneven\}$.

Based on the estimated illumination (e.g., $normal$ and $uneven$), we select the according set of values for p_{ratio}, $p_{bin\ threshold}$ and $p_{window\ size}$. The best parameter values for a certain illumination category have been determined automatically based on sets of test images for each category (see Section 4.2.3 for further information).

3.2.1.2 Algorithm 2

While being very robust, the performance of algorithm 1 decreases in the case of slightly blurry images and low image resolutions. Such conditions typically lead to smoother waveforms with only slightly visible extreme points. In such situations, and on devices featuring very good image quality, a more sensitive binarization algorithm is used.

The second binarization algorithm first smoothes the original waveform $f(x)$ by convoluting it with a small kernel $k(x)$, which reduces potentially present noise without affecting the more prominent extreme points in the signal:

$$f_{smoothed}(x) = (f * k)[x] \stackrel{\text{def}}{=} \sum_{i=0}^{length(k)-1} f(x - \frac{length(k)}{2} + i)k(i)$$

$$it\ holds: \sum_{x=0}^{length(k)-1} k(x) = 1$$

Afterwards, the first and second derivative of the resulting waveform $f_{smoothed}(x)$ is used to determine the type, position and size of extreme points in the waveform. Too small extreme points are ignored and based on the information about the resulting extreme points the pattern of black and white fields is derived.

Especially in the case of slightly blurry, noisy or low resolution images, it often depends on details like single pixel errors or local artifacts whether the complete waveform of a bar code can be binarized correctly. Like already mentioned in [14], the overall recognition performance of algorithm 1 and algorithm 2 can therefore be improved by slightly varying the relevant parameters influencing the binarization output; either in the spatial domain between different scan lines or time domain in consecutive images in a video stream.

3.2.2 Symbology Module: Code Detection and Decoding

3.2.2.1 Code Presence and Location Detection

The input of the code location component consists of the type $fields_{type}(i) \in \{black, white\}$ as well as width $fields_{width}(i) \in \mathbb{N}$ in pixels of the consecutive black and white fields. This information is passed on to all active symbology modules. Each module tries to detect if its symbology is contained in the given pattern. In many cases, the recognition can be aborted at this stage, e.g., if

there are not at least $p_{min.\ num.\ of\ fields}$ fields present in the pattern (59 fields in the case of EAN13 or UPC12 codes)[9]. This allows us to cancel the recognition as early as possible for scan lines that cannot contain a valid bar code. If a symbology passes this test, the start position of a corresponding code is searched in the field pattern. Requirements in doing this are again speed and the exclusion of patterns not belonging to bar codes, but are caused by other elements such as text lines printed on a product's package.

A valid start position is searched for by consecutively testing all possible start fields in the given pattern. A start index i_{start} is valid if it holds:

$$fields_{type}(i_{start}) = black$$

$$i_{start} < length(fields) - p_{min.\ num.of\ fields}$$

For each valid start index, a series of checks is performed, gradually moving from fast to perform checks to more time intensive ones. If one of the checks is not passed, we move on to the next start index. If all checks are passed, the according start position is reported to the *decoder* module. Performed checks depend on the structure and characteristics of the underlying code symbology, such as whether the length of encoded information is variable (EAN128 or Code39 codes) or fixed (EAN13, UPC-A, EAN8 and UPC-E codes). In the case of EAN13 bar codes, performed checks include the following:

- The three fields belonging to the start pattern must have approximately the same length
- The fields at positions marking the middle and end delimiter must have approximately the same size
- There must be a small silent area before and behind the code
- All field widths within the code pattern must have reasonable sizes[10]

3.2.2.2 Decoding

The decoding process depends on the specific bar code symbology. In the case of EAN13 codes, this process is straightforward: First, the lengths of fields belonging to the first digit $fields_{width}(i_{start})...fields_{width}(i_{start} + 3)$ are

[9] 3 (start delimiter fields) + 6*4 (fields from six left digits) + 5 (middle delimiter fields) + 6 *4 (fields from six right digits) + 3 (end delimiter fields) = 59 fields

[10] The max width of a bar is 4 times the unit width, but due to perspective distortion, fields might be slightly larger or smaller.

compared to all entries belonging to left-hand digits listed in Table 2.2. We determine the entry that corresponds best to the observed field widths. This way, we obtain information about the first digit d and its parity value p. Based on the latter, it can be distinguish if we are about to read the bar code from the "correct" direction or if it is upside-down, since the digit closest to the bar code's start delimiter always has an odd parity, while the one closest to the code's end delimiter has an even parity.

For the following steps, we assume the bar code to be in the correct position. For fields belonging to the six left bar code digits, the above comparison process with Table 2.2 is repeated to determine the encoded digits $d_1 \ldots d_6$ and the according parity values $p_1 \ldots p_6$. For digits on the right code side, comparisons are limited to the table for the right hand digits and digits $d_7 \ldots d_{12}$ are determined accordingly. Furthermore, in each comparison step confidence values $c_1 \ldots c_{12}$ are recorded that indicate how well the best found digits matched. Finally, the system digit $d_0$is derived with the help of Table 2.3 and the parity values $p_1 \ldots p_6$.

In order to reduce the number of false-positives, the decoding process is aborted if the parity values $p_1 \ldots p_6$ do not match any pattern in Table 2.3, or in the case a single confidence factor is below a set limit. In addition, we also calculate the bar code's check digit d_{check} from the values $d_0 \ldots d_{11}$ and check if it equals d_{12}:

$$d_{check} = \left(10 - \left(\sum_{x=0}^{5} d_{x \cdot 2} + d_{x \cdot 2+1}\right) mod\ 10\right) mod\ 10$$

Finally, the detected digits $d_0 \ldots d_{12}$, confidence factors $c_0 \ldots c_{12}$, as well as information about the check digit, are passed on to the result combination component.

For EAN128 codes, the decoding process is slightly more complex, due to the higher number of different encoded characters (105), the fact that characters are encoded in 6 consecutive bars, and the existence of different character sets. Special control characters allow for the switching between these character sets inside the same code.

3.2.2.3 Result Combination

The result combination component collects decoded digits and confidence values from each scan line, and combines this information to the final bar code number and a single quality factor. This quality factor ranges from 0 to 100 and indicates how sure we are about the correctness of the returned bar

code number (0 = very insecure, 100 = very sure) [11]. The final code number is then determined based on a set of rules. These rules have been experimentally determined using test sets of bar code images and optimized to reduce the number of false-positives as much as possible, while remaining sensitive enough to recognize even difficult bar codes. Rules vary with symbology, but include the following:

1. If not at least one scan line detected a code with a valid check digit and a reasonable confidence value, no final code is returned, since the probability of false-positives is too high
2. The more scan lines recognized the same code number, the higher the confidence in this code
3. The better the confidence values for single detected digits are, the higher the confidence in the final code

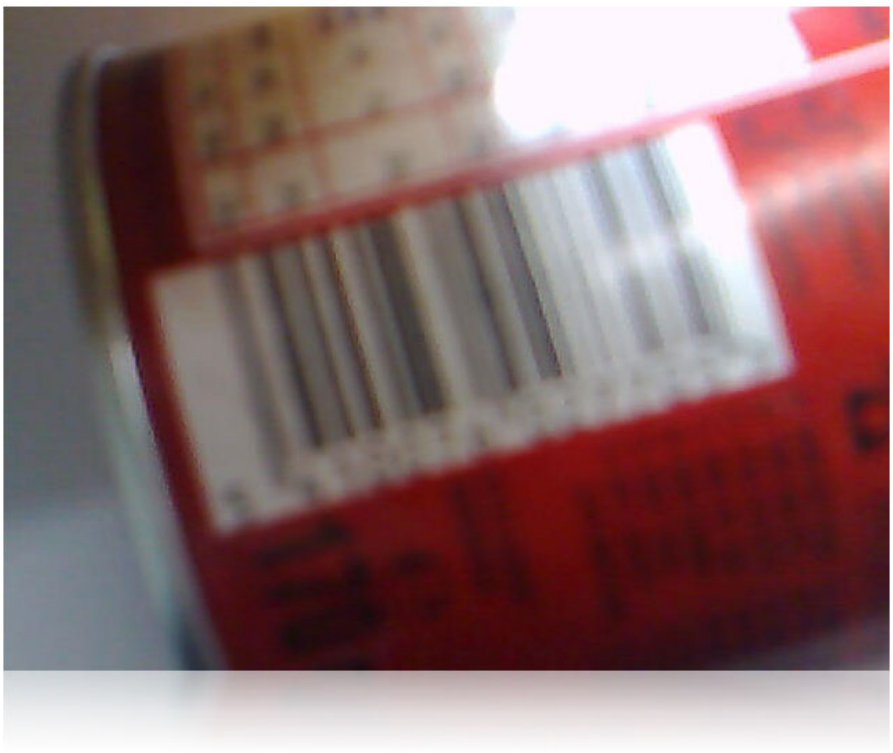


Figure 3.11 Close-up of blurry bar code image obtained on a device with fixed-focus camera that will be used as an example throughout the blurry decoder section.

[11] Throughout the sharp decoder, no floating point numbers are used in order to increase portability and the speed on weaker devices. For example, J2ME offers no support for floating point numbers in CLDC 1.0.

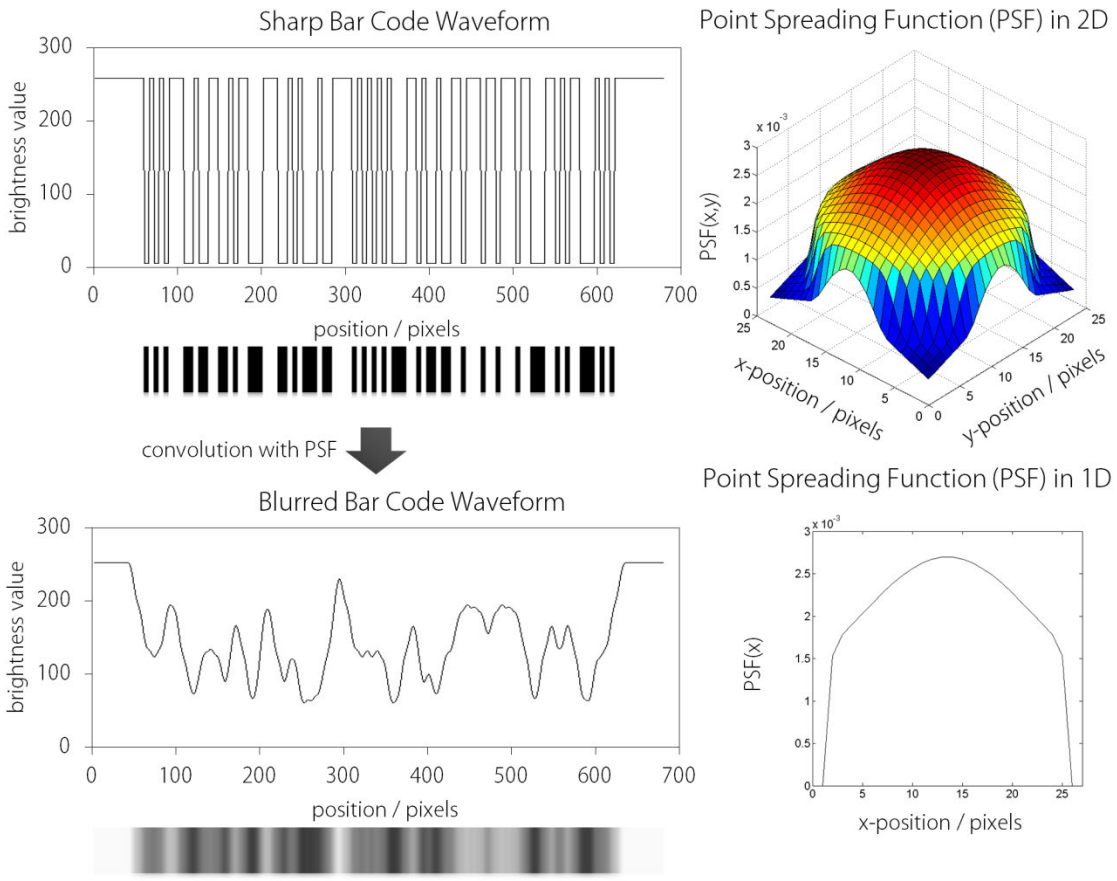


Figure 3.12 Underlying model of the blurry bar code recognition.

3.3 Blurry Decoder

Figure 3.11 shows a close-up of the blurry example image presented already in Figure 3.6. This image will be used as an example throughout the description of the blurry bar code decoder. The basic problem when trying the recognize bar codes in such blurry images is the fact that the individual bars making up the bar code cannot be distinguished any more, leave alone the precise widths of single bars. This rules out the recognition method used by the sharp bar code decoder and requires a different approach.

We assume an underlying model in which the blurry waveform of brightness values is obtained by convoluting the waveform of a sharp bar code with the point spread function (PSF) of the mobile phone's camera (see Figure 3.12). The point spread function describes the properties of an optical system and in particular how a single pixel in the original image will be reproduced in the resulting image. This model is the same as the one used in several related approaches for recognizing bar codes in blurry waveforms presented in Section 3.4. These methods try to mathematically determine the bar code number that corresponds to a sharp waveform that produces the observed

blurry waveform when convoluted with a fixed PSF. This combinatorial method has two major drawbacks, however, for recognizing bar codes on mobile phones:

- It is computationally very expensive and requires several seconds or minutes, even on high-end desktop machines for a single image.
- Reality is complex and differs from the described simple model in several ways. This either drastically increases the time required for the computations further, e.g., when considering additional degrees of freedom, or limits the approach to very specific situations.

Due to the many imperfections present on mobile phones and factors influencing the final blurry bar code waveform, the described model differs from reality. Specific sources for discrepancies include:

- The point spread function of the optical system is not constant on the whole image and can differ significantly dependent on the position inside the image, for example, due to barrel distortions caused by lenses that become relevant at a close distance to bar codes.
- Perspective distortions are likely to occur due to codes printed on round surfaces or an angular perspective on codes.
- Noise due to artifacts from image compression or sharpening steps further distorts the resulting blurry waveform.
- Lighting along the waveform is often non-uniform.
- The physical size of the code has an effect on the resulting waveform. The same code number encoded in bar codes of differing physical size obtained from the same mobile phone result in different waveforms.
- Further subtle facts like the printing quality of the bar code (thin black bars or thicker bars) have an effect on the result.

3.3.1 Our Approach

Our approach for determining the bar code number encoded in a blurry waveform differs from the combinatorial approach taken in related work in order to address the two main issues: algorithm speed and the many imperfections found in reality. Based on the determined exact start and end position of a bar code pattern in the scan line waveform, we subdivide the bar code waveform into several sections, each containing the waveform resulting from three consecutive digits and adjacent static elements.

Figure 3.13 Partitioning of EAN13 bar codes into four smaller segments, each covering three digits.

Figure 3.13 illustrates this partitioning. For each section, we pre-calculate the blurry waveform pieces that correspond to all possible sets of three digits and a known PSF. This is done offline and the pre-calculated patterns are stored in a *recognition table* file. In order to determine the underlying bar code number from a blurry waveform, we compare the waveform piece for each section with all pre-calculated patterns for this section and search for the best fitting pattern and its associated set of three digits. This comparison is optimized and can be performed very quickly. The sub-division of the whole bar code pattern into blocks of three digits (in the case of EAN13 codes) proved to be a good compromise between recognition speed and robustness.

Subdividing the whole waveform in sections covering only one digit would result in a high recognition speed, since each section waveform has to be compared to at most 20 different patterns (a digit can have values between 0 and 9, and one out of two parities). However, in the case of very blurry images, the shape of the waveform belonging to a specific digit is heavily influenced by the preceding and following digit, making a correct recognition impossible. On the other hand, pre-calculating the waveform of the whole bar code, or even half of the bar code, would result in too many patterns that cannot be stored and searched in an appropriate time. In the case of the six left digits, we would end up with $10*20^5 = 32*10^6$ different patterns (the first digit has always the same parity). Our subdivision into blocks of three digits

proved sufficiently robust against the influence of proximate digits and results in a total of 13000 pre-computed comparison patterns:

- $10*20^2 = 4000$ patterns for the first section
- $20^3 = 8000$ patterns for the second section (since not all combinations of digit parities are valid, see Table 2.3, it is sufficient to consider 7000 patterns)
- $10^3 = 1000$ patterns for the third section
- $10^3 = 1000$ patterns for the fourth section

In order to improve robustness and allow for the recognition of images with differing sharpness levels, we pre-calculate not only one recognition table, but several tables using different PSFs. We also allow the PSFs for different waveform sections to differ from each other.

As mentioned above, the observed blurry bar code waveform is influenced by many factors, e.g., image pre-processing, lens distortions and others. Furthermore, the exact measurement of PSFs on different image regions for a given mobile phone is non-trivial, at least without professional equipment. In order to address these issues, we do not try to model all imperfections, but instead optimize the number of recognition tables, the size and shape of PSFs for different code sections, as well as several other relevant factors based on a set of test images taken with a specific mobile phone.

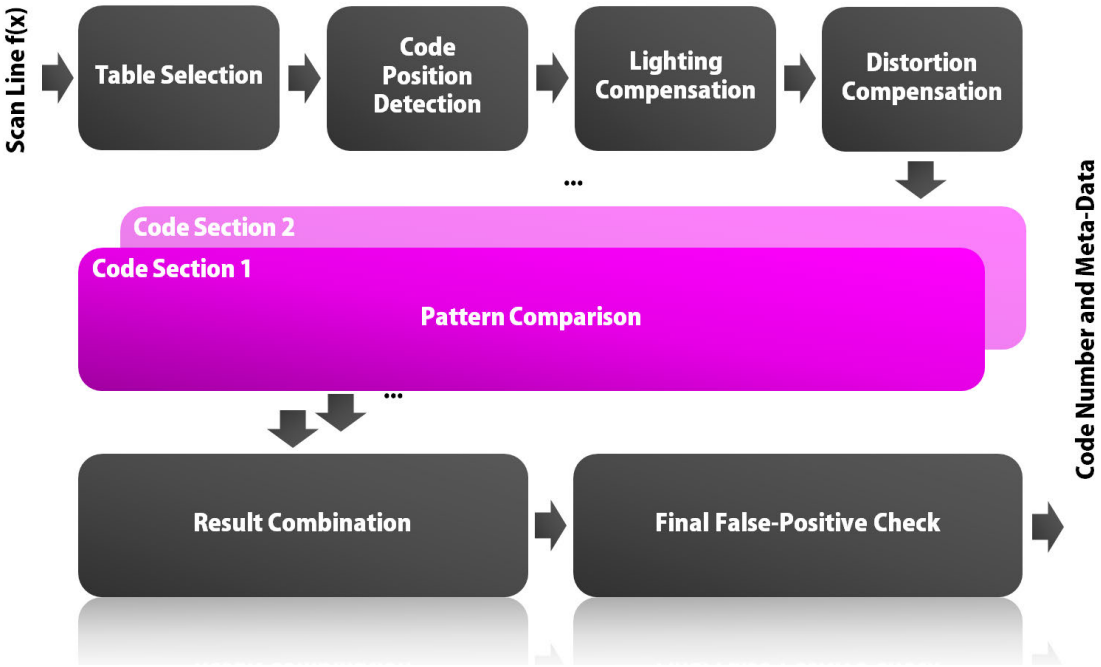


Figure 3.14 Components of the blurry bar code decoder.

Figure 3.14 illustrates the basic steps and components involved in decoding a blurry bar code. First, the appropriate recognition table is selected based on the measured sharpness of the signal in the provided waveform. Section 3.3.2 will provide further details on this and present the information contained in a recognition table in more detail. Afterwards, the bar code position in the complete waveform is searched for. Due to challenges inherent to the blurry waveform and the many possible sharpness levels of images, in combination with the requirement to locate the code position very precisely, this process is divided into two steps. First, the approximate position of the code waveform is detected. Afterwards, the exact bar code start and end position as well as the bar code symbology contained in the blurry waveform are determined in a combined step (Section 3.3.3). After extracting the waveform belonging to the bar code pattern, it is pre-processed in order to compensate for lighting effects such as shadows (Section 3.3.4) and geometric distortions caused by codes printed upside-down or on round surfaces (Section 3.3.5). The pre-processed and normalized waveform is then used as a base for extracting the waveforms belonging to different code sections, and for searching for the best fitting pre-calculated pattern from a recognition table for each section (Section 3.3.6). Based on the best fitting patterns found for each section, the final bar code number is then constructed using several measures to detect and rule our wrong code numbers (Section 3.3.7). Due to the many degrees of freedom, e.g., the physical code size, code geometry, perspective distortions, different codes symbologies and others, recognizing only correct and no wrong code numbers is challenging; especially in the case of blurry images. Therefore a final quality check is performed after a code number has been recognized in order to further reduce the number of false-positives (Section 3.3.8). The following paragraphs cover the relevant functionality of the presented components.

3.3.2 Recognition Tables and Table Selection

Like stated above, our approach consists of comparing pieces of the blurry code waveform with pre-calculated patterns. Recognition tables store the pre-calculated patterns and further data required for the recognition process. Specifically, they contain the following information (see Figure 3.15):

- Pre-calculated patterns for recognizing sets of digits in EAN13, UPC-A, EAN8 and UPC-E codes

- Pre-calculated patterns of code start-, end- and middle- patterns (delimiter patterns)
- Meta-data about the contained patterns, e.g., the exact relative start- and end-position of the patterns in the complete code waveform, the digits and parity values associated with each pattern, the total number of pre-calculated patterns as well as further information, including the PSFs used to generate the patterns)

The information in table files is organized and stored in such a way to ensure that,

- table files are as small as possible, since they have to be distributed to the mobile phones. Further implementation-related measures, e.g., table compression, support this (see Section 4.2.2).[12]
- table files can be loaded and organized in memory as fast as possible, in order to ensure fast startup times of the recognition engine on mobile phones. The information and patterns stored in the tables do not have to be further processed or re-arranged before they can be used in the recognition process.
- pattern comparison at a later stage can be performed as fast as possible. For that additional pre-calculated data is stored in table files. This includes versions of each pattern in different resolutions and pre-computed mean and standard deviation values for all patterns. Calculating this information at runtime would require too much time.

The information stored in table files has a two dimensional layout. Figure 3.15 shows an example of a table file and the basic table layout. Appendix 9.1.1 shows the detailed table layout. Usually, a set of 2 to 4 recognition tables covering different image sharpness levels is enough to allow for a robust recognition of bar codes in images of varying sharpness (see Section 4.2.2 for details). Pre-computed recognition tables are specific for mobile phone models or classes of mobile phones with similar optics.

[12] File size varies with the sharpness of pre-calculated patterns between 200kB and 900kB for a table file.

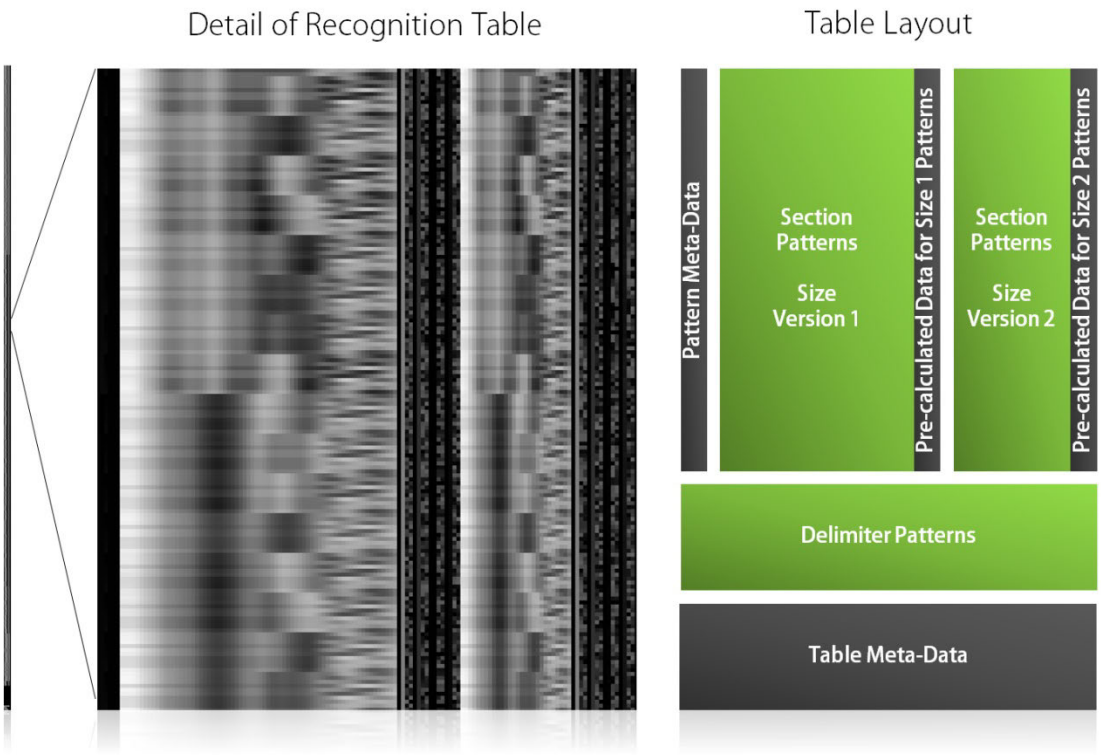


Figure 3.15 Example of a recognition table that contains blurry pre-calculated patterns (left image). Basic layout of information stored in a table (right image).

3.3.2.1 Table Selection Component

The first action performed by the blurry decoder is to decide which recognition table is applicable for the observed bar code waveform, because information stored in this table is required for most successive steps. A table is selected based on the shape and size of the PSFs used to pre-calculate its patterns. An ideal table-selection algorithm would determine the PSF underlying the given blurry scan line waveform and then select the table generated with the most similar PSF. Since due to the unknown underlying bar code number and existing imperfections this is very challenging and time consuming, we rely on a simpler selection method. We estimate the waveform's sharpness and select the appropriate recognition table based on this value. The sharpness value ranges from 0 for a "perfectly sharp" to 255 for a "perfectly blurry"[13] waveform. We tested several algorithms for determining the sharpness of a waveform, considering waveform characteristics such as standard deviation, the number of extreme points, and others, and found the method described in the following sub-section to be sufficiently precise for our use case.

[13] Corresponds to a horizontal line.

3.3.2.2 Sharpness Measurement Algorithm

First, we resize the original waveform $f(x)$ to a standard length of 640 pixels, since the length of the original scan lines varies with image resolution. Then, the resulting waveform is convoluted with a small kernel k_{p_1} in order to reduce noise:

$$f_{filtered}(x) = f(x) * k_{p_1} \; (see\ section\ 3.2.1\ for\ the\ def.of\ the * operator)$$

Based on $f_{filtered}(x)$, the biggest difference m between two consecutive values along the waveform is determined. In the case of a perfectly sharp signal consisting only of the values 0 and 255, this will be 255. In the case of a horizontal line (a "perfectly blurry" signal), we will obtain 0. In order to increase robustness, the above method is performed not on the complete waveform, but separately on $p_{num.of\ partitions}$ different partitions of the waveform. For each partition we obtain a value $m_x, 0 \leq x \leq p_{num.of\ partitions}$. The final sharpness value s is then calculated as:

$$s = 255 - \frac{1}{p_{num.of\ partitions}} \sum_{x=0}^{p_{num.of\ partitions}-1} m_x$$

$$p_{num.of\ partitions} \in \mathbb{N}, typically\ 4$$

This partitioning is necessary, since we do not know beforehand where exactly in the waveform the signal from the bar code pattern is. It reduces the influence of local effects such as varying sharpness along the scan line. For example, a blurry code with a sharp image background might result in a sharp signal at the waveform borders. Typical ranges for values of s are shown in Table 3.1.

Table 3.1 Conditions of images with sharpness values in the corresponding ranges:

Sharpness Range	Conditions of Bar Codes in Images
0...55	All bar code features are relatively sharp
100...145	Code features are slightly blurry
175	Thin bar code lines start to disappear
175...200	Code bars are not distinguishable any more, but human readable bar code numbers are still recognizable
215	Human readable numbers start to disappear
215...230	Very blurry images with unreadable numbers
230...245	Extremely blurry images

The obtained sharpness value is sufficiently precise to select the appropriate recognition table, even though it is not linear and varies due to lighting effects and the variety of elements potentially contained in a waveform besides the bar code pattern. The best mapping of ranges of sharpness values s to recognition tables is learned offline based on a test set of images. (See Section 4.2.2 for further details.)

3.3.3 Code Position and Symbology Detection

Since our recognition approach relies on extracting the waveform sections that belong to specific digits, it is crucial to determine the exact start and end position of the bar code pattern inside the waveform; ideally with an accuracy of 1-3 pixels in a waveform of 640 pixels length, in order to enable a fast and reliable search for the most suitable patterns. While this poses no problem in sharp waveforms, it is challenging in case of very blurry signals. Figure 3.16 shows a blurry waveform extracted from our example image. Issues complicating the exact code start and end position detection include:

- Image sharpness varies from sharp to very blurry images and has a strong effect on the shape of the obtained waveform.
- In blurry images, the transition from the silent area before and after a code to the start- and end-pattern is marked by a soft gradient of several (up to 25) pixels width (see lower-right image in Figure 3.16).
- It is hard to distinguish the area in the complete waveform, in which the bar code signal is located, from other elements such as text printed close to the code; especially in blurry images and since the exact size and position of the bar code in the waveform is unknown.
- In very blurry images, areas inside the bar code's waveform may appear that exhibit almost no features (extreme points). Such areas need to be distinguished from the silent areas adjacent to the bar code.

In addition to the challenges related to the code position detection, directly determining the symbology of codes in blurry images is not always possible. The number of visible extreme points in the code waveform varies and is not sufficient to distinguish code symbologies. In order to address these challenges, we separate the code position detection into two steps. A position detection step based only on the data available in the given waveform, and a following refinement of the found start and end position with the help of precalculated patterns stored in the recognition table.

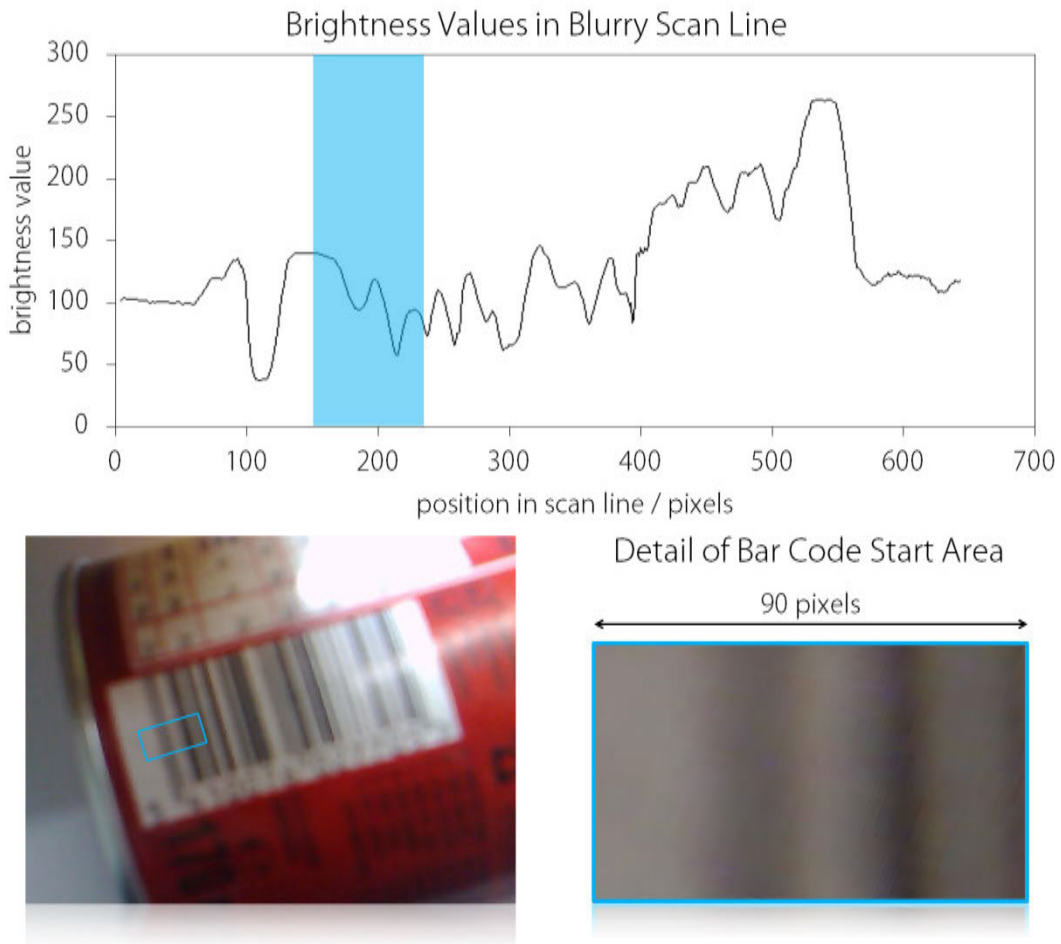


Figure 3.16 Blurry waveform extracted from the lower-left image (upper diagram) and close-up of the code's start area (lower-right image). The soft slope makes is hard to determine the exact code start position with an accuracy of1-3 pixels.

Figure 3.17 Detection of the approximate code start and end position.

3.3.3.1 Position Detection Algorithm

Input parameters for the code position detection algorithm are the waveform $f(x)$, resized to a standard length of 640 pixels and the sharpness value s determined in the previous recognition step. The algorithm is based on the position and type of extreme points found in $f(x)$. First, the original waveform $f(x)$ is convoluted with a Gaussian kernel k_{p_2}in order to reduce potentially present noise and artifacts from image sharpening. Based on the result $f_0(x)$, the first derivative $f_0\dot{(x)}$ is calculated. This waveform is again smoothed with a kernel k_{p_3} and too small values are deleted in order to remove less prominent edges:

$$f_1(x) = \begin{cases} (f_0\dot{(x)} * k_{p_3})(x) & (f_0\dot{(x)} * k_{p_3})(x) \leq p_{edge\ threshold} \\ 0 & (f_0\dot{(x)} * k_{p_3})(x) > p_{edge\ threshold} \end{cases}$$

The resulting waveform is in turn used to calculate the second deviation, which is also smoothed with a kernel $k_{p_3}$and finally produces $f_2(x)$:

$$f_2(x) = (f_1\dot{(x)} * k_{p_3})$$

Based on $f_0(x)$, $f_1(x)$ and $f_2(x)$, we calculate the position $slope_{pos}(i) \in \{0 \dots 640\}$ and type $slope_{type}(i) \in \{upward, downward\}$ of prominent slopes in $f(x)$. Figure 3.17 shows the position of detected slopes.

In a next step, the waveform is divided into three different sections: a middle area and the left- as well as right region adjacent to it (see Figure 3.17). Slopes located in the left section mark possible start positions of the bar code waveform, and slopes in the right area correspond to possible end positions. In most cases, the middle area will be located inside the bar code waveform. The underlying assumption is that the bar code waveform will take up a reasonable large portion of the complete waveform $f(x)$. Based on the detected slope positions $slope_{pos}(m_1) \dots slope_{pos}(m_n)$ in the central area, the average distance d_{slope} between adjacent slopes and a value d_{max} is calculated as:

$$d_{slope} = \frac{1}{n}\sum_{i=\,m_1}^{m_n-1} slope_{pos}(i+1) - slope_{pos}(i)$$

$$d_{max} = d_{slope} \cdot p_{distance\_factor}$$

$$p_{distance\_factor} \in \mathbb{R}, typically\ in\ range\ 1.5 \dots 3.0$$

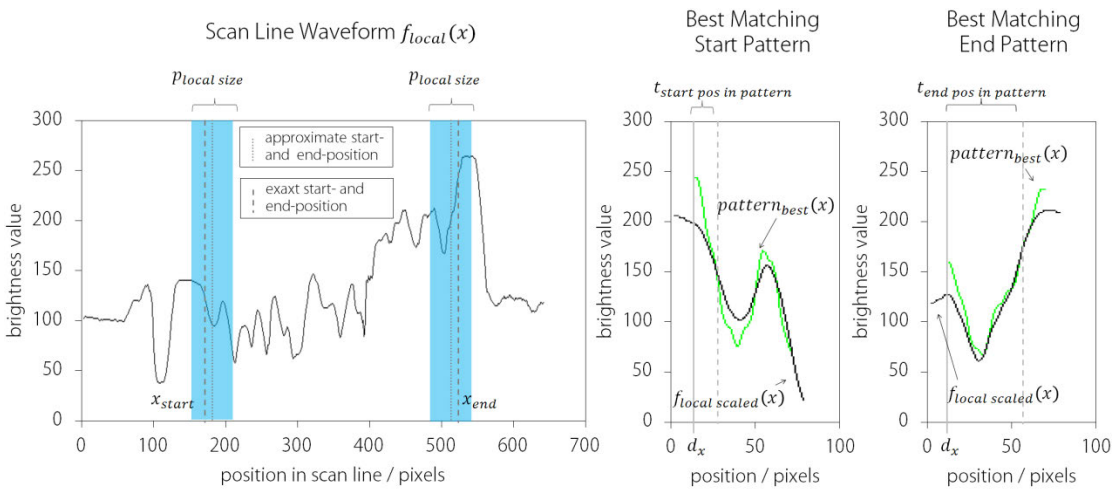


Figure 3.18 Illustration of the position refinement step. The extracted start- and end-areas (left diagrams) and best matching start- and end patterns (right diagram).

Based on the value d_{max}, we try to identify the slope positions in the left area that are likely candidates for the bar code start position. (The search for possible end positions in the right waveform area works accordingly.) This is done by iterating trough all slope positions in the left waveform area. A slope position $slope_{pos}(i)$ is a likely code start position, in the case it fulfills the following conditions:

1. The distance to the following slope position exceeds a certain limit:

   $$slope_{pos}(i+1) - slope_{pos}(i) \geq d_{max}$$

2. The slope has to be prominent enough:

   $$|f_1(i)| \geq p_{slope\ threshold}$$

3. If it is the correct slope type. For a start position it has to hold:

   $$f_1(x) < 0$$

Figure 3.17 shows all candidates for code start- and end-positions. Finally, we select the most likely candidate $slope_{pos}(i_{start})$ out of all possible ones. This is done by calculating a value $slope_{elevation}(i)$ for each candidate and selecting the candidate with the highest value as the bar code's start position x_{start}:

$$slope_{elevation}(i) = \sum_{k=slope_{pos}(i)-20}^{slope_{pos}(i)} f_0(k)$$

The underlying idea behind calculating the $slope_{elevation}(i)$ values is that the real code start and end position is adjacent to the code's silent areas, which are in most cases brighter than areas inside the code patterns that feature only few extreme points. Figure 3.17 shows the found code start- and end-positions candidates as well as the $slope_{elevation}(i)$ values of each position candidate. The optimal values for parameters used in this algorithm depend on the waveform's sharpness. They are therefore selected dynamically based on the estimated sharpness value s.

3.3.3.2 Position Refinement Algorithm

In the case of sharp waveforms, the detected code start and end position x_{start} and x_{end} are sufficiently precise. For blurry bar codes, this is usually not the case and requires a position refinement step. The following description focuses solely on the refinement process for the code's start position, but works accordingly for the bar code's end position. First, we select the waveform section $f_{local}(x)$ in a window of size $p_{local\ size}$ around the detected start position x_{start}. The recognition tables contain not only pre-calculated patterns for different digit combinations, but also patterns covering the bar code start-, end- as well as middle-pattern. Pre-calculated start patterns cover the start delimiter and the first code digit. In the case of EAN13 codes, this results in 10 different patterns. We use these to find the position in the extracted waveform section $f_{local}(x)$, where one of the pre-calculated patterns fits best.

Before doing this, we have to resize the extracted pattern $f_{local}(x)$ in order to ensure it has the same scale like our pre-calculated patterns. When pre-calculating all patterns, we assumed a certain bar code size in pixels. In a waveform of 640 pixels length, the pre-calculated pattern of the complete code had a length of 95* $t_{unit\ size}$ pixels[14]. The used value for $t_{unit\ size}$ is typically 5 pixels and stored in the recognition table. The pre-calculated start patterns cover the start delimiter (3 units), the first digit (7 units), as well as a certain part of silent area before the code (5 units), which results in a total pattern length of 15*$t_{unit\ size}$ pixels. In order to ensure that the extracted waveform has the same scale like the pre-calculated patterns, it is resized with a factor ρ:

$$\rho = \frac{p_{local\ size}}{x_{end} - x_{start}} \cdot (t_{unit\ size} \cdot p_{symbology\ size\ in\ units}(^{15}))$$

[14] The 59 different bars in EAN13 codes have a combined length of 95 units.

[15] 95 for EAN13 and UPC-A codes, 67 in the case of EAN8 and 51 for UPC-E codes

Having resized the local waveform to $f_{local\ scaled}(x)$, we calculate for each pre-calculated start pattern $pattern_{nr}(x)$ at each possible position in $f_{local\ scaled}(x)$ the difference between the two waveforms. For the comparisons, an optimized comparison algorithm based on the normalized cross-correlation [28] is used that is presented in Section 3.3.6. Compared to the regular cross-correlation, the normalized cross-correlation has the advantage of compensating for signal mean and scale variations due to lighting effects. Based on the difference values obtained, we search for the pattern $pattern_{best}(x)$ that fits best, and the relative position d_x in $f_{local\ scaled}(x)$ where it fits best. Figure 3.18 shows the extracted local waveforms, as well as the best found start- and end-pattern. The bar code start position x_{start} is then is the refined to:

$$x_{start} \leftarrow x_{start} - \left(\frac{p_{local\ size} \cdot \rho}{2} + d_x + t_{start\ pos\ in\ pattern}\right)\frac{1}{\rho}$$

$$t_{start\ pos\ in\ pattern} = 5 \cdot t_{unit\ size}$$
$$(paramter\ specifying\ the\ exact\ bar\ code\ start\ position\ inside\ the\ pattern)$$

3.3.3.3 Symbology Detection

Detecting the bar code symbology that is contained in a given waveform early on is important in order save time. In contrast to the sharp bar code decoder, in which a single decoding process can be performed very quickly, the decoding process for the blurry decoder takes considerably longer. This rules out the simple approach of trying all relevant symbologies successively.

Determining the code symbology contained in the waveform of a bar code pattern $f(x)$ is performed in several steps. If the waveform is relatively sharp, we analyze the number of detected extreme points between the code's start and end position and decide upon this value if it is an EAN13/UPC-A, EAN8 or UPC-E bar code. In the case of blurry images, this is not possible, since the number of extreme points can vary considerably according to the physical size of the underlying bar code or lighting conditions. In order to determine the code's symbology in such cases, information gathered by the code position refinement algorithm described above is used. The position refinement algorithm is executed several times, each time assuming another bar code symbology. For each code symbology, we collect not only the best start- and end-positions, but also information about how well the best fitting symbology-specific start- and end-patterns fit the bar code waveform. Based on these

values, we select the most likely contained symbology as the one with the best fitting start and end patterns. This approach works sufficiently well, even though EAN13 codes have the same start- and end-patterns like EAN8 bar codes. However, since EAN13 and EAN8 bar codes encode a different number of digits, their code length in terms of unit size differs. This results in differently resized sections $f_{local\ scaled}(x)$ of the original waveform at the position refinement step, which in turn has the effect that pre-calculated start- and end-patterns belonging to the correct symbology will fit best.

3.3.4 Lighting Compensation

After the exact bar code start- and end-position has been determined, the waveform section $f_c(x)$ between x_{start} and x_{end} that belongs to the bar code pattern is extracted, and a basic compensation for uneven lighting is performed on this waveform (see Figure 3.19). The algorithm used is targeted specifically for the compensation of soft shadows and uneven lighting. Based on the calculated mean values $m(x)$ around each pixel in $f_c(x)$, with a window size of 400 pixels, the adjusted new waveform $f_{ca}(x)$ is calculated as:

$$f_{ca}(x) = f_c(x) + (128 - m(x))$$

3.3.5 Code Distortion Compensation

Since our recognition approach relies on extracting the waveform sections that belong to certain digits, based on the detected code's start- and end-position, it is very sensitive to geometric distortions that can change the relative position of digits inside the bar code waveform $f_{ca}(x)$. Such distortions occur in images with a sideway perspective on the bar code, or bar codes printed on round and crumpled surfaces. However, due to our scan line-based approach, at least distortions due to vertically misaligned perspectives in relation to the bar code pose no problem, since such distortions have no influence on the waveform obtained along a scan line. The approach for distortion compensation that is described in the following is suitable of compensating arbitrary static distortions. Dynamically changing distortions, such as distortions occurring in the case of bar codes printed on a soft and constantly changing surface, cannot be addressed. In this section, we use the example of a bar code printed on a round surface to present the distortion compensation. However, the method can be directly applied to other types of static distortions as well.

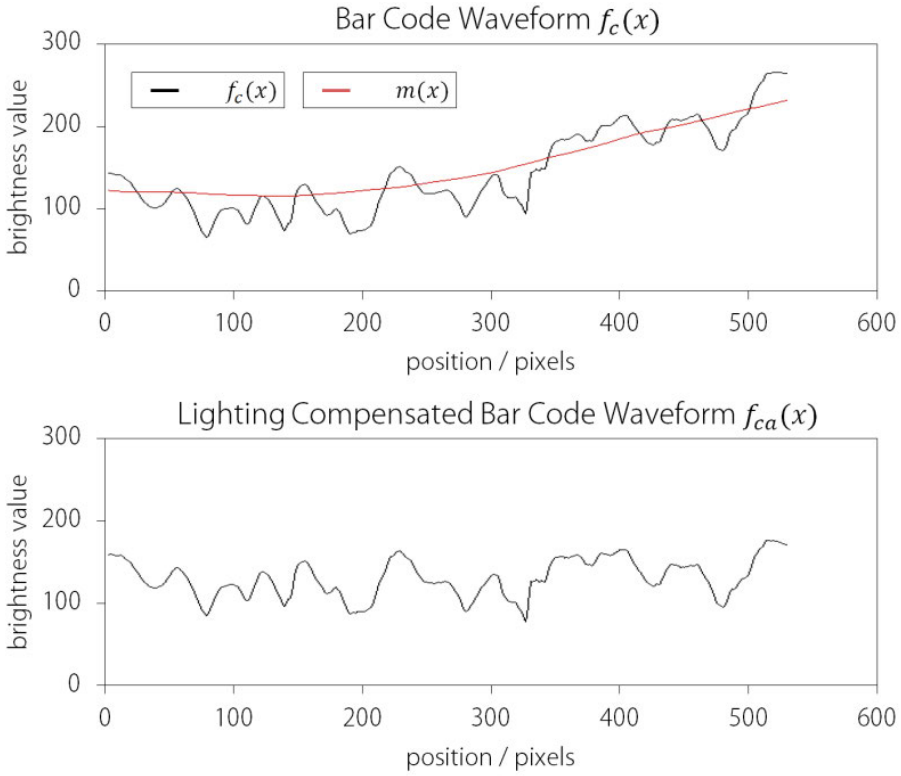


Figure 3.19 Original bar code waveform with m(x)-values (upper diagram) and lighting compensated result (lower diagram).

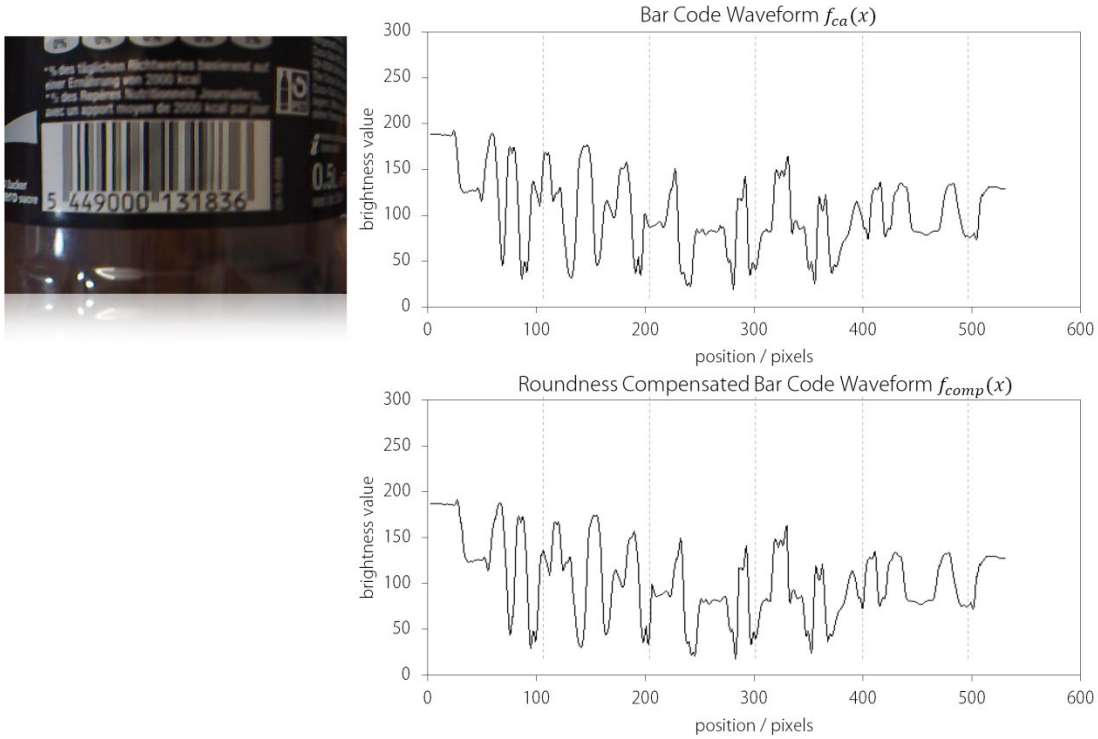


Figure 3.20 Original bar code waveform (upper-right diagram) extracted from a slightly blurry image of a bar code printed on a round surface (upper-left image). The lower-right diagram shows the distortion-compensated waveform that corresponds to the same bar code printed on a straight surface. The slight waveform shifts can be seen at the vertical dotted help-lines.

3.3.5.1 Distortion Compensation Algorithm

Figure 3.20 shows an example of a bar code printed on a round surface and the resulting waveform. For illustration purposes, we use a sharp image since it allows for a better monitoring of distortion effects. The algorithm stays the same for blurry images. The distortion effect on a code, even on a moderately round surface, results in offsets for the code's bars of up to 15 pixels compared to the same code on a flat surface. This is too much for the blurry decoding process that is able to compensate only for slight offset of up to 5 pixels. Modeling of the distortion based on the distance to the bar code, the roundness of the surface, and the perspective on the code is complex. Instead of modeling all details, we rely on measuring the occurring distortion.

In order to compensate for distortions, we then use the measured information stored in so-called *offset maps.* Offset maps are pre-calculated offline for different distortion types. An offset map $map_{dis.type}(x)$ contains for each pixel of the original waveform $f_{ca}(x)$ an offset value that describes how the pixel has to be moved in order to compensate for the distortion.

We created a Matlab tool that can automatically create offset maps from test images. In the case of round bar codes, two test images are taken, which contain patterns of equidistant lines that have the same physical size like a standard-sized bar code: One image $I_{round}(x, y)$ of the pattern on a round surface with a typical curvature, and one reference image on a straight surface $I_{straight}(x, y)$ (see Figure 3.21). From both images, horizontal scan lines and the brightness waveforms $f_{round}(x)$ and $f_{straight}(x)$ along these scan lines are extracted. After blurring both waveforms, the positions of extreme points that correspond to the n black bars $x_{round}(i)$ and $x_{straight}(i)$ are detected. Based on these, the values of the offset map $map_{round}(x)$ at the n bar positions is computed as:

$$map_{round}\big(x_{straight}(i)\big) = x_{round}(i) - x_{straight}(i) \quad \forall \quad 0 \leq i < n$$

In the case of round surfaces, all other values of $map_{round}(x)$ are interpolated by fitting a function $poly_3(x)$ of degree 3 to the already obtained offset values at the n test bar positions:

$$poly_3(x) = a_1 \cdot x^3 + a_2 \cdot x^2 + a_3$$

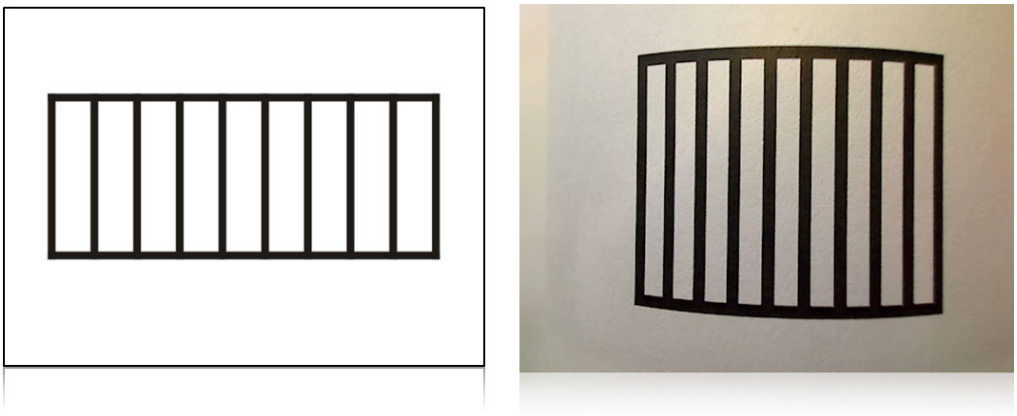


Figure 3.21 Test pattern used for pre-calculating offset maps that are used by the distortion compensation algorithm. Original pattern (left image) and the same pattern printed out and wrapped around a round surface (right mage).

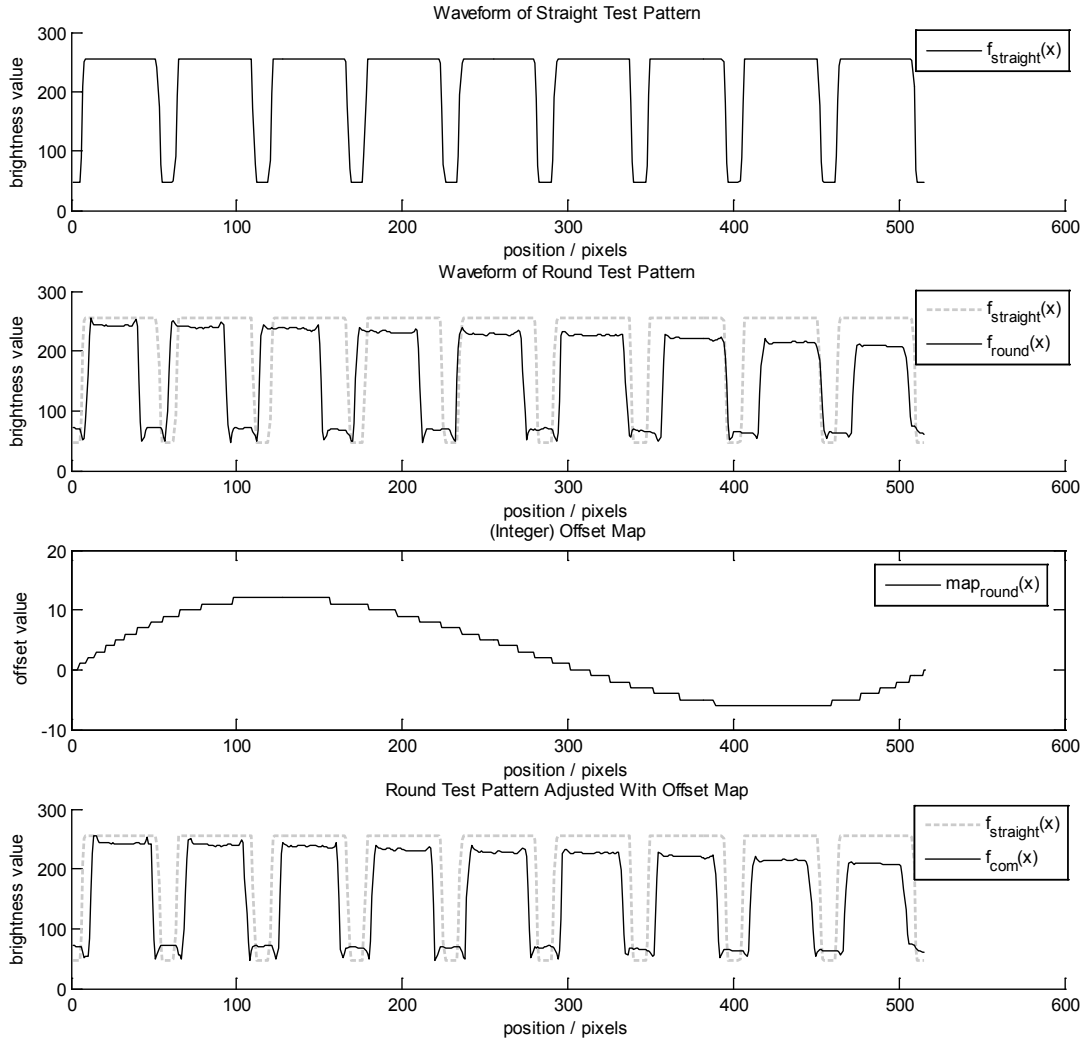


Figure 3.22 Waveform obtained from the straight (upper diagram) and round test pattern (second diagram from above). The offset map calculated from these two waveforms (second diagram from below) as well as the compensated waveform from the round test pattern using the determined offset map (lower diagram).

Figure 3.22 shows the waveform of the straight and round test pattern, the resulting offset map, as well as the corrected waveform of the round pattern using the information contained in the offset map. With the same approach, offset maps can be calculated for specific mobile phones and arbitrary surface geometries.

In order to compensate the distortion during the recognition process on the mobile phone, we resize the bar code waveform $f_{ca}(x)$ to its normalized length $l_{norm} = t_{unit\ size} \cdot p_{symbology\ size\ in\ units}$ and use the resulting waveform $f_n(x)$ to calculate the distortion-compensated waveform $f_{comp}(x)$ as:

$$f_{comp}(x) = f_n\big(x + map_{dis.type}(x)\big), \quad map_{dis.type}(x) \in \mathbb{Z} \quad \forall\, 0 \leq x < length(f_n)$$

In the case of codes on round surfaces, one important factor that influences the compensation result has not been considered so far: the position of the mobile phone in relation to the round code. In order to compensate for this, we pre-calculate not only one offset-map $map_{round}(x)$, but a set of offset-maps, each one assuming a slightly different perspective on the code. At runtime, we detect the most likely perspective of the mobile phone in respect to the round bar code and select the appropriate offset map. The perspective on the bar code is detected by considering the refined position of the bar code's middle pattern x_{middle}. This position is detected by the previous code position refinement step described in Section 3.3.3, according to the code's start- and end-position.

The approach of using pre-calculated offset maps has the advantage that many imperfections and the lens distortions of mobile phones are perfectly compensated for. Furthermore, correcting the distortions at runtime on the mobile phone is very simple and fast. Arbitrary static distortions can be compensated with this approach, as long as the test pattern used for the offset map generation has a high enough resolution to reproduce the distortion.

3.3.5.2 Distortion Detection

While the previous section presented means to compensate for code distortions, such distortions have to be detected in the first place. In the case of sharp or only slightly blurry images, most types of distortions can be detected based on the position and relative distance of extreme points found in the code waveform. This is not possible for blurry images. We tried different means to distinguish blurry bar codes on round surfaces or codes photographed from an angular perspective from well-aligned codes. All tested approaches turned out to work in general when making certain assumptions,

e.g., about the physical code size, but failed in the case of very blurry images or were not sensitive enough. Simply trying all possible distortion variants in parallel is too time-consuming. The approach currently taken to detect distortions utilizes the fact that the recognition is performed on video images with several frames per second. In the case of codes on round surfaces, we assume for each frame with a certain probability p_{round} that the code in the current frame is printed on a round surface and compensate the waveform for such a distortion. After the complete recognition process, a quality factor q is stored that indicates how well the best found code candidate fits. Based on these quality factors recorded from several previous frames, the value of p_{round} is dynamically adjusted.

When the presence of a bar code is first detected in the images, we assume with 66% probability that the code is not printed on a round surface and 33% probability that it is. After each frame, we monitor the quality factors. In the case the factors indicate that the best found code candidate fits better in frames, in which we compensated for round codes, p_{round} is changed from 33% to 66%. If in turn, results are better in the case no round bar code is assumed, we change p_{round} back to 33%. With this approach, the recognition of a single image requires the same time like before, but due to several frames per second in the case of video images, we have a chance of recognizing round bar codes. Furthermore, the dynamic adjustment of p_{round} ensures that over time the correct assumptions are made in most frames. The same approach is used in order to compensate for codes that are oriented upside-down in images. While working very well for only few types of distortions, this passive detection method has its limitations when the number of assumed distortions increases.

3.3.6 Pattern Comparison

The pattern comparison component is responsible for extracting the waveform sections belonging to different sets of digits, and for searching for the best fitting pre-calculated pattern for each section waveform. Input for this component consists of the length-normalized, lighting- and distortion-compensated bar code waveform $f_{comp}(x)$, the waveform's sharpness value s, as well as the selected recognition table t. Each of the four section waveforms $f_1(x) \dots f_4(x)$ that correspond to the bar code sections shown in Figure 3.13 contains the digit and parity information for three consecutive digits. These section waveforms are pre-processed and compared to all pre-calculated patterns.

3.3.6.1 Waveform Extraction and Preprocessing

Information about the length and relative start position of each section in $f_{comp}(x)$ is stored in the recognition table. Based on this information we can extract the waveform $f_{sec}(x)$ for each section. Despite the previous lighting compensation step, the signal in $f_{comp}(x)$ might still be distorted, e.g., to varying contrast in different image regions. Therefore, $f_{sec}(x)$ is pre-processed in order to obtain a normalized waveform that can be compared to the pre-computed patterns:

$$f_{sec}(x) \leftarrow 128 + \left(f_{sec}(x) - \overline{f_{sec}}\right) \cdot \frac{1}{std(f_{sec})}$$

$$std\left(f(x)\right) \stackrel{\text{def}}{=} \sqrt{\frac{1}{length(f)-1}\sum_{x=0}^{length(f)}\left(f(x)-\bar{f}\right)^2}$$

3.3.6.2 Pattern Comparison Challenges

Comparing the extracted section waveforms with the pre-calculated patterns poses two main challenges:

1. *Robustness and precision:* Especially in the case of blurry waveforms, pre-calculated patterns in the same sections differ only slightly. In combination with the presence of noise and slight distortions, the used comparison algorithm has to be very precise in order to detect the correct pattern. The biggest challenge are slight distortions or minor errors in the bar code position detection, which lead to shifted or differently scaled signals in $f_{sec}(x)$. The latter might fit, due to these errors, better to a pre-calculated pattern that corresponds to the wrong digit and parity combination and less well to the correct one.
2. *Performance:* In order to detect the closest pattern for all four section waveforms, we have to compare all 13000 pre-calculated patterns. Measures taken to increase the algorithm's robustness further increase the number of required comparisons. In combination with the very high comparison precision required, a naive approach would be computationally too time intensive for the real-time recognition of images on mobile phones.

3.3.6.3 Ensuring Robustness

Detecting the correct pre-calculated pattern for each section waveform $f_{sec}(x)$ is important. In the case a different pattern fits best, a wrong code number might be recognized. In order to increase robustness and avoid false detections due to slightly misaligned section waveforms, each waveform $f_{sec}(x)$ is cut out slightly larger than necessary. Each pre-calculated pattern $g_{nr\,sec}(x)$ is then compared at $p_{num.of\,pos}$ different (typically 5) positions to the section waveform $f_{sec}(x)$. For each $(f_{sec}(x), g_{nr\,sec}(x))$-pair we record the relative position (offset) $o_{nr\,sec}$ at which the pattern $g_{nr\,sec}(x)$ fits best, and consider the difference value $d\_min_{nr\,sec}$ at this position as the difference between these two waveforms.

In order to further increase robustness against misaligned section waveforms, we consider sections in a specific order. First, we search for the best fitting pattern in sections that are likely to produce more robust results, e.g., because fewer pre-calculated patterns exist for these sections. The resulting offset value $o_{nr\,sec}$ is then used to slightly adjust the position, where the waveform $f_{sec}(x)$ for the next considered section is extracted. For EAN13 codes, the order is: section 4, section 3, section 1 and then section 2 (see Figure 3.13 for section numbering). The comparison is started with section 4 since it has only 1000 pre-calculated patterns and is due to the included bar code end delimiter likely to produce more robust results compared to section 3.

Regarding the algorithm used to determine the similarity of two given waveforms, we found the normalized cross-correlation [28] to be the most robust approach. Figure 3.24 shows the extracted section waveforms $f_{sec}(x)$ and the best fitting pre-calculated patterns $g_{nr\,sec}(x)$ for all four sections. It can be seen that, except for section 2, the best matching pattern corresponds to the correct one. In section 2, two patterns fit slightly better than the pattern corresponding to the correct digits and parity values.

3.3.6.4 Ensuring Performance

In order to speed-up the overall comparison process while maintaining a high level of precision, several algorithmic measures have been taken that go beyond the implementation-specific optimizations presented in Section 4.1.3. In a first step, the number of necessary pattern comparisons is minimized:

1. The comparison process is aborted as soon as possible. After each section, we analyze the five best-fitting pre-calculated patterns and abort the recognition process if the best found pattern fits not well enough,

or if all five best matching patterns fit equally well. In the latter case, the chance of recognizing a wrong pattern is too high.

2. For each section, we analyze the parity values $\overrightarrow{dp_0} \dots \overrightarrow{dp_4}$ that correspond to the five best matching pattern. When processing the next section, we consider not all pre-computed patterns, but limit the comparison to patterns with corresponding parity values that result in a valid parity pattern when combined with any of the values in $\overrightarrow{dp_0} \dots \overrightarrow{dp_4}$.
3. In the case additional knowledge about the bar code numbers that should be recognized is available, this information can be used to limit the number of patterns that have to be compared. For example, this is the case if only books (all bar codes of books start with "978"), or a limited set of products with known bar code numbers should be recognized.

Besides reducing the total number of required comparison steps, two measures have been taken to accelerate the comparison process itself:

1. A hierarchical approach is used to search for the five best matching patterns to a given section waveform: In each step we compare a set $\overrightarrow{g\_input_{step}}$ of n pre-computed patterns with a specific comparison function φ to the section waveform $f_{sec}(x)$. This results in a set $\overrightarrow{g\_output_{step}}$ of the m best fitting patterns with $m < n$. Patterns in each step are not compared in their original resolution with a length of $l = length(g_{nr\,sec}(x))$, but in a reduced resolution and a length of $l_{reduced} = l \cdot t_{resolution\,factor}$ with $0.0 \leq t_{resolution\,factor} \leq 1.0$. The output of one step is used as input for the next one, successively reducing the number of the most similar patterns to $f_{sec}(x)$ in $\overrightarrow{g\_output}$. The underlying idea is to compare the large number of initial patterns in low resolutions and with a very fast, but less precise comparison function, and to compare only a subset of well-fitting patterns in a higher resolution with a more precise comparison function. Figure 3.23 shows the three steps currently used and the applied parameters for m and $t_{resolution\,factor}$.
2. The implementations of the three comparison functions $\varphi_1 \dots \varphi_3$ used in the different steps are highly optimized and as much information as possible has been pre-calculated. For example, recognition tables contain already pre-processed versions of comparison patterns in reduced resolutions, as well as the mean and standard deviation values for each pattern.

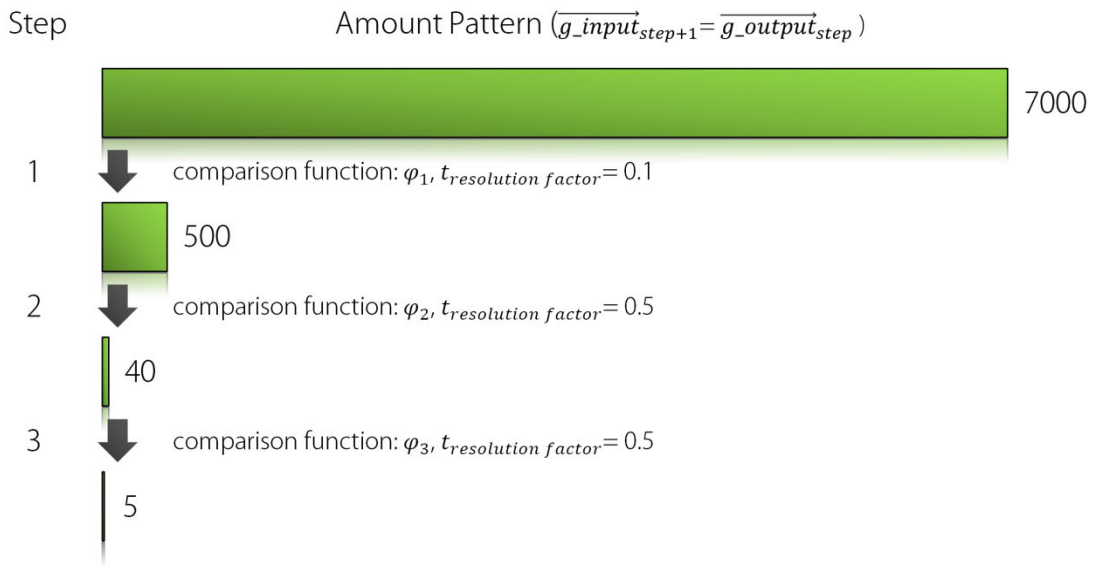


Figure 3.23 Hierarchical approach used to speed-up pattern comparisons. Patterns are compared in successive steps, in a reduced resolution and with different comparison functions.

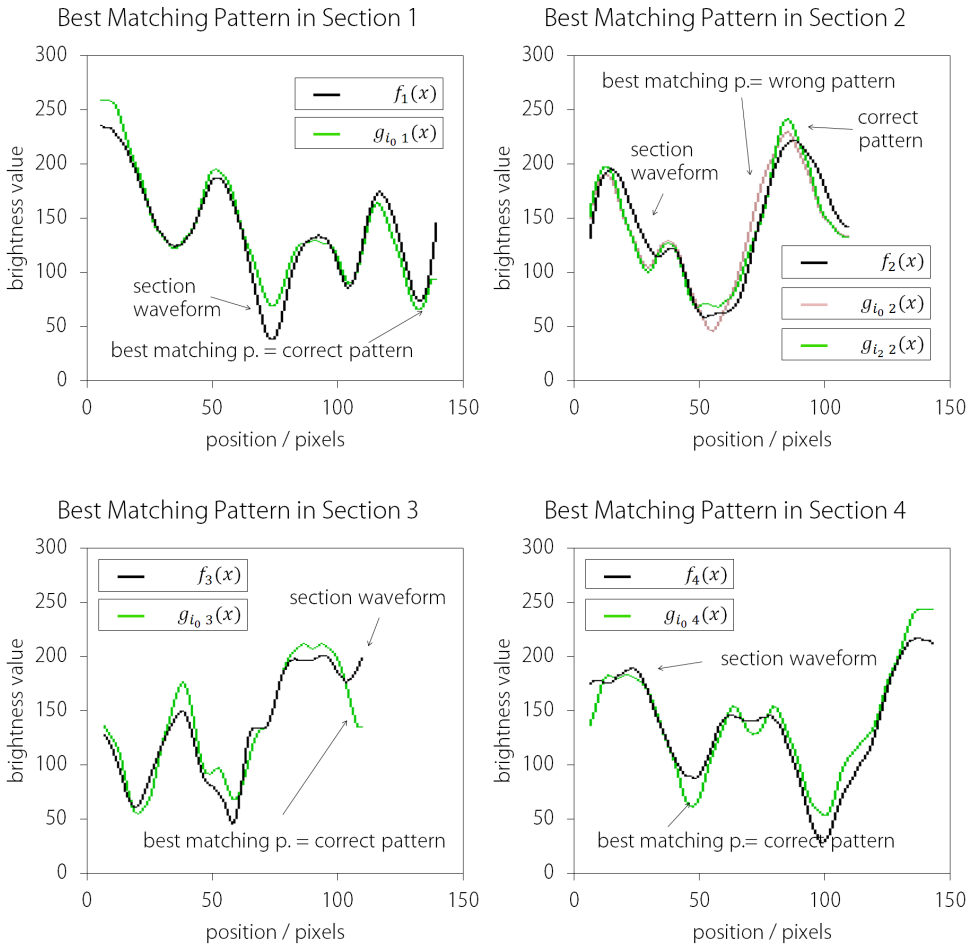


Figure 3.24 Section waveforms and their best fitting, pre-calculated patterns. In section 2, not the pattern corresponding to the correct digit and parity combination for our test code was found as the best fitting pattern, but a different one.

In the following, the three algorithms $\varphi_1 \ldots \varphi_3$ used for pattern comparisons are presented. For clarity reasons, we abstain from the fact that the pattern waveform $g = g_{nr\,sec}(x)$ is compared at different regions inside the section waveform $f = f_{sec}(x)$ and assume both waveforms have the same length.

3.3.6.5 Comparison Algorithm 1

The first comparison algorithm is used to compare the most patterns and has therefore been optimized for speed and the least possible memory access. It is simple, but sufficient to eliminate the majority of patterns that do not fit at all:

$$\varphi(f,g) = \sum_{x=0}^{lenght(f)-1} |f(x) - g(x)|$$

3.3.6.6 Comparison Algorithm 2

The second algorithm works similar to the first one, but in addition considers the first derivative of $f(x)$ and $g(x)$. Considering the first derivatives puts more focus on the correct type and position of extreme points, which increases robustness compared to algorithm 1. Factors describing how much influence the comparison result of the first derivatives has compared to the result when comparing the original waveforms have been determined experimentally on test sets of images. The deviation of the comparison pattern $g\dot{(}x)$ is calculated at runtime, since this can be done very fast and pre-calculating these values would double the memory requirements of recognition tables:

$$g\dot{(}x) \stackrel{\text{def}}{=} \sum_{x=0}^{length(g)-2} g(x+1) - g(x)$$

The deviation of the section waveform $f\dot{(}x)$ has to be calculated only once for all comparisons with pre-calculated patterns in a section. The pattern difference is then calculated as:

$$\varphi(f,g) = 0.2 \cdot \left(\sum_{x=0}^{lenght(f)-1} |f(x) - g(x)| \right) + 0.8 \cdot \left(\sum_{x=0}^{lenght(f)-2} \left|f\dot{(}x) - g\dot{(}x)\right| \right)$$

3.3.6.7 Comparison Algorithm 3

Algorithm 3 is used in the final comparison step and is therefore the most precise. This algorithm is based on a variant of the normalized cross-

correlation $\gamma(f,g)$ [29]. Similar to algorithm 2, pattern similarity is calculated based on the waveforms and their first derivations:

$$\varphi(f,g) = \frac{\gamma(f,g) + \gamma(\dot{f},\dot{g})}{2}$$

$$\gamma(f,g) \stackrel{\text{def}}{=} \frac{\sum_{x=0}^{length(f)-1}\left(\left(f(x)-\bar{f}\right)\cdot\left(g(x)-\bar{g}\right)\right)}{\sqrt{\left(\sum_{x=0}^{length(f)-1}\left(f(x)-\bar{f}\right)^2\right)\cdot\left(\sum_{x=0}^{length(g)-1}(g(x)-\bar{g})^2\right)}}$$

Only the best matching patterns are compared with this algorithm, but still around 50 * $p_{num.of\ sections}$ * $p_{num.of\ pos}$= 50 * 4 * 5 = 1000 comparisons on the 50%-sized patterns are performed. Applying the standard formula for calculating the normalized cross-correlation shown above would be too time consuming. We therefore leverage the fact that the pre-calculated patterns $g(x)$ are known beforehand, and use this information to optimize the calculation of $\gamma(f,g)$ by pre-computing as much information as possible:

$$\gamma(f,g) \stackrel{\text{def}}{=} \frac{\sum_{x=0}^{length(f)-1}\left(\left(f(x)-\bar{f}\right)\cdot\left(g(x)-\bar{g}\right)\right)}{\sqrt{\left(\sum_{x=0}^{length(f)-1}\left(f(x)-\bar{f}\right)^2\right)\cdot\left(\sum_{x=0}^{length(g)-1}(g(x)-\bar{g})^2\right)}}$$

$$\stackrel{\text{def. } std()}{\Longleftrightarrow} \frac{1}{length(f)}\cdot\sum_{x=0}^{\overbrace{length(f)-1}^{n}}\frac{\left(f(x)-\bar{f}\right)\cdot\left(g(x)-\overline{g}\right)}{std(f)\cdot std(g)}$$

$$\Leftrightarrow \underbrace{\frac{1}{length(f)\cdot std(f)\cdot std(g)}}_{c_1}\cdot\sum_{x=0}^{n} f(x)g(x) - \bar{g}f(x) - \bar{f}g(x) + \bar{f}\bar{g}$$

$$\Leftrightarrow c_1\cdot\underbrace{\left(\sum_{x=0}^{n}-\bar{g}f(x)\right)}_{c_2}\cdot\underbrace{\left(\sum_{x=0}^{n}\bar{f}\bar{g}\right)}_{c_3}\cdot\left(\sum_{x=0}^{n}f(x)g(x)-\bar{f}g(x)\right)$$

$$\Leftrightarrow c_1\cdot c_2\cdot c_3\cdot\left(-length(f)\cdot\bar{f}\cdot\sum_{x=0}^{n}g(x)\right)\cdot\left(\sum_{x=0}^{n}f(x)g(x)\right)$$

$$\Leftrightarrow c_1\cdot c_2\cdot c_3\cdot\left(-length(f)\cdot\bar{f}\cdot\bar{g}\cdot length(g)\right)\cdot\left(\sum_{x=0}^{n}f(x)g(x)\right)$$

$$\overset{\substack{length(f)=\\ length(g)}}{\Longleftrightarrow} c_1 \cdot c_2 \cdot c_3 \cdot \underbrace{\left(-length(f)^2 \cdot \bar{f} \cdot \bar{g}\right)}_{c_4} \cdot \left(\sum_{x=0}^{n} f(x)g(x)\right)$$

$$\Leftrightarrow \underbrace{c_1 \cdot c_2 \cdot c_3 \cdot c_4}_{c_{pre-computed}} \cdot \left(\sum_{x=0}^{n} f(x)g(x)\right)$$

Finally, $c_{pre-computed}$ can be further transformed, to produce the final equation:

$$\gamma(f,g) = c_{pre-computed} \cdot \sum_{x=0}^{length(f)-1} f(x) \cdot g(x)$$

$$c_{pre-computed} = \frac{1}{std(g)} \cdot \bar{g}^3 \cdot c$$

$$c = \frac{length(f)^4 \cdot \bar{f}^3}{std(f)}$$

The values for $1/std(g)$ as well as $\bar{g}$ for each pre-computed pattern are already stored in the recognition tables, and the value of c has to be computed only once for each size version of a section waveform. This drastically accelerates the computation of the normalized cross-correlation in our case.

3.3.7 Result Combination

Table 3.2 shows the output of the pattern comparison component for our example bar code. It contains information about the five best matching patterns in each bar code section. For the n-th best matching pattern in section i, information includes the three digits $\vec{d}(n,i) = d_1(n,i) \ldots d_3(n,i)$ corresponding to the pattern, digit's parity values $\vec{p}(n,i) = p_1(n,i) \ldots p_3(n,i)$, and a difference value $diff(n,i)$ describing how well the pattern fits the section waveform. The offset value $o(n,i)$ indicates at which relative position the pattern fits the section waveform best. The result combination component tries to construct a correct code number out of this information. In the case of sharp images, the patterns that correspond to the correct digit and parity values will most likely be the best matching patterns in each section. For very blurry images, this is not always the case and therefore requires an approach to detect the correct pattern combination.

Table 3.2 Output of the parameter comparison component: Information about the five best matching patterns in each code section. In sections 1, 2, and 4, the best matching patterns correspond to the correct digit and parity values for our example bar code (5410076027705). In section 2 the correct pattern is the third best fitting:

Parameters	Section 1	Section 2	Section 3	Section 4
best fitting patterns (n = 1)				
$\boldsymbol{diff(1,i)}$	293	362	414	432
$\vec{\boldsymbol{d}}\boldsymbol{(1,i)}$	(4 1 0)	(0 3 6)	(0 2 7)	(7 0 5)
$\vec{\boldsymbol{p}}\boldsymbol{(1,i)}$	(o e e)	(o o e)	(o o o)	(o o o)
$\boldsymbol{o(1,i)}$	0	0	-2	0
second best fitting patterns (n = 2)				
$\boldsymbol{diff(2,i)}$	397	368	459	482
$\vec{\boldsymbol{d}}\boldsymbol{(2,i)}$	(4 1 4)	(0 7 8)	(0 9 7)	(3 0 5)
$\vec{\boldsymbol{p}}\boldsymbol{(2,i)}$	(o e o)	(o o e)	(o o o)	(o o o)
$\boldsymbol{o(2,i)}$	0	0	-2	0
third best fitting patterns (n = 3)				
$\boldsymbol{diff(3,i)}$	444	389	575	525
$\vec{\boldsymbol{d}}\boldsymbol{(3,i)}$	(4 1 9)	(0 7 6)	(0 2 8)	(7 9 5)
$\vec{\boldsymbol{p}}\boldsymbol{(3,i)}$	(o e o)	(o o e)	(o o o)	(o o o)
$\boldsymbol{o(3,i)}$	2	0	-2	-2
fourth best fitting patterns (n=4)				
$\boldsymbol{diff(4,i)}$	455	490	600	541
$\vec{\boldsymbol{d}}\boldsymbol{(4,i)}$	(4 1 2)	(9 3 6)	(0 9 3)	(3 6 5)
$\vec{\boldsymbol{p}}\boldsymbol{(4,i)}$	(o e o)	(o o e)	(o o o)	(o o o)
$\boldsymbol{o(4,i)}$	0	0	-2	0
fifth best fitting patterns (n=5)				
$\boldsymbol{diff(5,i)}$	537	531	607	572
$\vec{\boldsymbol{d}}\boldsymbol{(5,i)}$	(2 1 0)	(0 3 8)	(0 2 3)	(3 9 5)
$\vec{\boldsymbol{p}}\boldsymbol{(5,i)}$	(o e e)	(o o e)	(o o o)	(o o o)
$\boldsymbol{o(5,i)}$	0	0	-2	0

Table 3.3 The 10 best found code candidates and their difference value that indicates how likely they are. A smaller difference value corresponds to a more likely code:

Code Candidate	Difference Value $\boldsymbol{diff_{candidate}}$
5410076027705	397
5410038028805	781
5410078023705	1325
5210076027305	1480

5410936028705	1599
5410076097395	1670
5410076028795	1703
5210078027365	1708
5410936027395	1810
5210078027365	1813

In a first step, a list containing all possible $5^4 = 625$ complete bar code number candidates is constructed. A single candidate $\overrightarrow{cand}(n_1, n_2, n_3, n_4), 0 \leq n_x < 5$ is constructed as:

$$\overrightarrow{cand}(n_1, n_2, n_3, n_4) = \langle d_{system}, \vec{d}(n_1, 1), \vec{d}(n_2, 1), \vec{d}(n_3, 1), \vec{d}(n_4, 1) \rangle$$

$$d_{system} = sytem\ digit\ encoded\ in\ \vec{p}(n_1, 1)\ and\ \vec{p}(n_2, 1)$$

For all possible candidates, we calculate the check digit and remove the ones that are not valid. For each remaining candidate, we calculate a difference value and sort the candidates list according to this value, with the most likely candidate that features the smallest difference value on top. The underlying idea behind this value is to promote candidates that are likely to correspond to correct code numbers, and penalize the ones that are likely to correspond to wrong code numbers. The difference value of a candidate is based on how well its corresponding patterns fit the section waveforms and is modified by factors $\theta_1 \ldots \theta_m,\ \forall\ \theta_x \in \mathbb{R}_+$:

$$diff_{candidate}\left(\overrightarrow{cand}(n_1, n_2, n_3, n_4)\right) = \left(\frac{1}{4} \cdot \sum_{i=1}^{3} diff(n_i, i)\right) \cdot \theta_1 \cdot \ldots \cdot \theta_m$$

Conditions influencing the factors $\theta_1 \ldots \theta_m$ include to following:

- The more patterns in any section fit better than the ones belonging to the analyzed candidate, the less likely it is that this candidate represents the correct code number
- If all difference values for the candidate's patterns in the different sections are similar, this is better than situations with relatively bad fitting patterns in some sections and very good fitting patterns in others. The latter might indicate problems in sections with high difference values.

- It is better if all offset values of the candidate's patterns are consistent. Very large differences between two consecutive offset values indicate that at least one of the detected patterns is incorrect.

Table 3.3 shows the 10 best fitting code candidates, sorted according to their difference values. All candidates with a difference value smaller than a fixed threshold value $d_{diff\ max}$ are passed on to the next component that performs a final check in order to further reduce the number of false-positives.

3.3.8 Final False-Positive Check

With the measures taken in the previous result combination component, the number of false-positives on a difficult set of test images can be reduced to around 2-3%, while staying sensitive enough to recognize bar codes also in non-perfect conditions. However, when lowering the $d_{diff\ max}$ parameter further, such that less false-positives are recognized, the number of correctly recognized codes is also drastically reduced. In order to overcome this problem, an independent method of distinguishing correct from wrong code numbers is required that relies not on the results directly obtained from the pattern comparison component. One remaining source of uncertainty in the detected code numbers is related to our approach of comparing not a complete bar code waveform but separate independent waveform sections. This approach leads to two effects: In the case of blurry, pre-calculated patterns for section 1 and 4, the waveform at the right respectively left border, and in the case of patters for sections 2 and 3 the waveform at both borders, will be influenced by the digit adjacent to the pattern. This effect is already addressed during the pattern comparison stage by comparing not the full pre-calculated patterns, but slightly smaller pattern pieces. Nevertheless, this leads to the situation that the waveform areas at the intersection of two sections have not been considered in our recognition method. The second effect is related to the fact that we normalize section waveforms independently from each other before the pattern comparison step, which results in lost information about the relative scale of section waveforms between each other.

In order to address these uncertainties related to our section-based approach and to further reduce the number of false-positives, a final check is performed on the best-rated code number candidates. For each of these code numbers, the complete blurry code waveform is constructed, based on information about the PSFs stored in the selected recognition table. Figure 3.25 shows the constructed waveform in the case of our most likely code candidate

(5410076027705) and the original code waveform obtained from the image. Each constructed waveform is then compared to the original blurry code waveform, and a final bar code number is selected based on the results:

- If all candidate waveforms fit equally well, we abort the recognition and return no bar code, since the chance of picking a wrong code number is too high.
- If the constructed waveform corresponding to the first (most likely) number candidate fits best and the others fit considerably worse, we can be sufficiently sure to have recognized the correct code number.

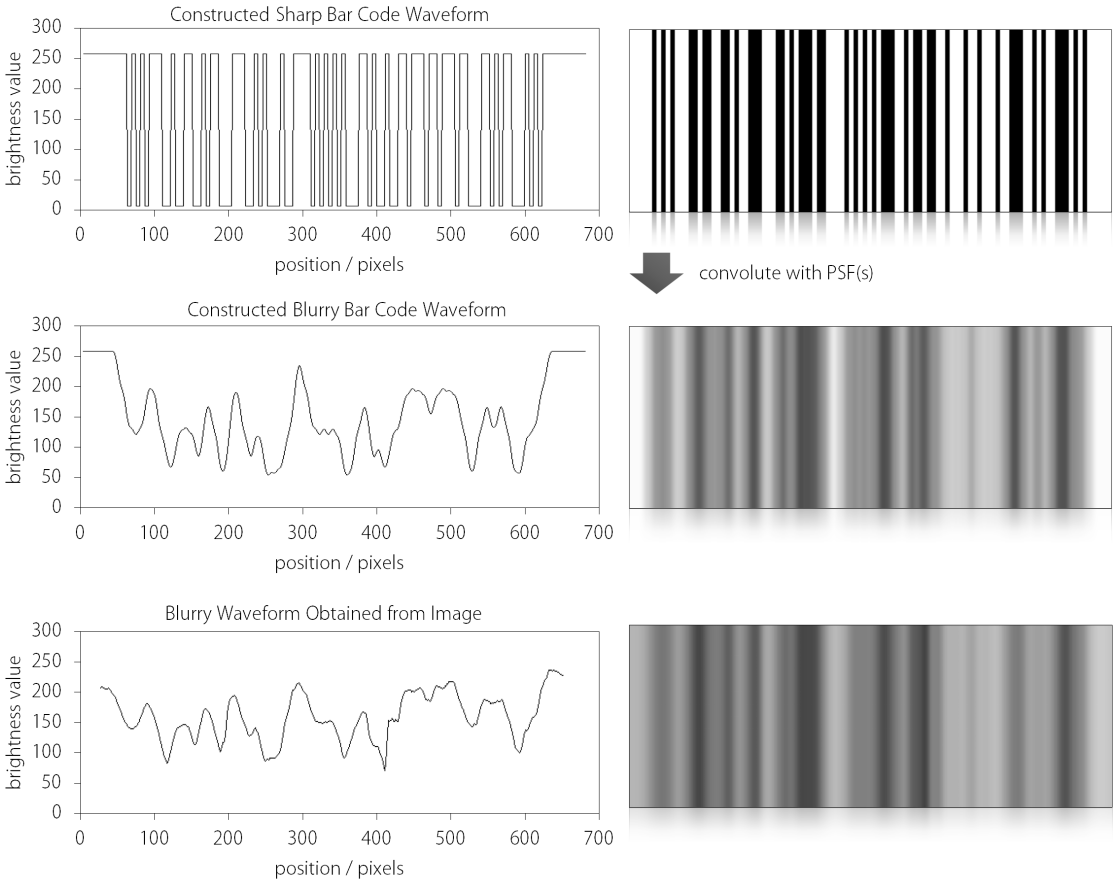


Figure 3.25 Final false-positive check by constructing the blurry code waveform that corresponds to the most likely code candidate (upper and middle diagram) and comparing it to the blurry waveform obtained from the image (lower diagram). The patterns on the right side of each diagram correspond to the brightness values shown in the diagrams.

Figure 3.26 Nokia 6630 Symbian smartphone with laser scanner attachment for mobile phones.

3.4 Related Work on Bar Code Recognition

Because bar codes are in use since the 1970ties, a variety of dedicated devices for their recognition, such as laser scanners, CCD devices, wands, and slot scanners are available. However, the process of recognizing and decoding bar codes in a reliable way remains a non-trivial task up to this day, even with these specialized devices [30, 31]. In addition to these devices, also vision-based systems have been implemented for the decoding of bar codes in automated document processing or logistics [32, 33]. However, these systems typically employ specialized camera equipment and are used in a controlled environment, e.g., with respect to the lighting situation, in order to scan bar codes in documents or on packages – this is very different from a camera phone held in the hand of a consumer in a supermarket aisle, scanning a product on a shelf. Regarding the recognition of bar codes on mobile phones, some algorithms have been proposed, and in recent years a number of commercial solutions emerged. This section presents related work for the optical recognition of bar codes, with a focus on the recognition of codes on mobile phones. Some of the following information has already been published in [22] and [14].

3.4.1 Specialized Recognition Systems

A number of professional systems for the visual decoding of bar codes are available [34, 35]. These solutions typically consist of complete systems, comprising software and appropriate hardware, such as professional cameras and lighting, and are highly optimized for particular use cases. Examples include scanning stations for letters or software specialized in recognizing bar codes in scanned documents [36] or specific code symbologies [37]. In addition, other approaches such as recognizing the human readable characters often printed below bar codes have been presented [38]. Although they offer good recognition rates, these systems often require special conditions, such as sharp and high-resolution images or controlled lighting. Furthermore, their requirements in terms of system resources are in general too demanding for mobile phones, particularly their execution in real-time on these devices.

An alternative to the optical recognition of bar codes with mobile phones is the use of small laser scanners that can be attached to selected phone models [39, 40]. However, the additional energy consumption of these devices as well as the fact that they have to be attached manually to the mobile phone before scanning a product poses a serious hindrance in consumer-oriented application scenarios (see Figure 3.26).

3.4.2 Academic Work

In the academic literature, bar code recognition was discussed in several publications. Most solutions are based on the Hough transform [23] as proposed by Youssef and Salem [41] or Muniz et al. [42], and some rely on neural networks to handle imperfect codes [43]. Furthermore, several methods have been published for specific problems such as locating bar codes in images [11, 44-47], the direct operation on (JPEG) encoded images [48] or recognizing codes in the DCT (discrete cosine transform) domain in order to speed-up processes [49]. While being robust against noise and local problems such as dirty of damaged codes, these methods are sensitive to perspective distortion and codes printed on round or otherwise distorted surfaces. Even though these recognition algorithms offer in general a good recognition rate, they are usually optimized for use cases different from the recognition of codes on mobile phones, and therefore often rely on sharp images and are computationally too expensive for an implementation on mobile devices.

As an alternative, several algorithms have been proposed, specifically for mobile phones. In order to compensate for the often limited processing power

of mobile devices, many of these algorithms do not analyze the entire image. Instead, they process only the information along certain scan lines through the bar code. Using this approach has the drawback that a bar code's position and orientation has to be detected beforehand, in order to position the scan lines in the image. Furthermore, these algorithms are affected by local image defects as a result of reflections, crumpled or damaged bar codes, as well as low-quality images that exhibit noise and low contrast. As already stated in [14], there are algorithms that have been targeted specifically at mobile devices, e.g., by Chai and Hock [46], Ohbuchi et al. [50] and others [51, 52], but these have not been implemented or evaluated on real camera phones. An exception is the recent work by Li and Zeng [53], who presented and implemented an algorithm for the recognition of bar codes on the iPhone. However, the full image binarization, sharpening, and edge-detection steps required by their method are likely to produce performance problems on less powerful devices. The biggest difference between all of these solutions and our algorithm is the fact that they require sharp images and therefore devices with built-in autofocus cameras. In practice, attaching macro lenses that result in sharp images of close objects to the phone, each time a code should be scanned, is not a viable option.

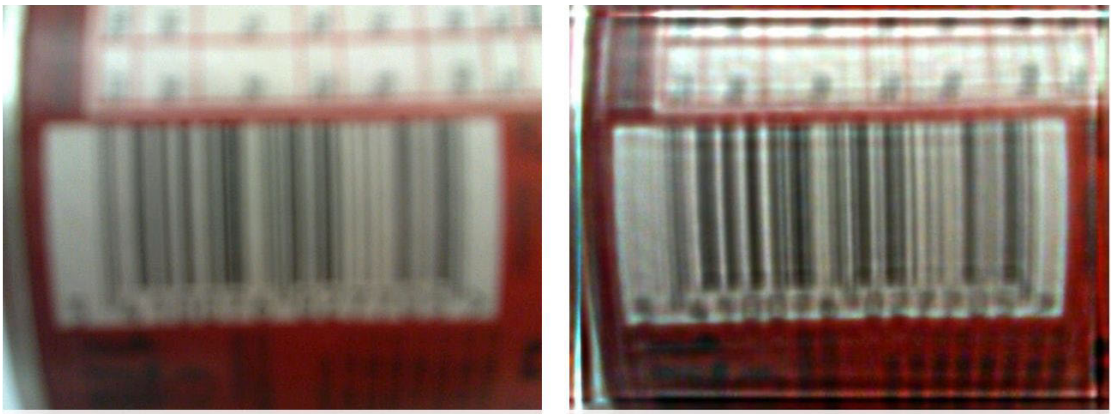


Figure 3.27 Result (right image) when using Matlab to apply a deconvolution with the Lucy-Richardson method to the blurry image on the left side, using a fixed, beforehand optimized PSF.

There are several approaches available for the sharpening of blurry images. Algorithms such as general deconvolution algorithms [54-57] or algorithms that take certain characteristics of bar codes into account [58, 59] can help. However, even with specialized solutions such as [59], the images remain in general too blurry for a robust recognition. Blind deconvolution approaches

[60] that additionally try to automatically estimate the underlying point spread function (PSF) are also a powerful tool for image enhancement, but perform in our case no better. Figure 3.27 shows the result of a deconvolution using the Lucy-Richardson method [61] and a known point spread function (PSF). Theoretical work by Joseph and Pavlidis covers the problem of bar code detection using peak locations in blurred waveforms, as obtained when scanning bar codes with a laser scanner [62, 63]. The proposed methods are well suited for laser beam scanners but do not consider effects such as uneven lighting or code position detection and are not optimized for recognition speed.

Another interesting option for recognizing blurry bar codes has been presented by Esedoglu [64] and Wittman [31]. Like in [62, 63], these approaches are based on modeling a blurry bar code signal as the signal of a sharp code convoluted with a known PSF of the camera used to take the image. Compared to the general convolution approaches presented above, these approaches additionally consider knowledge about the structure of bar codes. The correct bar code number is determined using an analytical approach that tries to find the code number belonging to the sharp signal that produces the closest results to the recorded blurry signal from the camera when convoluted with the known PSF. Like the general or even most optimized deconvolution methods, these approaches have in general two difficulties when applied for the recognition of bar codes on mobile phones. First, the underlying model is often too simple to meet the conditions found on mobile devices, and second, the proposed analytical search used to determine the underlying bar code number is computationally too intensive for the recognition in video images with multiple frames per second (FPS) on standard mobile phones[16]. Assumptions made that are usually not met in our scenario include the following:

- *Gaussian-shaped PSFs:* The PSF obtained due to out-of-focus blur is not necessarily perfectly Gaussian-shaped
- *Constant PSFs:* Imperfections in cheap camera lenses result in varying PSFs for different image regions

[16] In case of Wittman, the algorithm required 6 minutes in MATLAB with a 2.4-GHz processor for a single bar code waveform.

- *No distortions:* Code geometry and perspective imperfections resulting from non-perfectly aligned mobile phones or non-straight bar code surfaces are usually not modeled or models are fairly limited

Regarding the recognition of blurry bar codes on mobile phones, Wang et al. [65, 66] proposed a well-designed algorithm based on a statistical method capable of coping with distorted and low-resolution images. Furthermore, codes are at first located in images using Wavelets. While not requiring the accurate extraction of the edge or peak locations of bar code bars, the proposed algorithm still relies on extreme points found in the bar code waveform during character segmentation, and is therefore not able to recognize bar codes in very blurry images. Furthermore, the proposed algorithm has been tested on images obtained from a real mobile phone (Nokia 3650), but no actual implementation of the recognition method on mobiles is presented, leaving the performance of the presented approach unclear. Ramtin Shams [66] also developed a method suitable for reading bar codes from low-resolution images with blurs and noises, even under non-uniform lighting. Similar to Wang et al., this method also relies on relatively sharp images and no actual implementation on mobiles is presented.

The work by Rocholl et al. [67] is the most relevant work here, as it also proposes an algorithm for the recognition of blurry bar codes in images based on the above-mentioned analytical approach, but includes several speed optimizations and has been implemented on a mobile device. Major differences between our solution and the work presented by Rocholl et al. include the following:

- In the proposed work, each digit is guessed individually based on the blurry bar code waveform, greatly reducing the required time for decoding the complete bar code number. However, this comes at the price of not being able to recognize images that are so blurry that the waveform of a single digit is severely influenced by the previous and following digit (e.g., as is the case in our example image shown in Figure 3.11). Furthermore, considering each digit separately is likely to increase the number of wrongly recognized bar code numbers (false-positives).
- No method for reliably reducing false-positives, except by means of the check digit and the determination of the first system digit based on the parity pattern is presented. This is likely to lead to very high false-positive rates and may result in the problem that when trying to

reduce the number of false-positives to a reasonable amount by adjusting threshold values, the number of correctly recognized bar codes is also drastically reduced. The presented brief evaluation contains only the number of correctly recognized codes, without mentioning the rate of false-positives.

- Despite being specifically targeted at mobile devices, the presented method requires around 20 times as long for the recognition of a single image on the iPhone compared to our solution. The presented algorithm requires nearly 2 seconds [68], while our method requires between 70 and 100ms for a single image on the same device.
- The proposed algorithm provides fewer features and is less flexible compared to our method. For example, it provides no orientation detection of bar codes, is not able to recognize different code symbologies or codes printed upside-down or on round surfaces. Due to its approach, it is likely to exhibit a poor recognition performance on sharp, but crumpled or otherwise distorted images of bar codes.

The work by Rocholl et al. seems to be promising in terms of the underlying potential of the approach. However, knowing further details regarding the problem of false-positives, the set of bar codes and conditions the recognition has been tested on, and the implementation on the iPhone itself would have been interesting. In order to be able to faithfully compare it to our recognition method, information about practical challenges, such as locating the start- and end-positions of bar codes in complex, blurry images and questions like how well their method is suited for different PSFs as caused by different mobile phone cameras and imperfect lenses, or non-perfectly Gaussian shaped PSFs would have been interesting too.

3.4.3 Commercial Solutions

Given the commercial interest in mobile services, a number of commercial solutions for recognizing bar codes has recently become available for the major mobile phone platforms iPhone [69], Android [70] and Nokia/Symbian [71]. Since Android comes with a bar code scanning engine built into the operating system, a larger variety of scanners are available on the iPhone. Only few solutions are available for Symbian devices, which might be due to the large variety of software platforms and different device types in the Symbian domain that complicate the recognition and software development process. In contrast to our recognition algorithm, most available solutions require sharp

images and therefore devices with autofocus cameras. However, three other scanners also support the recognition of codes in blurry images: RedLaser [72], ShopSavvy [73] and pic2shop [74].[17] Compared to these three solutions, our algorithm offers a higher recognition speed and accuracy as well as unique features such as the recognition of arbitrarily oriented bar codes. Chapter 5 provides a detailed analysis of the best available commercial solutions in terms of features, recognition speed and accuracy in comparison to our algorithm.

3.5 Summary

This chapter presented our algorithm for recognizing bar codes on mobile phones. The algorithm's architecture differs in several aspects from related work. It combines two decoder architectures with different strengths and weaknesses in order to address the various challenges in recognizing bar codes in realistic application scenarios. Furthermore, our approach is capable of recognizing codes in very blurry images and relies on pre-computed patterns in order to achieve a high recognition speed and robustness. Based on a set of parameters, the algorithm can be adjusted for a variety of conditions. Strengths of the presented algorithm include its fast execution speed and high recognition accuracy as well as its flexibility in terms of required system resources, supported image resolutions, and sharpness levels. A remaining problem for some practical scenarios is the limited recognition performance in the case of irregularly distorted bar codes in very blurry images, e.g., codes printed on heavily crumpled surfaces.

The next chapter of this thesis will discuss proof-of-concept implementations of this algorithm and provide relevant details to the parameter optimization process. Chapter 5 will then present an in-depth performance evaluation of these proof-of-concept implementations and compare them to related work.

[17] For commercial systems, there are no details available related to the underlying algorithms and recognition methods used.

4 Implementation

In this chapter we present proof-of-concept implementations of the recognition algorithm on three major mobile platforms and discuss relevant implementation-specific topics, such as device independence and performance optimizations (Section 4.1). Furthermore, we briefly present the relevant tools used to develop, as well as optimize, the bar code recognition process (Section 4.2), and conclude with several measurements done with the implementation that confirm the appropriateness of selected algorithm design decisions (4.3).

Figure 4.1 Implementations on the Nokia N95 8G (Symbian), iPhone 3GS (iOS) and HTC Desire (Android).

4.1 Recognition Engine

4.1.1 Proof-of-Concept Implementations

In order to confirm the practicability of the presented recognition method, we implemented proof-of-concept implementations of the recognition algorithm and scan GUI for three major mobile phone platforms: iOS, Android, as well as C++ Symbian. A typical application consists of three basic components presented in Figure 4.2. The mobile application itself, the implementation of the recognition algorithm including the recognition tables, and a third component that is responsible for accessing the camera images and implementing the recognition GUI. Figure 4.1 shows screenshots of three mobile applications using our bar code recognition.

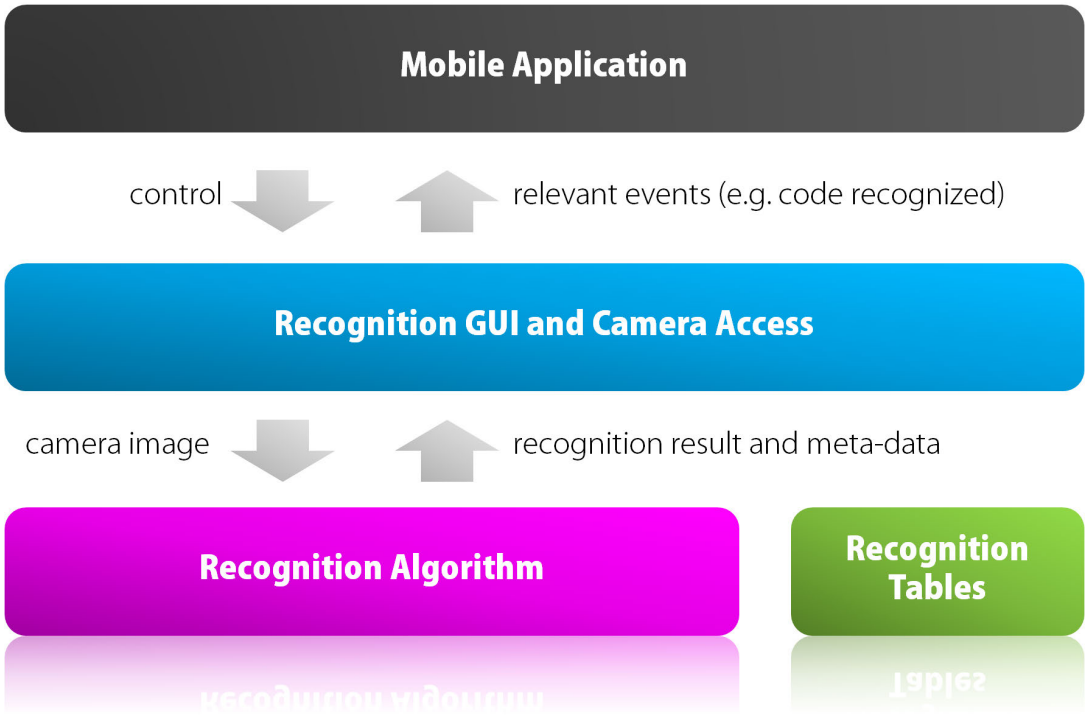


Figure 4.2 Components of a mobile application that uses the bar code recognition.

4.1.2 Multi-Platform Support

The specific mobile application and the component responsible for the camera access and graphical user interface are platform-specific and have to be implemented separately for each new mobile phone platform that should be supported. In contrast to this, the recognition algorithm itself is implemented in standard C++ and can therefore be directly used on all platforms that support the inclusion of native code, e.g., iOS, Android, C++Symbian/QT and Windows Mobile 7. In order to abstract from platform specifics in the algorithm, we rely on two concepts:

1. Configuration files are used to optimize the recognition algorithm for different device classes and abstract therefore from different optics or available resources like CPU, RAM, image resolution, and camera frame-rates on devices.
2. In order to abstract from the platform-specific implementations of images obtained from the camera, we use the concept of *generic images*.

A generic image is an interface defining basic operations for accessing image data that is used by the implementation of the recognition algorithm. The recognition algorithm relies only on these operations and is therefore inde-

pendent from the underlying image implementation. Before an implementation-specific image obtained from the camera is passed on to the recognition algorithm, it is converted to a generic image. However, the conversion of image orientation, format and color model to a standard format is not done beforehand for the whole image, but in a "lazy" manner on-the-fly, and only for the image areas and pixels that are required by the recognition algorithm.

4.1.3 Performance Optimizations

The above-mentioned lazy implementation of generic images speeds-up the recognition process considerably, especially in combination with our scan line-based recognition method, and the optimized orientation detection in low resolution sub-images of the original image, since only a subset of all image pixels have to be considered. Especially in the case of higher resolution images, this proves to be a large performance advantage compared to implementations requiring a full conversion of the whole image, and algorithms performing operations such as binarization on the entire image. Figure 4.3 shows the percent of pixels of an image with width w and height h that have to be analyzed by our algorithm when recognizing a bar code. We assumed parameters as they are used in the current implementation: 32 scan lines, the orientation detection with a window of size $w/2 \times h/4$ and the consideration of every n-th pixel. The value of n is decreased or increased with image resolution in order to obtain always a similarly-sized edge window for the orientation detection. Particularly in higher resolution images, like they become increasingly available on modern smartphones (e.g., 1280x960 pixels on the iPhone 4G), we have to consider only around 4% of the pixels in an image to successfully recognize a bar code. Nevertheless, our algorithm still benefits from higher image resolutions in terms of recognition performance due to the scan line-base decoders.

Figure 4.4 shows the time required by implementations on the three test devices for the recognition of a bar code in a single image, dependent on the image characteristics. The times listed do not vary as much as the different device specification (Table 4.1) might imply. This is due to the fact that the times measured are influenced by several factors:

- All devices feature a single core processor and the load of tasks running in parallel to the image recognition vary significantly. For example, on the Nokia N95 we update the screen each time we passed an image to the recognition algorithm, while on the HTC Desire or

iPhone, the higher resolution screens are updated asynchronously from the bar code recognition each time a new frame is available from the camera. The latter results in a higher frame rate but also much increased CPU load.
- Compiler specifics can have a major influence on performance, especially in the case of the blurry decoder, which involves many reoccurring operations and memory accesses.

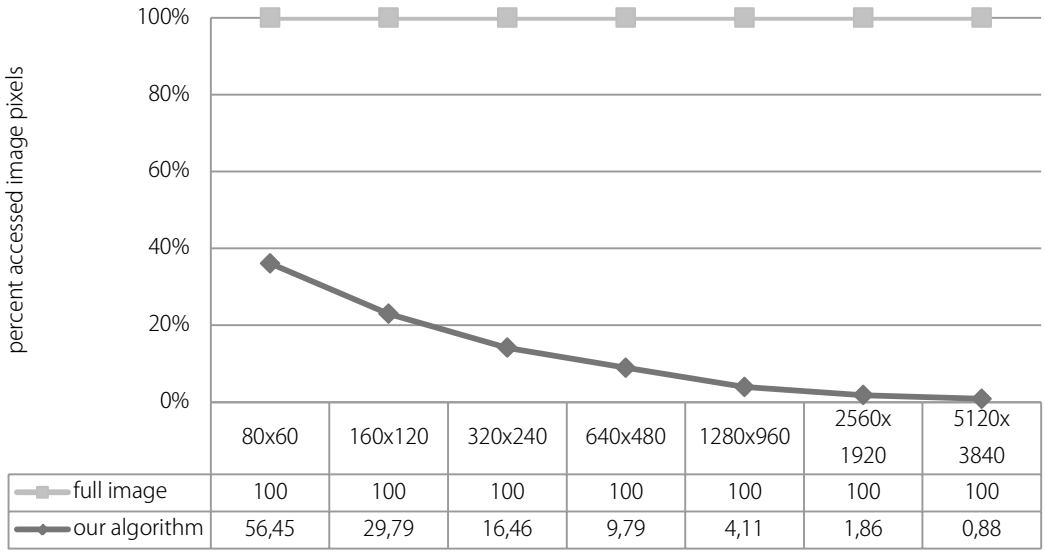
Implementation-specific measures taken to improve the speed of the recognition engine include the following:

- In the case of C++ Symbian, we use a highly optimized method for resizing the camera images to screen resolution before they are displayed.
- As many values as possible have been pre-computed. For example, look-up-tables for the calculation of sin, cos and $sqrt$ values. Dependent on the speed of the underlying CPU, we either directly compute these values at runtime or use lookup tables.
- Optimized utility methods for the resizing, comparing, and convoluting of patterns, or for the sorting of values are used. All operations are either performed in-situ or use pre-allocated memory in order to avoid often reoccurring memory allocations at runtime.
- All often reoccurring operations rely on integer operations instead of floating point calculations.

Besides speed, the runtime-memory consumption as well as overall size of the distributable recognition engine is important. In the case of the iPhone 3GS, the implementation with two recognition tables requires around 7MB runtime-memory. Figure 9.2 in the Appendix provides further details regarding what algorithm parts require how much memory. Figure 4.5 shows the same for the space requirements of the whole distributable. The size of the distributable is limited by using only few tables and compressing table files on the mobile devices.

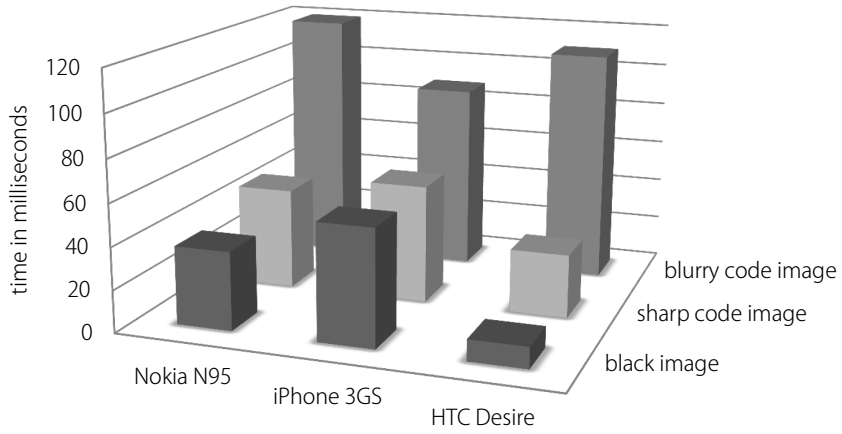
Table 4.1 Relevant specifications of test devices:

Phone Model	CPU	Memory	Recognition Resolution	Screen Resolution
Nokia N95 8g (Symbian)	332 MHz ARM11	128MB	320x240 up-scaled to 640x480	240x320
iPhone 3GS (iOS)	600 MHz ARM A8	256MB	320x436	320x480
HTC Desire (Android)	1GHz Snapdragon	576MB	480x640	480x800



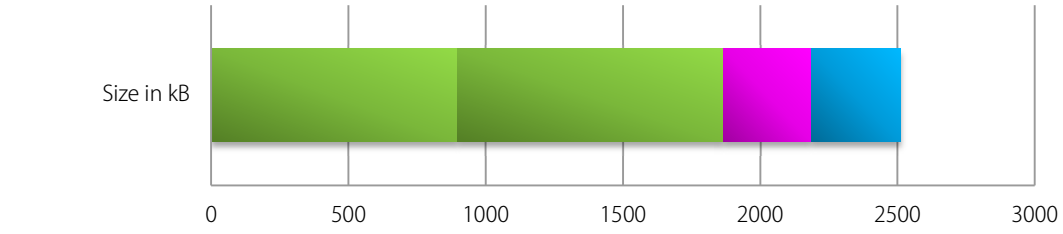
	80x60	160x120	320x240	640x480	1280x960	2560x1920	5120x3840
full image	100	100	100	100	100	100	100
our algorithm	56,45	29,79	16,46	9,79	4,11	1,86	0,88

Figure 4.3 Percentage of image pixels that have to be observed during the recognition process as a function of image resolution.



	Nokia N95	iPhone 3GS	HTC Desire
black image	37	55	10
sharp code image	48	56	30
blurry code image	119,5	88,5	110

Figure 4.4 Average times in ms for processing a single camera frame on the three devices. Processing includes: Obtaining the image from the camera, recognizing the code, displaying the image on the screen and drawing the GUI.



	Size in kB
Table 1	896,85
Table 2	969,306
Recognition Code	318,97
GUI	325,95

Figure 4.5 Size of the recognition engine distributable in the case of the iPhone 3GS.

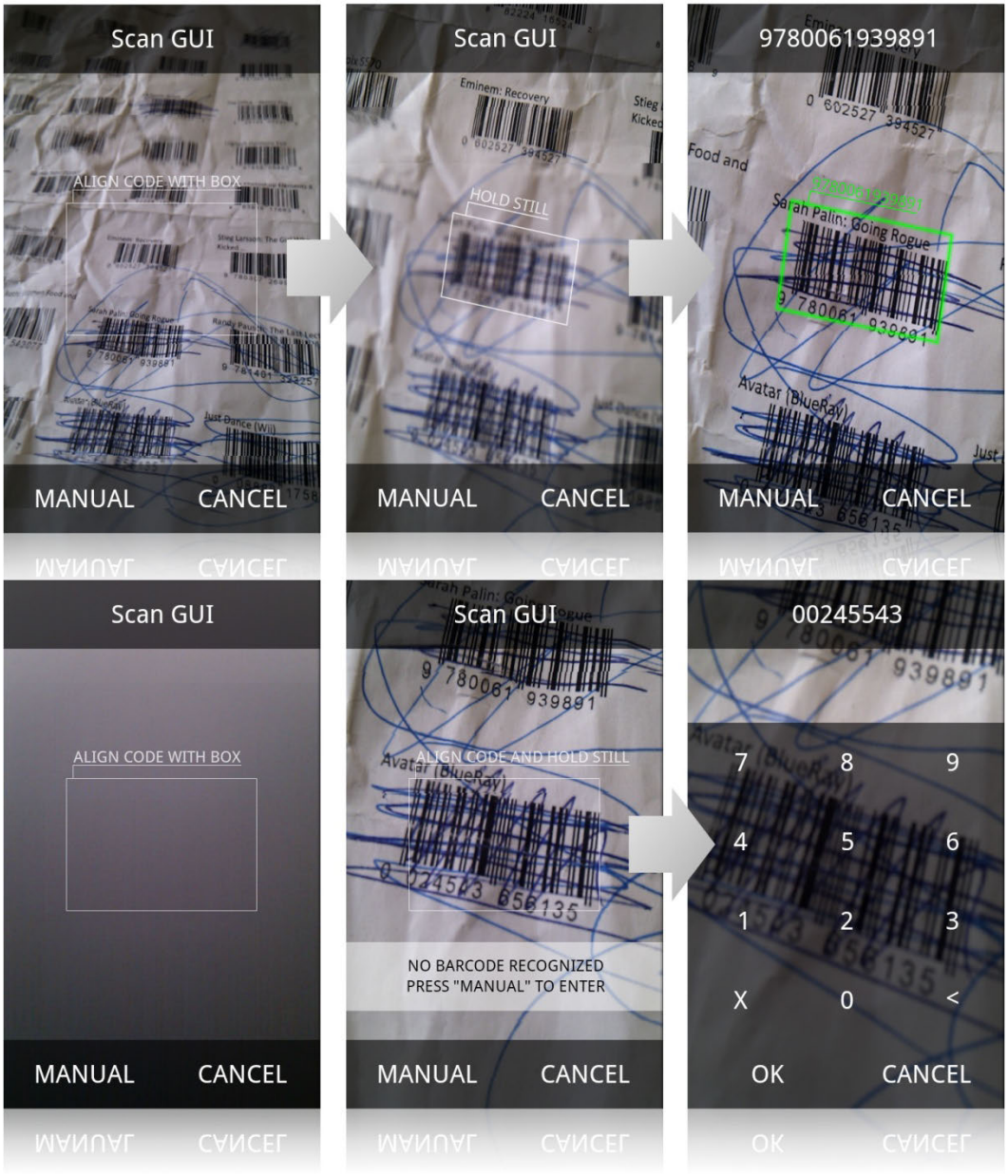


Figure 4.6 Screenshots from the recognition GUI on the HTC Desire.

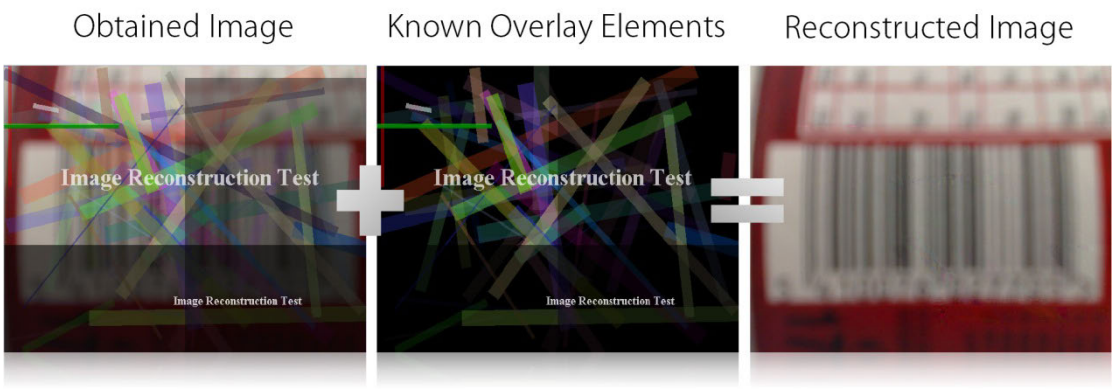


Figure 4.7 Example of the image reconstruction based on known overlay data.

4.1.4 GUI

Figure 4.6 shows the graphical user interface (GUI) for scanning bar codes we implemented on Android. It features a rectangle that indicates the rough size and position a bar code should have on the screen for an optimal recognition. However, rotated codes or codes that are smaller or larger than the shown rectangle can also be recognized. If a bar code is detected in the image, the displayed message changes to "HOLD STILL", and the code's position is marked. In the case the code has been recognized, the screen flashes, a sound is played and the color of the marker around the code changes to green. If a code that is visible in the images cannot be recognized for some time, it is hinted to the user that he or she should consider entering the bar code manually. Manually entering the code is supported with the help of a soft keyboard.

4.1.4.1 Device-Specific Challenges

Practical challenges included the fact that the application orientation on Android devices is always "landscape" if the camera images are visible. This prevented the use of already available GUI elements such as soft keyboards and required us implement all elements from scratch and draw them 90 degrees rotated. On iOS up to version 4, no direct access to real-time images from the camera was provided. Instead, images had to be accessed indirectly by opening the camera perspective and taking screenshots from the device's screen. However, this resulted in the limitation that all GUI elements drawn on the screen were contained in the next screenshot and therefore in the image used for the bar code recognition. We therefore added the possibility to our recognition engine to compensate for drawn overlays. Given an image and information about the drawn overlay elements, the drawn elements can be removed from the original image if they have not been drawn opaque. Figure 4.7 shows an example.

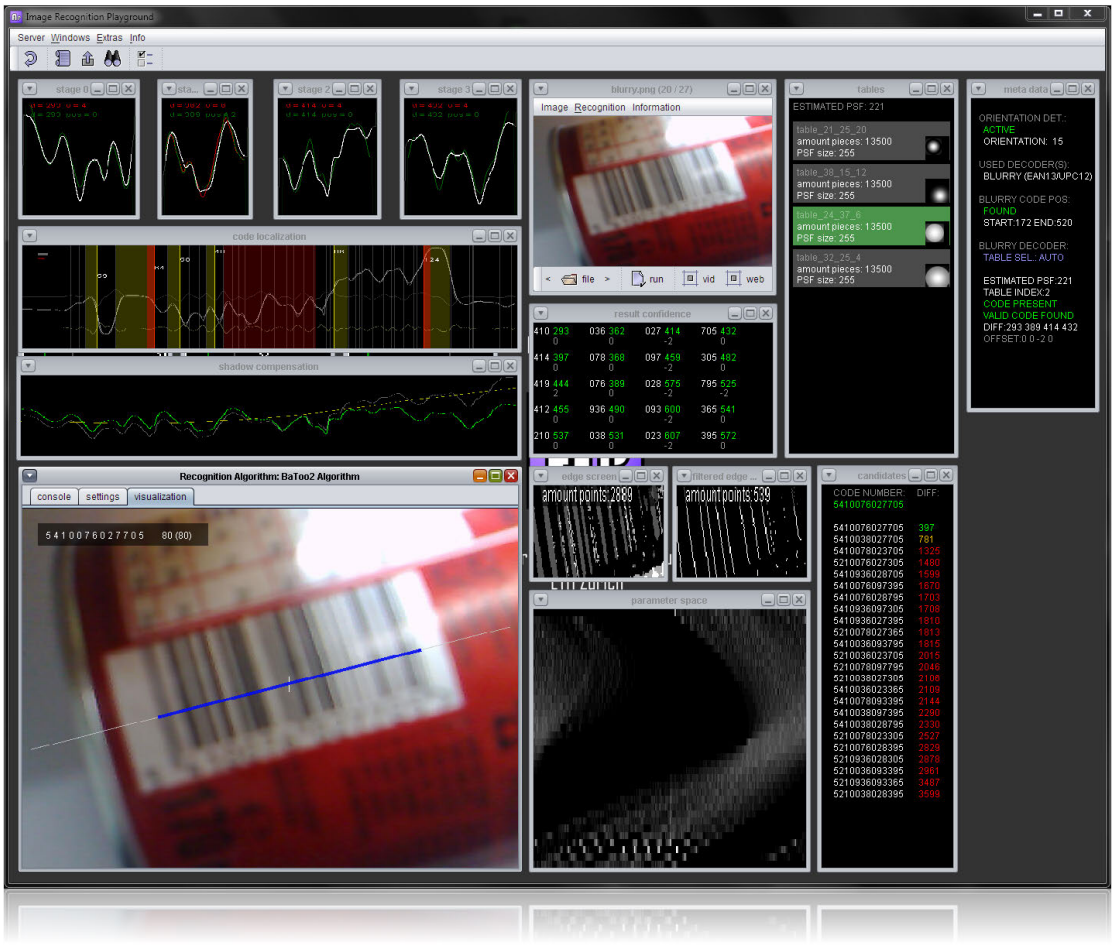


Figure 4.8 Screenshot of our algorithm test and visualization environment.

4.2 Development Tools

4.2.1 Algorithm Test Environment

Developing the bar code recognition algorithm was complicated by two facts: The variety of possible bar code images, and the fact that the comparison patterns-based approach for the blurry bar code recognition is non-transparent, meaning that even minor implementation errors or effects like the rounding of numbers can have strong effects on the result and are hard to detect. The variety of possible codes and images might result in the recognition to work on a test image but to fail on another, only slightly different image. In order to address these challenges, we developed a visualization and test environment for image recognition algorithms in Java. Selected aspects of the algorithm have first been explored using Matlab [75], but the final algorithm has been developed and optimized in Java with the help of our test environment. Afterwards, the Java code has been ported to C++ for efficiency and portability

reasons. Figure 4.8 shows a screenshot of the Java test environment. It provides the following features:

- The environment allows for the real-time visualization of the state of relevant algorithm components, and thus supports the search for bugs as well as the prototyping and comparison of different solutions to a given problem.
- Images to test the algorithm can be loaded directly from disk, or can be streamed live from a webcam, a web resource or directly from mobile phones using Bluetooth or WLAN. We implemented according client applications for Android and C++ Symbian devices; the latter using the SPARK environment presented in chapter 7.
- Automated tests can be performed in order to observe the effects of changes to the recognition algorithm or parameters. This can not only be done on single images, but on a large number of representative test images.

4.2.2 Recognition Table Creation

Recognition tables contain pre-calculated patterns of bar code sections and are used in the blurry decoder presented in Section 3.3. Tables are generated using a Matlab script. Parameters for this script describe the table that should be generated and include the set of code numbers that should be recognizable with the generated table (either all numbers or a known subset of numbers), parameters describing the PSFs for the different bar code sections, the size in which the patterns should be pre-calculated, as well as the size of the pattern borders that should be ignored during comparison. (See Section 3.3 for an explanation the meaning of the different parameters.)

4.2.2.1 PSF Parameterization

The point spread functions used to pre-calculate recognition tables can be specified by three parameters: $size$, specifying the size of the PSF in pixels, as well as σ and τ describing the shape of the PSF. Figure 4.9 shows how these parameters are used to construct a PSF based on a disk- and Gauss-shaped function, and Figure 4.10 illustrates how the PSF shape changes in relation to different values of the two parameters σ and τ.

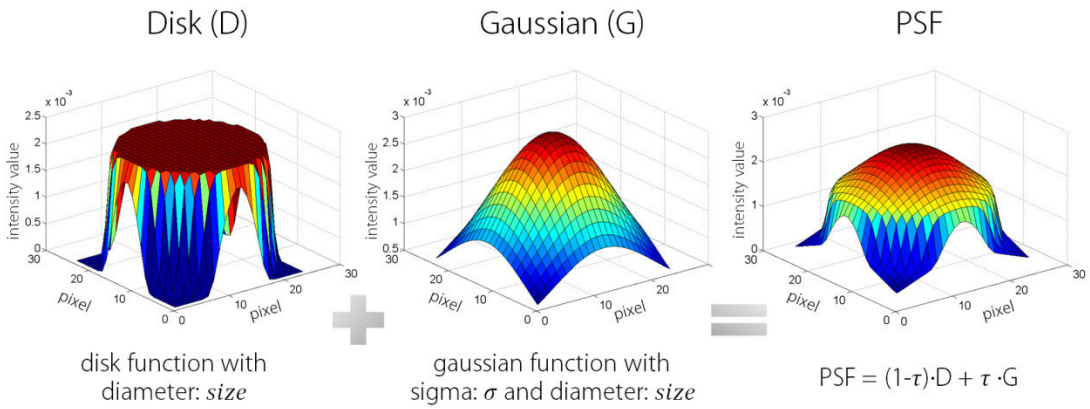


Figure 4.9 PSF construction based on the three parameters size, σ and τ.

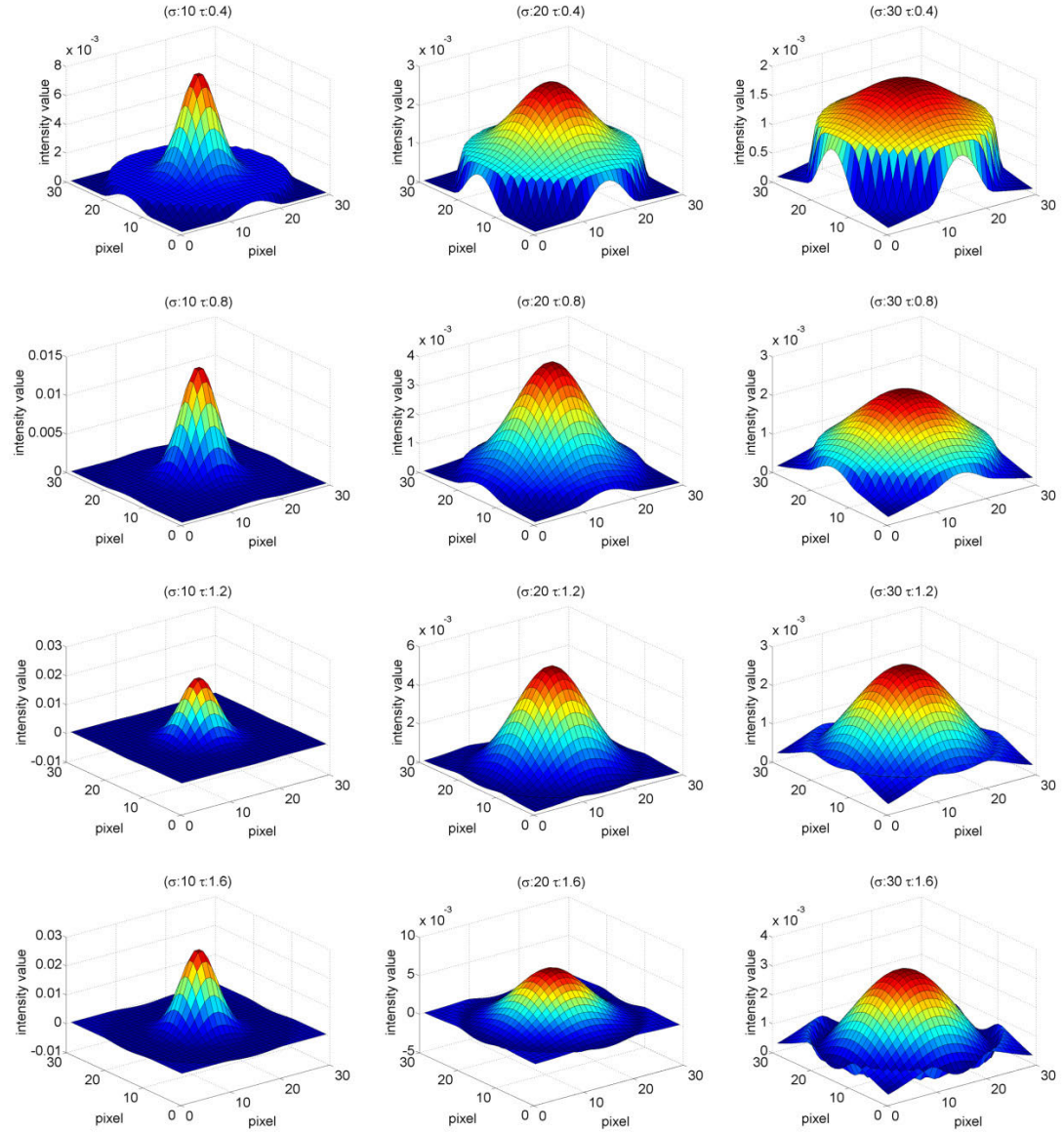


Figure 4.10 PSF shapes of *size* 30 as a function of the parameters σ and τ.

4.2.2.2 PSF Measurement

Since measuring the exact PSF shape and size for a given mobile phone model without specialized equipment is complex, we determine the PSF parameters that correspond to a specific mobile phone model and physical bar code size indirectly with the test-pattern I_{test} shown in the upper-left image in Figure 4.11. Given a specific mobile phone, we take images of this pattern in three different sizes. In each of these images $I_1 \ldots I_3$, the position of the four prominent corner blocks is searched using a 2D normalized cross-correlation. The lower-left image in Figure 4.11 shows the result of the normalized cross-correlation between the test image and a block pattern. A subsequent search for the four most prominent local maxima reveals the exact position of the four corner blocks. Based on this information, we de-skew the original image I_x to match the test-pattern I_{test} in position and scale. Afterwards, the image brightness values along a horizontal scan line are extracted at three different positions in the image I_x – at the left side $f_{left}(x)$, in the middle $f_{middle}(x)$, and the right side $f_{right}(x)$. Due to the preceding de-skewing process, the extracted blurry waveforms $f_{left}(x)$, $f_{middle}(x)$, and $f_{right}(x)$ are aligned with the sharp waveforms $f_{test-left}(x)$, $f_{test-middle}(x)$, and $f_{test-right}(x)$ extracted at the same positions from the sharp test image I_{test}. For each of these three pairs of a sharp and blurry waveform, the sharp waveform is convoluted with different PSFs and the pattern difference d between this blurred waveform and the original blurry waveform is measured using the normalized cross-correlation. We then take the PSF parameters that resulted in the smallest difference value d. The PSF parameters are measured at three different image positions, since image sharpness between these positions can vary significantly on some phone models, e.g., due to lens distortions. Figure 4.12 shows the original waveform $f_{middle}(x)$ and the constructed blurry waveform based on $f_{test-middle}(x)$ using the best found PSF. It can be seen that the constructed blurry waveform matches the original blurry waveform fairly well, which indicates that our model for the shape of PSFs is sufficiently precise.

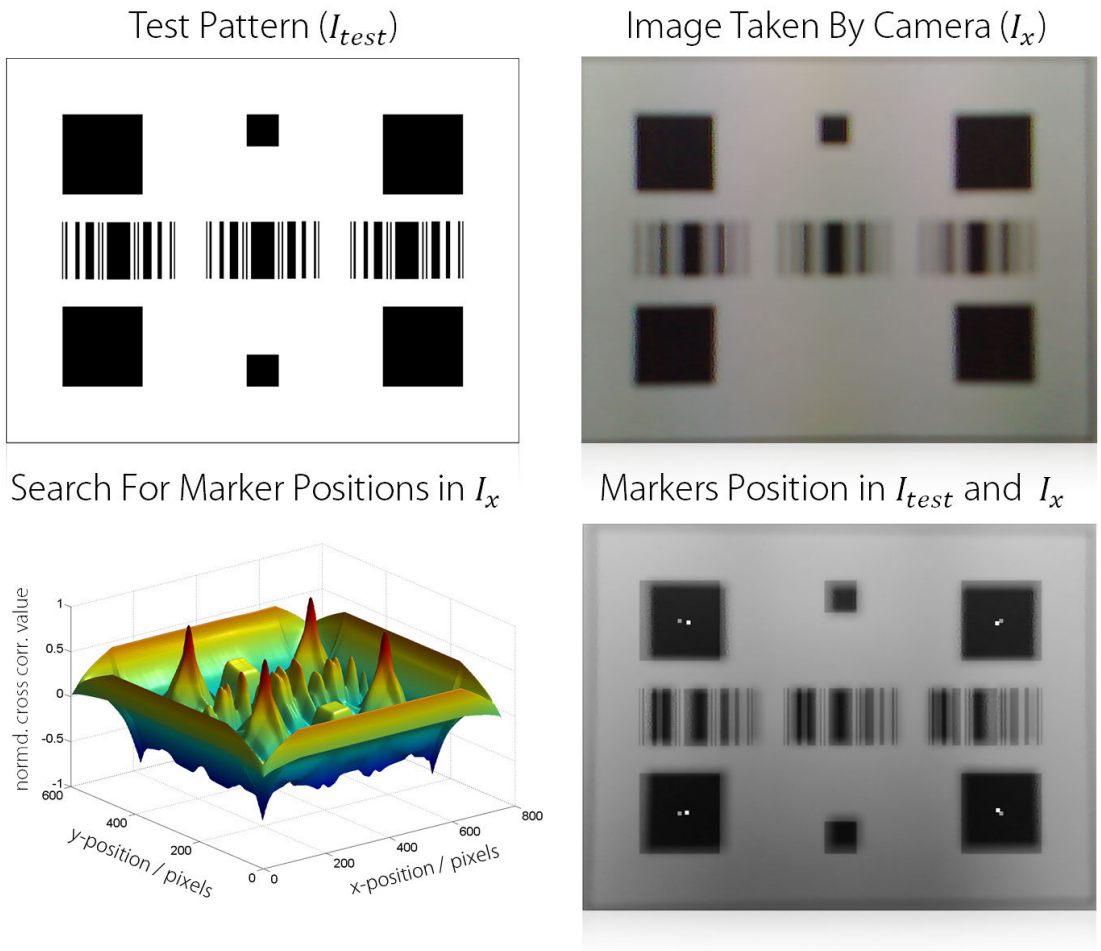


Figure 4.11 Test pattern used for indirect PSF measurement (upper-left image) and image obtained from Nokia N95 (without using the AF) of this pattern (upper-right image). Result from the normalized cross-correlation when searching for block patterns (lower-left image) and detected marker positions in both images used to de-skew the taken test image (lower-right image).

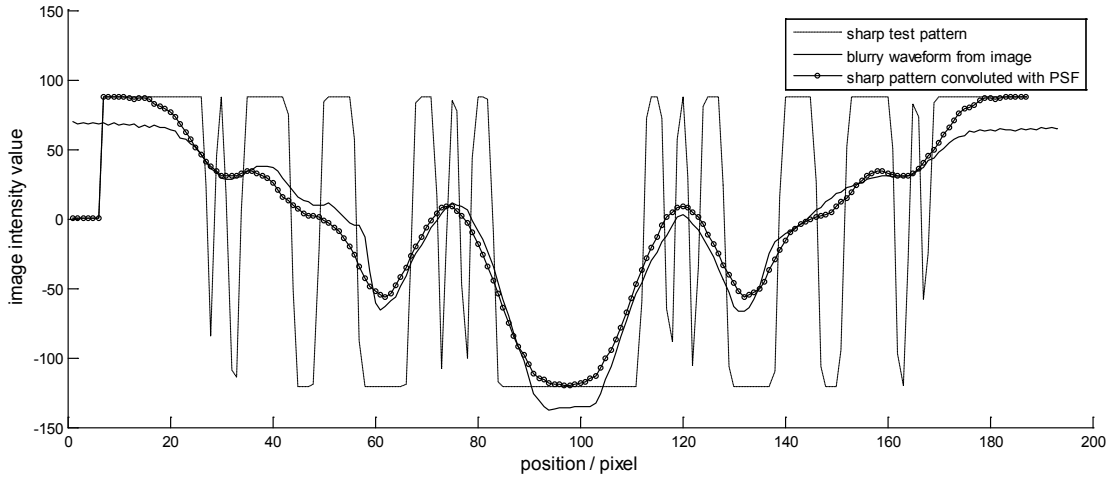


Figure 4.12 Diagram showing the sharp pattern waveform, the blurry waveform obtained from the test image and the waveform that has been constructed by convoluting the sharp pattern with the best found PSF.

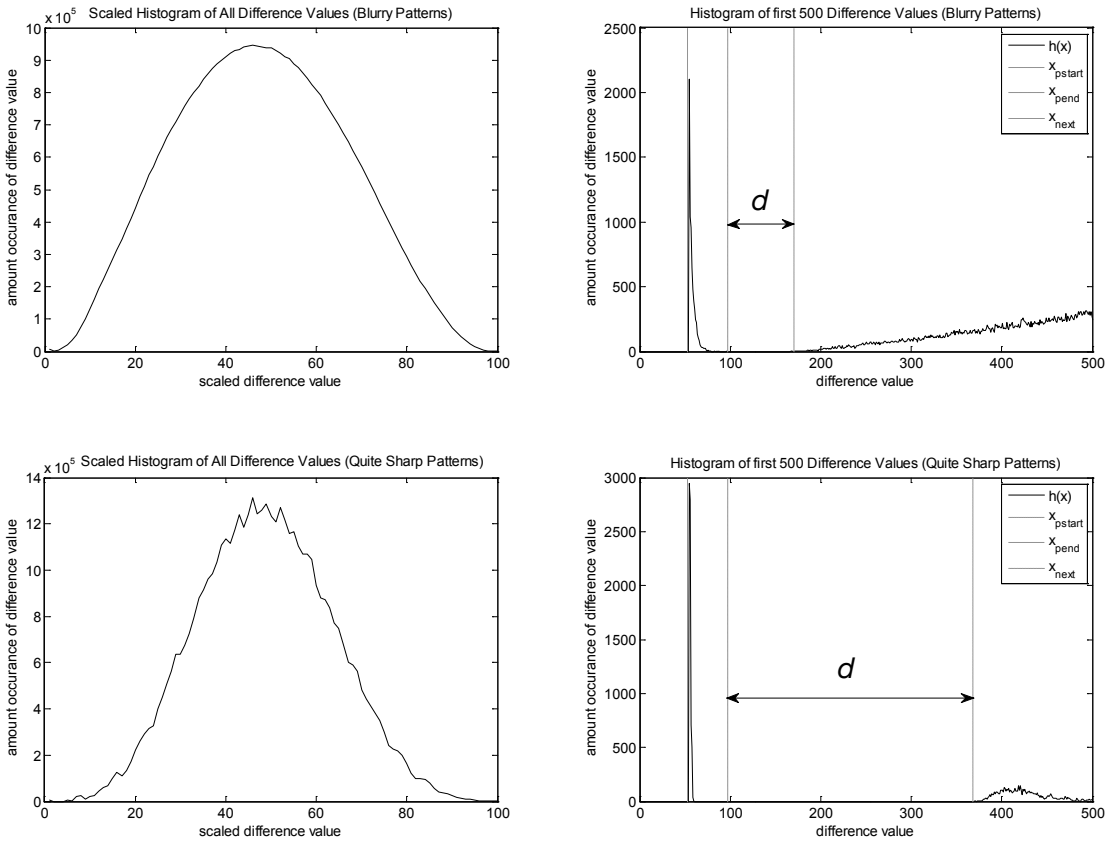


Figure 4.13 Histogram of difference values when comparing each of the 7000 pre-calculated patterns in section 2 to each other in the case of blurry and sharp patterns (left diagrams). Corresponding histograms limited to the first 500 entries (right diagrams).

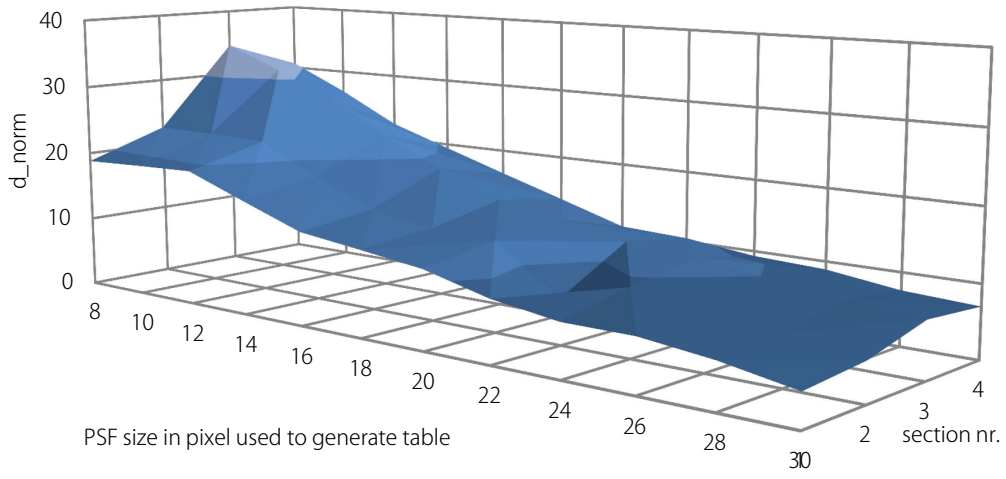


Figure 4.14 Distinguishability of pre-calculated patters as a function of table sharpness and the observed bar code section.

4.2.2.3 Similarity of Pre-Calculated Patterns

In the case of blurry pre-calculated patterns, patterns in the same section differ only in details from each other. However, all patterns for a bar code section remain distinguishable with the pattern comparison algorithm presented in Section 3.3.6. The most critical bar code section in this regard is section 2, since it contains the most (7000) pre-calculated patterns. Figure 4.13 show the histogram $h(x)$ of difference values obtained, when comparing each pattern in section 2 to each other with comparison algorithm 3 (Section 3.3.6.7). Several observations can be made and are described in the following:

The peak in $h(x)$ between the marked positions x_{pstart} and x_{pend} results from comparing each of the 7000 pattern with itself. Due to implementation specifics ,e.g., lookup tables for calculating the sqrt-values, comparing the same pattern does not necessarily result in a value of zero. It holds that:

$$\sum_{i=x_{pstart}}^{x_{pend}} h(i) = 7000$$

The first position $x_{next} > x_{pend}$ with $h(x_{next}) > 0$ corresponds to the minimal found difference value between two non-equal patterns in the observed bar code section. We can therefore use the distance $d = x_{next} - x_{pend}$ as a measure how distinguishable the pre-calculated patterns are. In case of $d = 0$, it is not possible to distinguish all patterns from each other. The larger d is, the better distinguishable the patterns are.

Figure 4.14 shows the values of $d_{norm} = d/(max\ difference\ value)$ as a function of the sharpness of patterns contained in a recognition table and the observed bar code section. It can be seen that the distinguishability of patterns (the value of d_{norm}) increases with pattern sharpness[18], and that patterns in sections 3 and 4, which contain only 1000 different pre-calculated patterns, are better distinguishable. In relation to the inner sections 2 and 3, the distinguishability of patterns in sections 1 and 4 that are located at the bar code borders is slightly lower, despite them containing the same number of, or even less, patterns. The reason for this is the fact that the pre-calculated patterns of the outer sections 1 and 4 include the bar code start- and end-delimiters, as well as silent areas, which are very similar for all patterns.

[18] A smaller size of the PSF corresponds to sharper patterns given the other two parameters controlling the PSF's shape remain the same.

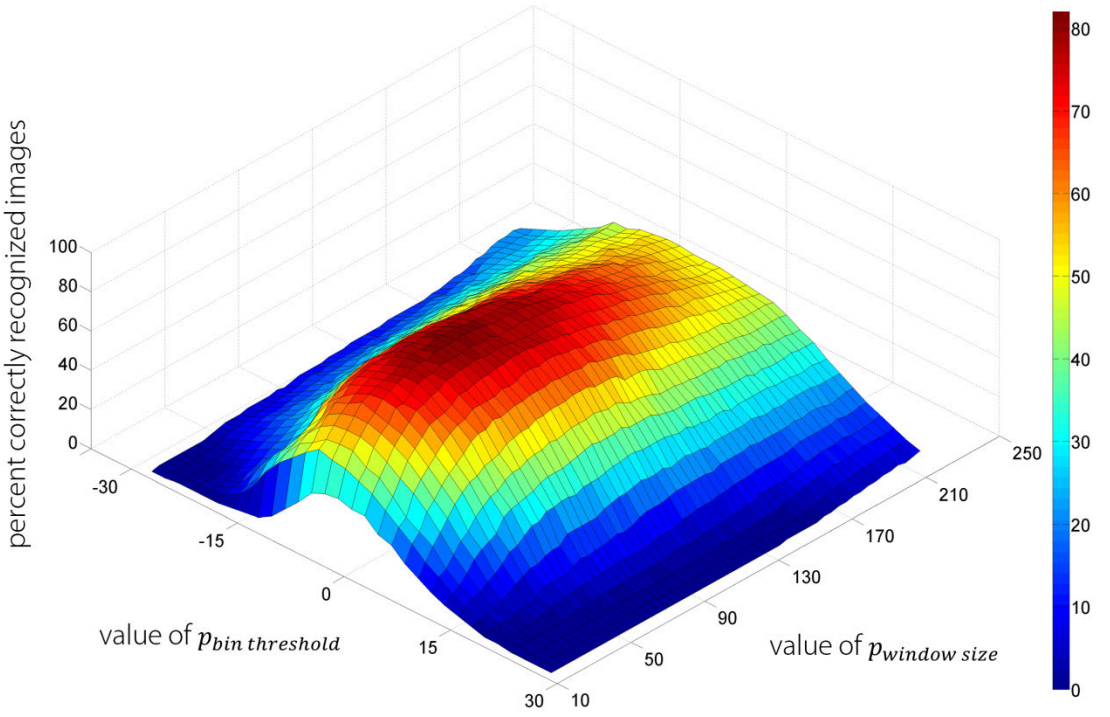


Figure 4.15 Percentage of correctly recognized codes by the sharp decoder from a test set of difficult-to-recognize sharp images as a function of the two main parameters controlling the scan line binarization process.

4.2.3 Device-Specific Optimizations

4.2.3.1 Parameter Optimizations

Like already stated in Chapter 3, a set of parameters stored in configuration files controls the recognition algorithm. The optimal parameter values for a specific mobile phone model are determined based on a set of test images of bar codes taken with this mobile phone. We generated sets of test images for important mobile phone types and more generic sets of test images that contain images obtained by common autofocus-enabled devices. A typical test set comprises around 1500 images of codes of varying physical size and in the case of autofocus devices different sharpness levels. For each test image there exists also a data file that contains meta-data about the test image, e.g., the correct number of the bar code contained in the image. The first step taken in order to optimize a set of recognition parameters $p_1 \ldots p_n$ for a test set of images $I_0 \ldots I_m$, is to construct all possible value combinations $\vec{v}_1 \ldots \vec{v}_r$ for the recognition parameters:

$$\vec{v}_x = \{v_1, \ldots v_n\},\ v_y = valid\ value\ of\ parameter\ p_y\ \ \forall 0 \leq x \leq r, 0 \leq y \leq n$$

Afterwards, the bar code recognition is performed r times on each image, each time with a different set of values $\vec{v}_x$ for the parameters $p_1 \dots p_n$. For every image and value combination, the recognition result, i.e., correct code recognized, no code recognized, or wrong code recognized[19] is recorded. Based on the aggregated results, we can extract the set of parameter values $\vec{v}_{optimized}$ that performs best according to selected optimization criteria. Examples for optimization criteria include the following, or combinations of the following criteria:

- Most images correctly recognized
- Least images wrongly recognized
- In all sets of images that contain the same bar code number at least one image should be recognized correctly

Figure 4.15 shows an example for the optimization of the two main parameters $p_{bin\ threshold}$ and $p_{window\ size}$ that control the binarization of waveforms in the sharp decoder (Section 3.2.1). The diagram shows the percent of correctly recognized sharp test images in the case of different combinations of these two parameters. Measurements are based on a test-set of 608 normal lit test images. It can be seen that for the analyzed test set of images values of $p_{bin\ threshold} = 0$ and $p_{window\ size} = 90$ result in the most correctly recognized images. The optimal values can be different in the case of dark or unevenly lit images, or images that exhibit prominent sharpening or compression artifacts.

Since the computational effort of the described method grows exponentially with the number of parameters that should be optimized, the optimization is limited to sets of parameters that comprise only few parameters, and parameters that are largely independent from each other. Furthermore, our implementation has been parallelized and optimized for speed. It can therefore be performed concurrently on an arbitrary number of threads. By switching from the currently used exhaustive search strategy in the space of all possible parameter combinations to more guided strategies using heuristics, further improvements are possible.

[19] The meta-data available for each image file contains the correct code number, which allows us to detect if a code has been recognized correctly or wrongly.

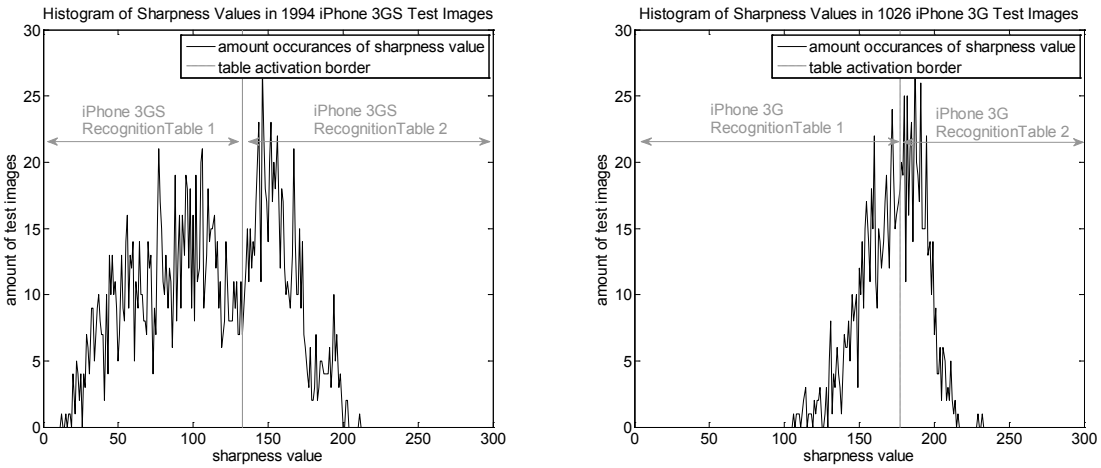


Figure 4.16 Sharpness distribution of test images taken on a device with (iPhone 3GS) and without (iPhone 3G) autofocus camera. The sharpness range covered by pre-calculated recognition tables is also shown.

4.2.3.2 Table Set Optimizations

In Section 4.2.2 we described how the optimal PSF parameters for a given test image can be determined. However, when trying to determine the optimal set of recognition tables that results in the most correctly recognized images from a given test set of images, three questions remain:

1. How many tables are sufficient to obtain a good recognition rate on the whole test set of images?
2. What test images should be selected to calculate the PSF parameters for the recognition tables to generate?
3. Which of the pre-calculated recognition tables should be used at runtime in what sharpness range? Answering this question is further complicated by the fact that the measured sharpness values also vary slightly dependent on other image features than the waveform's sharpness. (See Section 3.3.2 for details.)

In order to optimize the recognition for a given set of test images $I_0 \dots I_m$ we therefore used the following approach to determine the optimal number of tables, PSF parameters, as well as activation sharpness-ranges for these tables. In a first step, we constructed 27500 recognition table files, each one created with slightly different PSF parameters. This is done by varying the three parameters $size$, σ and τ that control the shape of PSFs at different bar code sections. Next, we performed the recognition once for each combination of a test image I_x and a recognition table T_y. In the case of 1500 test images,

this results in $41250*10^3$ recognition runs. For each run, we recorded the following values[20]:

1. The outcome of the recognition, i.e., a correct code is recognized, no code is recognized or a wrong code is recognized
2. The quality factor indicating how well a detected code fits
3. The sharpness value measured by the blurry decoder along the central scan line

The values obtained when recognizing all test images with a single recognition table are stored in a *protocol file* for this table. Appendix 9.1.2 shows an example of a protocol file.[21] Based on the information contained in the resulting 27.500 protocol files, we search for an optimal result according to a set of optimization criteria, similar to the procedure used for parameter optimization. For a given test set of images, the obtained result comprises the following information:

1. The minimum number of tables required to obtain a good recognition rate on the test set of images
2. The optimal PSF parameters for each recognition table
3. The range of sharpness values in which each recognition table should be used

Figure 4.16 shows the measured sharpness values of images in a test set from the iPhone 3GS that features an autofocus camera (left diagram) and the iPhone 3G that has only a fixed focus camera (right diagram). It can be seen that the sharpness values of images obtained on the iPhone 3GS contain more sharp images (lower sharpness values) than the test images on the iPhone 3G, which are all blurry. Test images on both devices have been taken from bar codes on real products of different physical size, ranging from small to large bar codes. In order to limit the size of the final recognition engine distributable, we used only two recognition tables on each device. Figure 4.16 shows the sharpness ranges, in which the two recognition table are used. The recog-

[20] Similar to the parameter optimization, all relevant processes for the table set optimization have been parallelized. On a desktop PC with 16 CPU cores and 1500 test images, performing all recognitions takes on average 72 hours.

[21] In order to reduce the space requirements as well as speed-up the loading of the 27500 protocol files into memory, protocol files are packed in ZIP archives.

nition performance of the blurry decoder can be improved by using more recognition tables. For example, on the iPhone 3GS, most images with sharpness values between 0 and 133 can be correctly recognized using recognition table 1. However, in the cases where the sharpness of the image and precalculated patterns in the table differ too much, the final step in the blurry decoder that checks for false-positives might reject a correctly recognized code number, since the confidence in the result is too low (see sections 3.3.7 and 3.3.8 for details). On the iPhone 3GS, this is in practice not a big problem, since many sharp images (e.g., images with sharpness values below 100) can be recognized by the sharp decoder. The implementations of our bar code recognition engine used in the user study (see Chapter 5) all use two recognition tables.

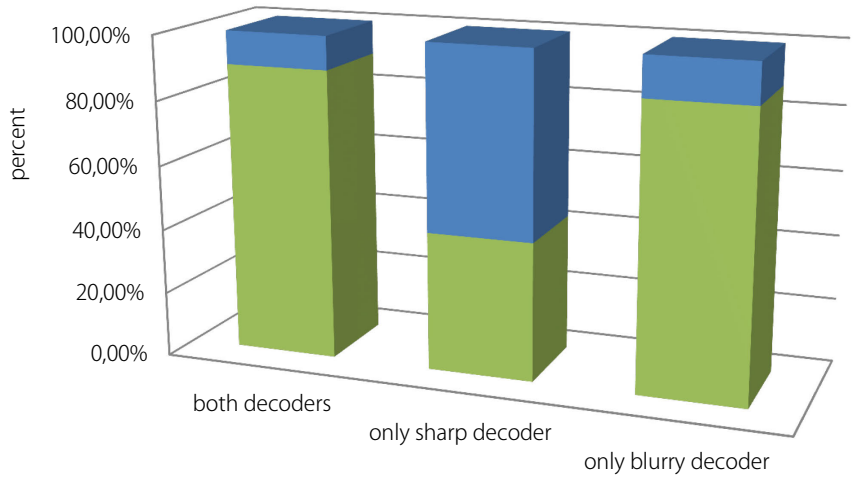
4.3 Measurements

4.3.1 Performance of Decoder Types

Like already stated in the recognition algorithm, we combine two approaches to recognize bar codes: the sharp decoder and the blurry decoder. Figure 4.17 shows how many out of 1994 test images taken with the iPhone 3GS can be recognized when limiting the recognition to either the blurry or the sharp decoder. The sharpness distribution of used images can be seen in the left diagram in Figure 4.16. Even when both decoders are used, not 100% of all test images are recognized. This has the following reasons:

1. Images include difficult to recognize codes, e.g., very small codes
2. Only two recognition tables are used for the blurry decoder
3. The recognition algorithm is used in a "single image"-mode, which results in the situation no information can be used that is obtained over several video-frames as on the mobile devices

It can be seen, that on this test set of images and two recognition tables, the blurry decoder can correctly recognize the majority of images (87.63%). Like expected, the sharp decoder is limited to sharp images and can recognize therefore only 42.78% of all images. An additional challenge for the sharp decoder on this test set is the low resolution of test images of 320x436 pixels. This results in scan lines of 320 pixels length, in which the whole bar code pattern was on average only180 pixels wide. The latter complicates the binarization process, especially in slightly blurry images.



	both decoders	only sharp decoder	only blurry decoder
not recognized	10,31%	57,22%	12,37%
wrongly recognized	0,00%	0,00%	0,00%
correctly recognized	89,69%	42,78%	87,63%

Figure 4.17 Recognition result on a test set of 1994 images of varying sharpness taken with the iPhone 3GS. Two recognition tables are used for the blurry decoder.

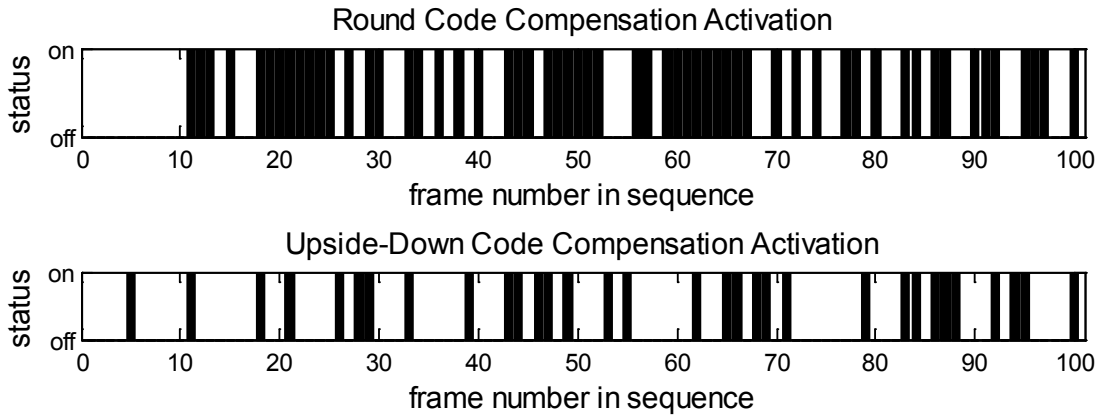


Figure 4.18 Measured activations of the round and upside-down code distortion compensation in a series of 100 video frames. The video images showed the bar code visible in Figure 3.20 that is upright and printed on a round surface.

4.3.2 Distortion Detection

Detecting bar codes on round surfaces or codes that are oriented upside-down in images is difficult, especially in the case of blurry images. Due to this, Section 3.3.5 introduced a mechanism that activates the distortion compensation for bar codes on round surfaces or upside-down oriented codes in dependence of probability values. These probability values indicate how likely it is that a distortion is present in the observed video images and are adjusted

over time, based on the meta-data collected by the recognition engine. Figure 4.18 depicts measurements from our implementation that show in which frames in a series of 100 images the compensation for round and upside-down codes has been activated, for an upright code on a curved surface. The results indicate that our indirect detection of distortions in video images works in practice as expected.

4.3.3 Influence of Image Resolution on Recognition Rates

The resolution of the video images available for the recognition on different mobile operating systems and devices can vary drastically. We therefore examined how image resolution affects recognition rates. In order to do this, we used a test set of 322 test images. The test set included codes of varying difficulty, e.g., rotated and upside-down oriented codes, as well as codes on round and crumpled surfaces. All images have been taken with the iPhone 4G, have a native resolution of 720x1280 pixels and are either sharp or only slightly blurry. We performed the recognition on the images in this test set and varied both the image resolution and the decoders used for the recognition. Figure 4.20 shows the results. When interpreting the results, it has to be considered that bar codes contained in the images covered in general only 50-60% of the image's width, meaning that if codes could be recognized in an image of 360x640 pixels, many of these codes had a length of only180 pixels. The following can be observed:

1. Above an image width of 180 pixels, the recognition result is in large parts independent of the underlying image resolution.
2. The recognition performance of the sharp decoder decreases rapidly for images with a width of 270 pixels and less. Since the sharp decoder relies on detecting and distinguishing different widths of the code's bars, this is expected.
3. The blurry decoder is able to recognize images even in very low resolution images. Well printed codes on flat surfaces can be correctly recognized in images with widths as low as 100 pixels; a few codes even in the 68 pixel wide images. This corresponds to bar code patterns consisting of only 50 or less pixels.

Figure 4.19 The same test image in three different resolutions and the corresponding details of the contained bar codes.

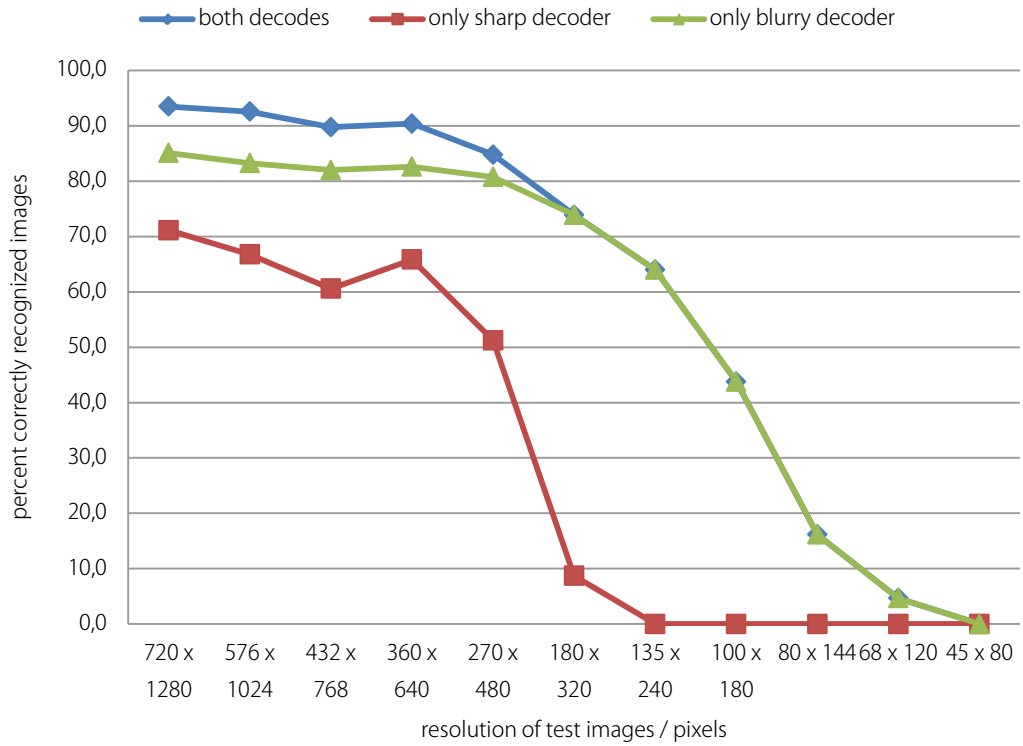


Figure 4.20 Percentage of correctly recognized test images as a function of image resolution. Results are shown for the standard recognition algorithm that uses both decoders, as well as situations in which we limit the recognition to the sharp decoder, and the blurry decoder.

4.4 Summary

In this chapter, we verified the practicability of the bar code recognition algorithm described in Chapter 3 by providing proof-of-concept implementations on three different mobile phone platforms. We presented relevant details regarding the pre-calculation of recognition tables and tools used to develop and optimize our recognition method. The presented results regarding the speed and memory consumption of implementations and the influence of image resolution on recognition rates show the intended flexibility of our recognition method. This supports its applicability for a wide range of mobile devices. The following Chapter 5 provides an in-depth evaluation of the scan accuracy and speed of our implementation, and compares it to the most relevant other bar code scanners available.

5 Evaluation of Bar Code Scanners

In this chapter, we provide an in-depth evaluation of existing bar code recognition engines. Given the proliferation of smartphones, cheap data rates, and mobile applications over the last years, the number of available mobile services has dramatically increased. This is particularly true with platforms like the iPhone (and, increasingly, Android) and in shopping-related application areas such as product-price comparisons. Due to this increased interest in mobile services, a series of bar code scanning engines has emerged in the last 2 to 3 years. The more prominent of these applications are RedLaser or the Google Scanner [76] built into the Android OS. While a user study conducted by Reischach et al. in 2009 revealed poor recognition performance of most scanners [77], many scanners have evolved and improved since then.

In order to evaluate recognition engines, we conducted a user study with 16 participants to evaluate "real-word" performance (Section 5.1) and a detailed analysis under controlled conditions to determine the features and characteristics of the best scanners available (Section 5.2). Based on the quantitative and qualitative results of the user study and scanner analysis, we came up with general observations about users' scanning behavior and the capabilities and limitations of bar code scanners (Section 5.3.1). Finally, we use this information to state user interface guidelines for mobile phone-based bar code scanners (Section 5.3.2).

With respect to the comparison of our proposed recognition engine with other solutions, results show that our recognition engine outperforms other solutions in terms of scan speed and accuracy on all tested mobile devices.

5.1 User Study

This study evaluates the usefulness of existing bar code scanners for consumer-oriented mobile services under realistic conditions. We compare the five best-performing solutions from our scanner analysis (Section 5.2) on three mobile phones in a realistic set-up with 16 participants. We perform a quantitative analysis, in which the scanning speed and accuracy of scanners (how often code numbers are recognized correctly, recognized incorrectly, or not at

all in a given time period) are measured, and a qualitative evaluation of the users' comments and scan behaviors.

5.1.1 Study Setup

The following major factors influence the success and time required to scan bar codes with mobile phones:

1. The *used mobile phone model* and its built-in camera. The camera's resolution, autofocus characteristics (such as focus speed and minimal focusable distance), as well as built-in software for autofocus and exposure control
2. The *selected set of test products* (e.g., the weight and shape of products) and the bar codes on these products
3. *Environmental conditions* such as the lighting situation
4. *Participants' experience* in scanning bar codes and the handling of certain mobile phone models

5.1.1.1 Used Mobile Phones

We selected three mobile phones with different characteristics for the test: The iPhone 3GS [78], iPhone 3G [79] and an Android HTC Desire [80] (see Figure 5.1). The iPhone 3GS has been selected as our standard platform, since it is a modern smartphone, features an autofocus camera and, due to its commercial success, most scanner applications are available and optimized for this platform. As a second phone, we selected the iPhone 3G that has a fixed-focus camera, in order to test the ability of scanners to recognize codes in blurry images. Third, we selected the Android HTC Desire, in order to be able to compare our solution to Google's most recent recognition engine, which is available only on this platform. In addition, the HTC Desire is a powerful phone (1GHz CPU) featuring a high-quality camera with an autofocus that is faster and able to focus on closer distances than the camera in the iPhone 3GS can. This allows us to observe the effect of the camera and autofocus quality on the results. We excluded C++ Symbian/QT devices (Nokia), since many tested recognition engines are not available on this platform and we had to limit the numbers of scanners considered in our study.

Figure 5.1 Mobile phones used for the study: An iPhone 3GS with autofocus camera (left image), iPhone 3G with a fixed-focus lens (middle image) running iOS, and the HTC Desire with a fast autofocus camera running Android OS (right image). The "ScanDK"-demo application uses our bar code recognition method.

Table 5.1 Bar code scanners tested in the user-study and scanner analysis. Scanners able to recognize codes in blurry images (e.g., on the iPhone 3G) are marked with an asterisk:

Scanner Name	Software Version	Tested Platform
RedLaser*	2.9.0	iPhone3G/3GS
Shopsavvy*	4.0.1	iPhone3G/3GS
ScanDK*	1.4.0	iPhone3G/3GS, HTC Desire
Pic2shop*	4.1	iPhone3G/3GS
NeoReader	2.00	iPhone3GS
i-nigma	3.07.01	iPhone3G/3GS
QuickMark	3.8.9	iPhone3GS
Barcode Ninja	1.0	iPhone3GS
Barcode Scanner	2.1	iPhone3GS
BarcodePlus	1.2	iPhone3GS
Google Build-In Scanner	OS version 2.2	HTC Desire

5.1.1.2 Tested Bar Code Scanners

The number of bar code scanners considered in this study has been limited, since we wanted to limit the expenditure of time by each participant to approximately one hour, while still covering a realistic set of demo products and three different mobile phone models. Among the scanners tested in our beforehand performed scanner analysis (Section 5.2), we selected all solutions that could recognize bar codes in blurry images, which coincides with the most renown and commercially successful scanners at the time of the study in November 2010: RedLaser [72], ShopSavvy [73] and pic2shop [74]. As a representative of all other scanners that performed well in our scanner analysis but were not able to recognize blurry bar codes, we selected i-nigma [81] on the iPhone 3GS because it is able to recognize rotated bar codes. On the Android platform, we compared our solution to the Google scanner. For each scanner, we used the most recent software version available at the time. Our recognition method is implemented in the ScanDK application (see Figure 5.1). Table 5.1 provides an overview of the tested applications on each device and their software version.

5.1.1.3 Test Products

We selected 11 test products for our user study. The set of products has been chosen to represent typical bar code types found in reality. It consists of 6 well recognizable codes and 5 more difficult codes. We deliberately included some difficult-to-recognize codes in order to challenge scanners, even though we expected that due to the blurry images, not all scanners on the iPhone 3G would recognize all codes. In order to be able to better compare the performance of scanners on the iPhone3G on blurry images, we had users scan a second test set consisting of four easy-to-recognize codes on books. All test products featured EAN13 or UPC-A bar codes, since not all scanners were able to read other code symbologies. Figure 5.2 shows the used demo products, their bar codes and briefly describes the specifics of each code.

5.1.1.4 Participants

We conducted the study with 16 participants (9 male, 7 female) ranging in age from 21 to 43. Participants came from several professional backgrounds. All participants owned a mobile phone: 4 owned an iPhone, 6 an Android device and 9 owned other devices, mostly phones from Sony Ericsson.

Figure 5.2 Test products: Easy-to-recognize code (1), normal bar code that was often scanned upside-down (2), colored and thin bar code resulting sometimes in blurry images (3), thin and multiple codes (4), code on non-straight and slightly transparent surface (5), slightly crumpled and transparent code frequently exhibiting glare (6), code on heavy product that was often scanned from an angular perspective (7), small code (8), crumpled code with often occurring glare (9), code on round surface (10), multiple codes (11), code on non-straight surface with frequently occurring glares (12).

Figure 5.3 General setup with demo products on table (right image) and a stationary web cam filming the global scene (left image).

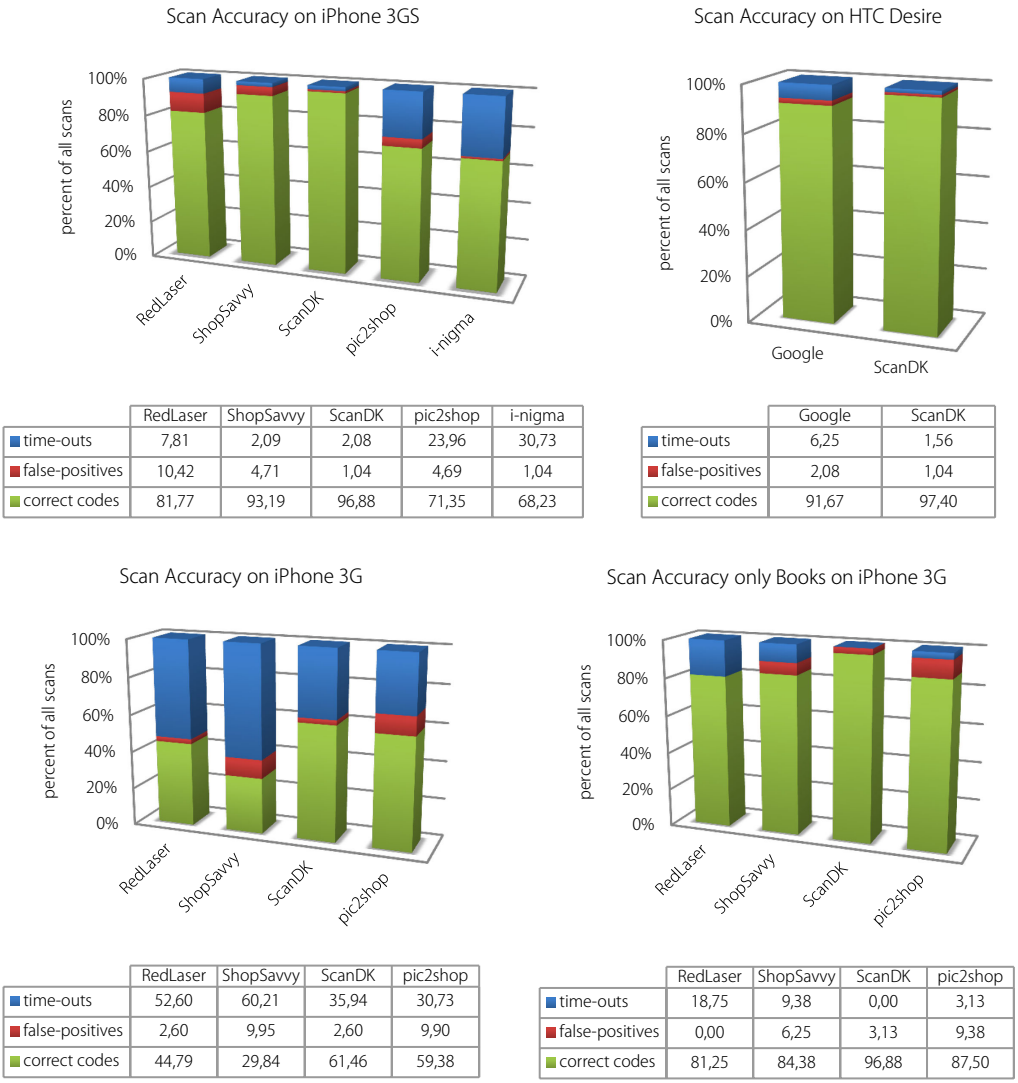
Figure 5.4 Setup to fixate web cam above mobile phone, in order to record user interaction and the scanning process with 30 frames per second (left image). Example screenshot from recorded video (right screenshot).

The study consisted of an introduction, three *scan sessions* (one for each mobile phone) and a final questionnaire. Each scan session consisted of a *scan run* for each tested scanner. Each scan run started with a brief introduction to the scan software and a test phase, in which participants could try to recognize the first test product (number 1 in Figure 5.2) until they felt confident with the scanner. Afterwards, participants scanned the bar codes of all demo products, always in the same order. At the end of each scan run, they were given a brief questionnaire about the scanner they had just tested. We rec-

orded both the global scene (see Figure 5.3) as well as the action on the mobile phone's screen (see Figure 5.4) with webcams. Recording the global scene allowed us to observe how users approach test products, e.g., if they take and align the product in their hands or if they move the phone to the product's bar code. Recording the phone's screen allowed us to perform the following analyses based on the recorded videos after the study:

- Measure the time an application required for each scan. We stopped the time between a product's bar code being visible on the phone's screen and its recognition. If the code could not be recognized within 15 seconds, we aborted and rated this try as *time-out*. In this case, users continued with the next product
- Record if a bar code has been recognized correctly (*correct*) or incorrectly (*false-positive*)
- Record user comments and reactions during the scan runs
- Observe how participants approach bar codes and orient the mobile phone, e.g., if they try to scan bar codes upside-down
- Observe how users react in case a code cannot be recognized after a few seconds, e.g., if they try to vary the distance to the code
- Analyze typical recognition problems, respectively why a code might not be recognized, e.g., if there is glare on the code's image or if the user is holding the phone too far away

The order in which participants tested the mobile phones (scan sessions) and individual scanner applications on these phones (scan runs) was randomized in order to account for learning effects. Table 9.1 and Table 9.2 in the appendix show the order in which different participants tested mobile phones and scanner applications.



	RedLaser	ShopSavvy	ScanDK	pic2shop	i-nigma
time-outs	7,81	2,09	2,08	23,96	30,73
false-positives	10,42	4,71	1,04	4,69	1,04
correct codes	81,77	93,19	96,88	71,35	68,23

	Google	ScanDK
time-outs	6,25	1,56
false-positives	2,08	1,04
correct codes	91,67	97,40

	RedLaser	ShopSavvy	ScanDK	pic2shop
time-outs	52,60	60,21	35,94	30,73
false-positives	2,60	9,95	2,60	9,90
correct codes	44,79	29,84	61,46	59,38

	RedLaser	ShopSavvy	ScanDK	pic2shop
time-outs	18,75	9,38	0,00	3,13
false-positives	0,00	6,25	3,13	9,38
correct codes	81,25	84,38	96,88	87,50

Figure 5.5 Scan accuracy on the iPhone 3GS (upper-left image), HTC Desire (upper-right image) and iPhone 3G (lower-left image) for all test products. iPhone 3G results for well recognizable codes on books (lower-right image). ScanDK is our method.

5.1.2 Quantitative Results

5.1.2.1 Scan Accuracy

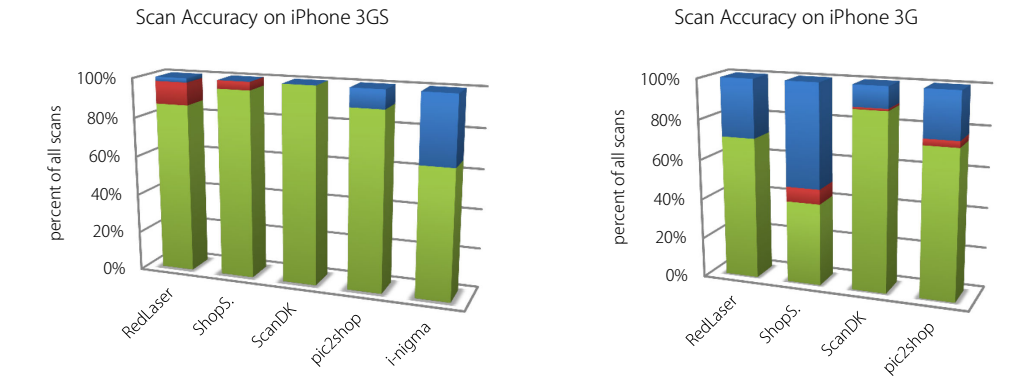
We evaluated scanner accuracy based on three variables: The number of bar codes recognized correctly and the number of codes recognized incorrectly in the set time limit of 15 seconds, as well as the number of time-outs, meaning no recognition within the time limit. Figure 5.5 shows the results for the tested scanners on the three mobile phones. Recognition accuracy on the iPhone 3GS on our main test set of products was in general very good, with pic2shop and i-nigma recognizing fewer codes correctly and a high number of false-positives in the case of RedLaser. On the HTC Desire, both the Google built-in solution as well as our scanner (ScanDK) was able to recognize most codes

with only few false-positives. On the iPhone 3G (without autofocus), scan accuracy was much lower, because of the blurry images in combination with our main test set of products that included problematic (e.g., crumpled) bar codes.

When looking at the scan accuracy for a well-recognizable subset of codes (products 1,2,3,7,8 and 11 in Figure 5.2) shown in Figure 5.6, it can be seen that on the iPhone3GS all scanners except i-nigma exhibit a good recognition performance, and on the iPhone3G scanner performance is much improved for most scanners. This corresponds with the results of the iPhone3G on our second test-set including only very well recognizable codes (see lower-right image in Figure 5.5). In return, looking at the scan accuracy for difficult-to-recognize codes (products 4, 9, 10 and 12 in Figure 5.2) reveals a decreased recognition accuracy of all scanners on the iPhone3G (see Figure 5.7).

This leads to the observation that recognizing bar codes on mobile phones without an autofocus camera works very well for clearly visible bar codes on a straight surface, such as those found on books, CDs, or electronic products, but has its limitations when trying to recognize bar codes on grocery items, with their large variety of flexible bar code surfaces and geometries.

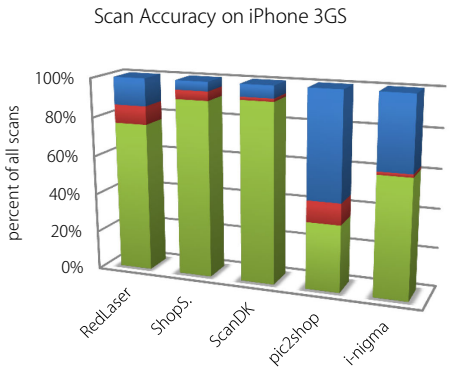
Our scanner was able to recognize more codes correctly than other solutions and produced fewer false-positive results, with the exception of RedLaser on the subset of easy-to-recognize codes on the iPhone 3G. These results also hold for different subsets of test-persons (novice or expert scanners) and parameters like the assumed time-out value of 15 seconds.



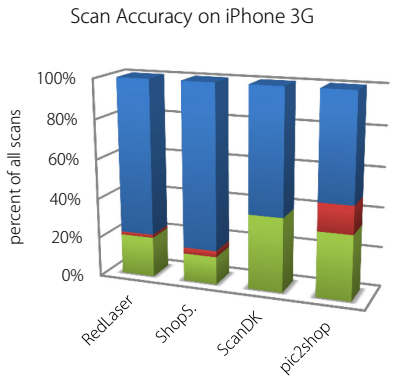
	RedLaser	ShopS.	ScanDK	pic2shop	i-nigma
time-outs	2,08	0,00	0,00	9,38	34,38
false-positives	11,46	4,21	0,00	0,00	0,00
correct codes	86,46	95,79	100,00	90,63	65,63

	RedLaser	ShopS.	ScanDK	pic2shop
time-outs	29,17	51,58	10,42	22,92
false-positives	0,00	7,37	1,04	3,13
correct codes	70,83	41,05	88,54	73,96

Figure 5.6 Results scan accuracy on iPhone 3GS (left image) and iPhone 3G (right image) for easy-to-recognize test products.

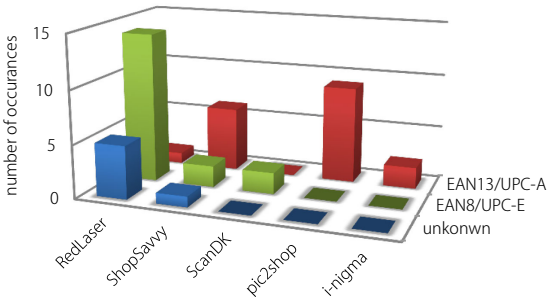


	RedLaser	ShopS.	ScanDK	pic2shop	i-nigma
time-outs	14,06	4,69	6,25	54,69	37,50
false-positives	9,38	4,69	1,56	10,94	1,56
correct codes	76,56	90,63	92,19	34,38	60,94

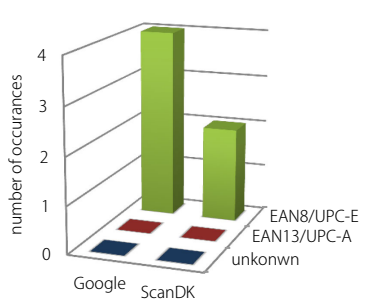


	RedLaser	ShopS.	ScanDK	pic2shop
time-outs	78,13	82,81	62,50	53,13
false-positives	1,56	3,13	0,00	14,06
correct codes	20,31	14,06	37,50	32,81

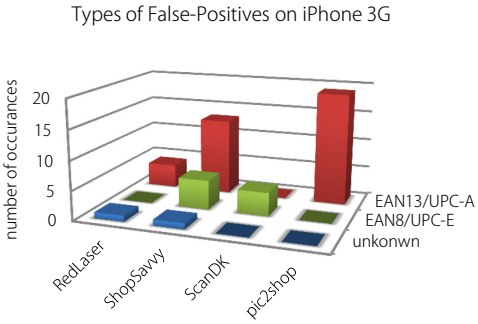
Figure 5.7 Results scan accuracy on iPhone 3GS (left image) and iPhone 3G (right image for difficult-to-recognize test products.



	RedLaser	ShopSavvy	ScanDK	pic2shop	i-nigma
unkonwn	5	1	0	0	0
EAN8/UPC-E	14	2	2	0	0
EAN13/UPC-A	1	6	0	9	2



	Google	ScanDK
unkonwn	0	0
EAN13/UPC-A	0	0
EAN8/UPC-E	4	2



	RedLaser	ShopSavvy	ScanDK	pic2shop
unkonwn	1	1	0	0
EAN8/UPC-E	0	5	4	0
EAN13/UPC-A	4	13	0	19

Figure 5.8 Number and symbology of wrongly recognized codes on iPhone 3GS (upper-left image), HTC Desire (upper-right image) and the iPhone 3G (lower-left image).

Since incorrectly recognized code numbers often pose a serious problem (e.g., for mobile services such as the allergy-check application presented in Figure 1.1), we analyzed the symbologies the incorrectly recognized codes had. Figure 5.8 shows the occurring types of false-positives based on all test products. All test-codes used in the user study were EAN13 or UPC-A codes. In the case of our recognition engine, all occurring false-positives were EAN8 or UPC-E codes. This is not surprising, since the parameters for false-positive avoidance have not been systematically adjusted for these shorter bar code symbologies so far, like they have been for EAN13 and UPC-A codes. This has two implications:

1. The number of occurred false-positive codes can likely be reduced by performing similar parameter optimizations already conducted in the case of EAN13 and UPC-A codes.
2. The recognition of EAN8 and UPC-E codes in applications that do not require them (e.g., services to books) could be deactivated. For example, pic2shop can recognize only EAN13 and UPC-A codes.

5.1.2.2 Scan Speed

We also evaluated the average time a single code scan required, based on the phone screens' recordings taken during the user study. We measured the time between the moment a product's code was visible on the phone's screen and had a reasonable, recognizable size until the recognition of this code (correct or false-positive). If a code could not be recognized within 15 seconds, we aborted and did not consider this time. Results are shown in Figure 5.9. Our scan engine is the fastest solution on each tested platform. On the iPhone 3G, our scanner was nearly three times as fast as the slowest solution (RedLaser). Especially in situations where the scan accuracy of readers is comparable, our solution has a major speed advantage over other scanners. For example, in the case of the iPhone 3G and easy-to-recognize codes on books, our solution is more than 5 times as fast as the slowest solution (RedLaser), or on the Android platform, where we required on average only 60% of the time for a scan the Google scanner requires. The timing results obtained depend on the assumed time-out value of 15 seconds. However, even if we vary the time-out value or the subset of considered products, the general results stay the same.

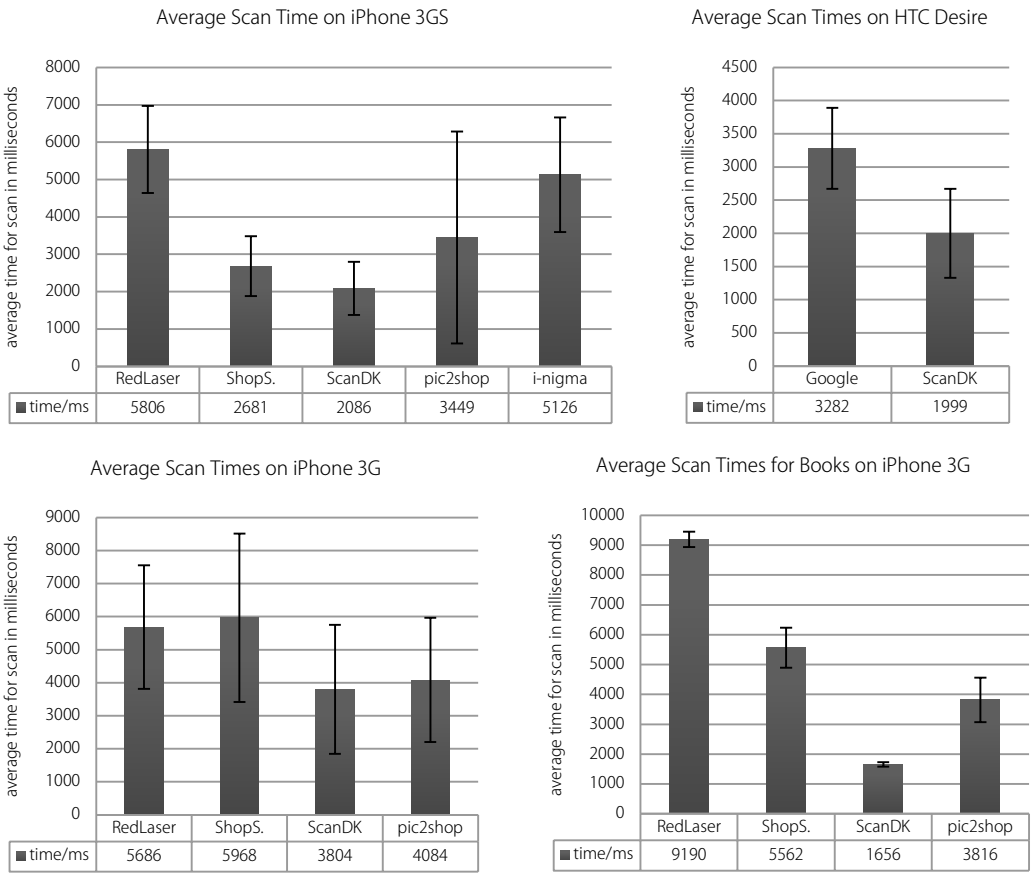
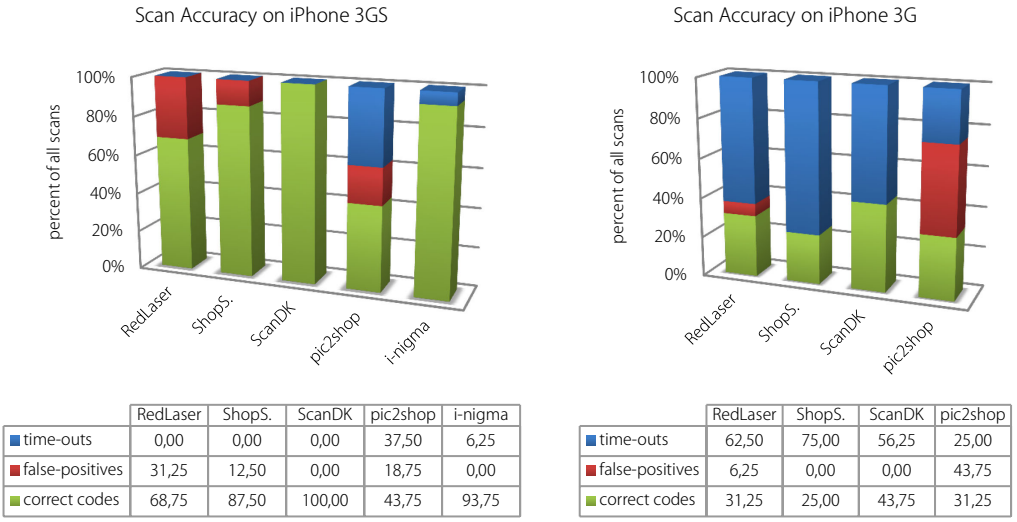


Figure 5.9 Average times for a single scan on the iPhone 3GS (upper-left image), HTC Desire (upper-right image), iPhone 3G (lower-left image) for all products and the iPhone 3G for well recognizable codes on books (lower-right image).



	RedLaser	ShopS.	ScanDK	pic2shop	i-nigma
time-outs	0,00	0,00	0,00	37,50	6,25
false-positives	31,25	12,50	0,00	18,75	0,00
correct codes	68,75	87,50	100,00	43,75	93,75

	RedLaser	ShopS.	ScanDK	pic2shop
time-outs	62,50	75,00	56,25	25,00
false-positives	6,25	0,00	0,00	43,75
correct codes	31,25	25,00	43,75	31,25

Figure 5.10 Scan accuracy for bar code on a round surface on the iPhone 3GS (left image) and iPhone 3G (right image).

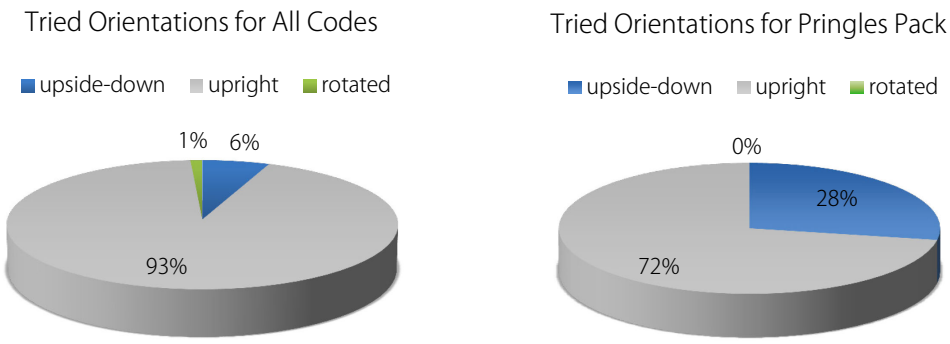


Figure 5.11 Percentage of situations in which users tried to scan codes from certain orientations for all test products (left image) and only for the Pringles test product (right image).

5.1.2.3 Alignment of Codes

In addition to scan accuracy and speed, we investigated how users align bar codes on the phone's screen. Based on the web cam videos, we analyzed how often people tried to scan codes upright, upside-down or rotated. Figure 5.11 shows the percentage of scans in which users tried to scan a code in a certain position. When considering all test products, in 6% of all scans users tried to scan a code upside-down. However, when looking at specific products such as the test product 2 shown in Figure 5.2 (a Pringles pack), which often resulted in the bar code being upside-down when someone took the product naturally in his or her left hand, 28% of users did try to scan the code upside-down. Analyzing user behavior on the videos showed that most users did not try to turn around an upside-down code, even if the reader was not able to recognize the code for some time.

Given all scans, in only 1% of cases users tried to scan codes that were not horizontally on the phone's screen but otherwise oriented. A possible reason for this is that in our test setup people had generally good access to the products and could hold most of them in their hands. In addition, the layout and user interface elements of most tested scanners instructed users to orient the codes horizontally.

5.1.2.4 Recognition of Round Codes in Blurry Images

When looking at the scan accuracy on the iPhone 3G in the case of the water bottle that features a round bar code (see Figure 5.10), it can be seen that the round code poses a challenge to all readers and is likely to produce false-positives (e.g., pic2shop with 44% false-positives on the iPhone 3G). However, a correct recognition in 44% of all scans in case of our scanner on the iPhone 3G indicates that the compensation for round codes described in Section

3.3.5 works in practice, especially since glare often complicated the recognition of this code.

5.1.2.5 Recognition Problems

Each time a code could not be recognized, we documented the prevailing conditions. Figure 5.13 shows how often certain conditions occurred in the case of time-outs for different scanners. Results indicate that in the case of i-nigma, the only tested scanner on the iPhone 3GS not able to scan blurry codes, blurry images were involved in many time-outs. The same problem with blurry codes also occurs for the Google scanner on the HTC Desire. On the iPhone 3G it is not surprising that in all time-outs, images have been blurry, however, special code shapes (crumpled, round or otherwise distorted codes) seem to be a prominent problem on the iPhone 3G.

These results indicate that blurry codes remain a problem, even for devices equipped with autofocus cameras. On such devices, images can be blurry, for example, if bar codes are small (and therefore the camera cannot focus on them), when the autofocus is slow, does not work or cannot focus on the bar code.

5.1.3 Qualitative Results

Qualitative results comprise both feedback from users obtained as part of the scanner and final questionnaire as well as recorded user comments and observed behavior.

5.1.3.1 Scanners-Related Feedback

After each scan run with a specific scanner, we asked users in the scanner questionnaire to rate the scanner's recognition speed as well as how easy it was to scan a code with this scanner. Figure 5.14 shows the results. Complementary to this feedback given by users, we counted the number of positive and negative user reactions towards the scan performance during the scan process, based on the recorded videos (see Figure 5.15).

In the final questionnaire we also asked users "Given the choice, which scanner(s) would you like to use?" on each platform. While most users preferred our scanner, pic2shop on the iPhone 3G and the Google scanner on the HTC Desire were also chosen often (see Figure 5.15). Asked why they preferred certain scanners over others, most users cited "speed of recognition" as the most important factor.

Results indicate that although all of the scanners were good and relatively fast (compared to results of previous studies like [77]), the difference in scan

speed and accuracy was noted by users and correlate with user ratings and reactions. Especially slow recognition speed seemed to provoke negative user reactions, whereas the effect of false-positives on negative user reactions was not that prominent. The latter may change, for example, if we had required users not to simply scan a product's code, but to retrieve product-related information, in which case an incorrectly recognized code would have a real impact.

Figure 5.12 User interfaces of scanners tested in the user-study. Interface types include rectangle-shaped (RedLaser, ShopSavvy and ScanDK), scan line-shaped (pic2shop, Google) as well as no (i-nigma) viewfinder elements.

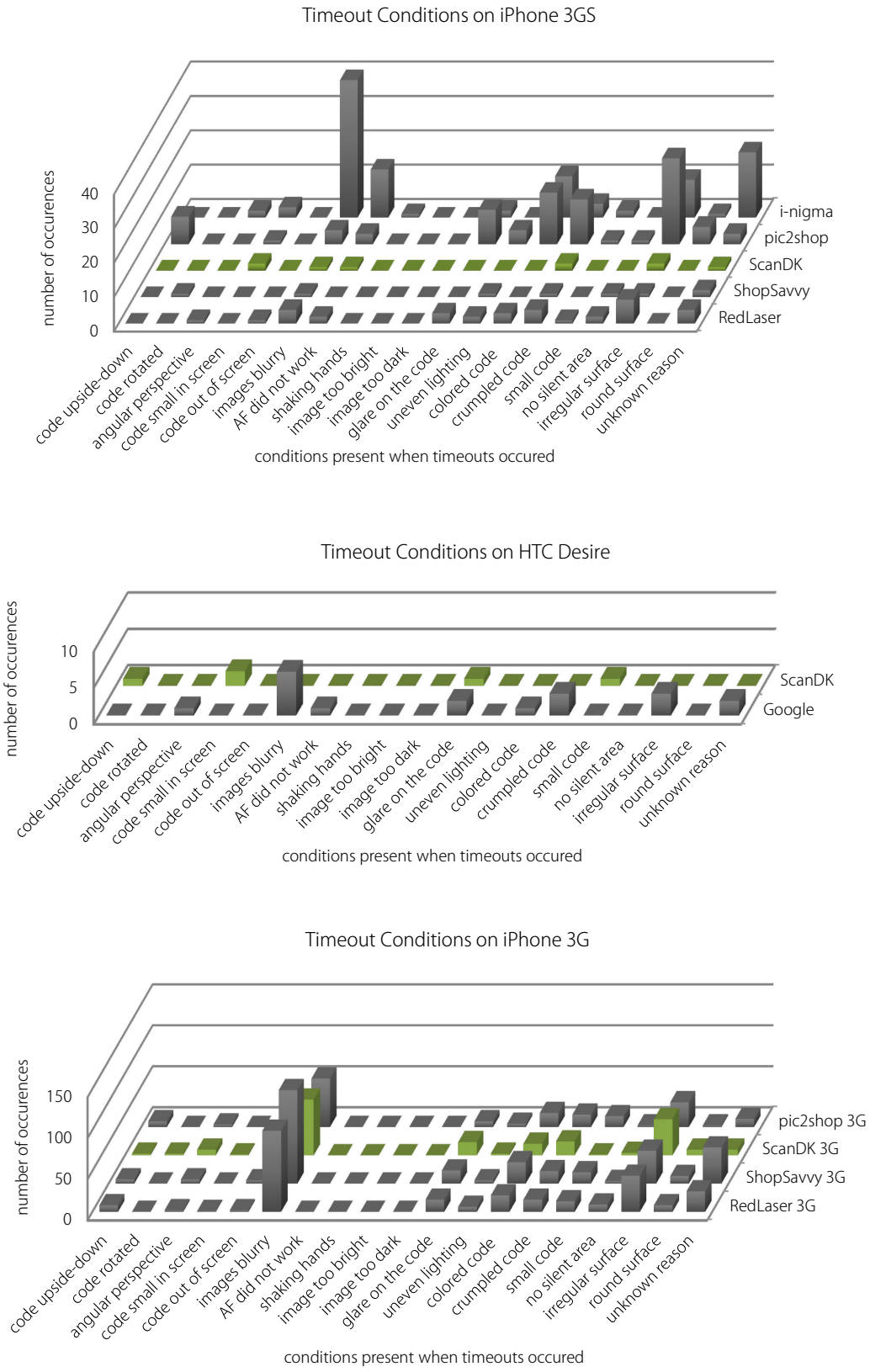


Figure 5.13 Prevailing conditions in case of time-outs for the different scanner applications and mobile phones. In case of the iPhone 3GS it is visible that although the mobile phone has a built-in autofocus camera, i-nigma could often not recognize a code because the image was blurry.

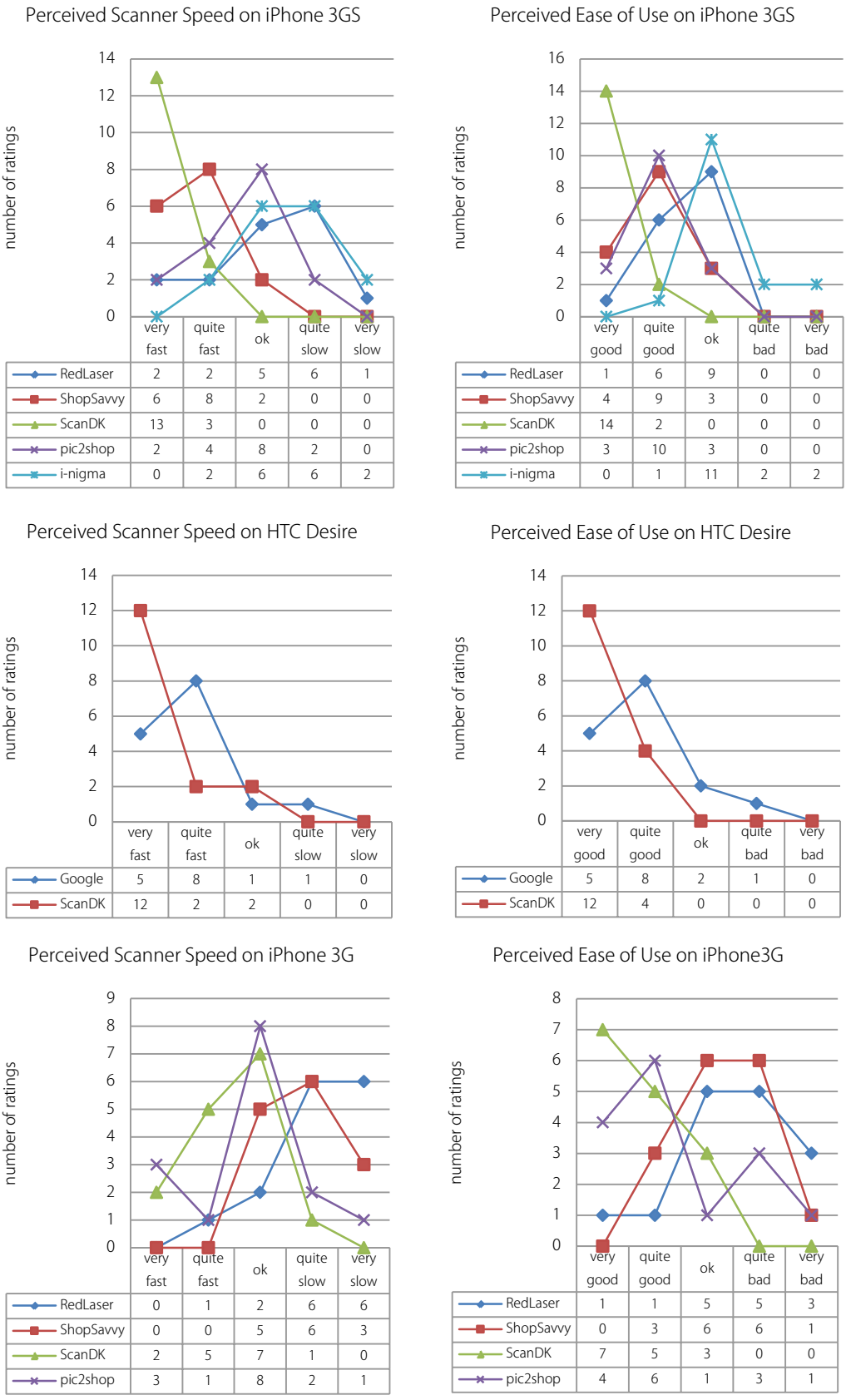


Figure 5.14 Scanner speed and scanner ease-of-use as rated by users.

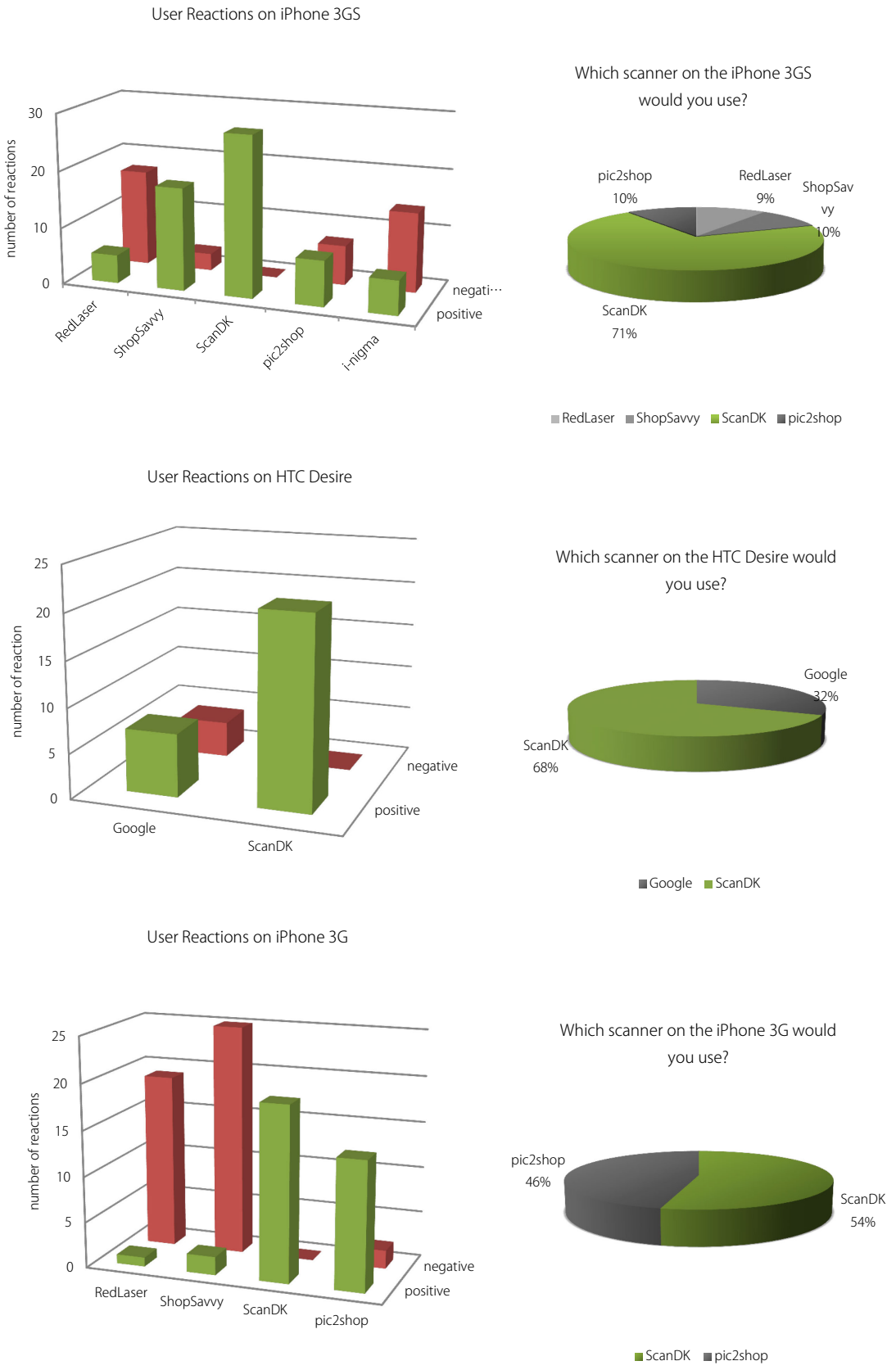


Figure 5.15 Number of positive and negative user reactions towards the performance of scanners (left images). Number of times a specific scanner was selected after asking users "Given the choice, which scanner(s) would you like to use?" (right images).

5.1.3.2 User Interface-Related Feedback

The tested scanners feature different types of user interfaces (see Figure 5.12). The user interfaces can be divided into three classes based on the type of viewfinder elements they provide:

1. Rectangle-shaped viewfinder elements, requiring or motivating the user to locate the bar code inside the rectangle (RedLaser, ShopSavvy, our scanner and Google)
2. Scan line-shaped viewfinder elements, requiring or motivating the user to ensure the displayed line crosses the bar code on the screen (pic2shop, Google)
3. No user interface elements (i-nigma)

Asked what kind of viewfinder type they preferred, 9 users answered the rectangle-shaped viewfinder, 10 preferred the line and none preferred no viewfinder at all (multiple selections were possible). Regarding the scan orientation (landscape or portrait), 11 users preferred scanning in portrait mode and 1 user the landscape orientation. Another user stated he would like to have the choice. Regarding user comments and reactions during the scan runs, the scan line based user interfaces were perceived as very easy to understand and use. Among the comments in the case of pic2shop were: "*The good thing about the red line is that he tells me how to hold it (the phone)*" [22] or "*very user friendly*".

In contrast, the feedback provided by pic2shop and by other scanners when the bar code could not be recognized in the first few seconds, was often unclear to users. Comments included "*I don't know what this red or green means, it is completely random. It showed me already so many green lines, the code should be definitely recognized by now*" (pic2shop), "*I don't get it, you have no clue about what is happening*" (i-nigma) or "*I'm not sure if I'm too stupid or the scanner*" (RedLaser).

5.1.3.3 General Observations and User Behavior

Based on user comments and behavior we made the following observations:

[22] User comments have been translated from German. The original comments transcribed from the recorded videos are listed in Appendix 9.2.2.

- It seems difficult for many users to determine which codes are easy and which are hard to recognize: While it is clear to most users that very crumpled, thin or small codes are difficult to recognize, round codes, codes on uneven surfaces like the water pack, toothpaste or otherwise difficult codes were not perceived as difficult. Comments included: "*I wondered a few times why certain codes are recognized so fast and others not at all. How can this be?*", "*What is so difficult about this code?*" or "*Why do they (scanners) have problems with this code?*"
- Many users do not seem to recognize glare as a problem: Users who were experienced[23] with bar code scanners tried to avoid getting glare on codes. However, many others did not try to avoid glare, e.g., in the case of the water bottle, even if the code could not be recognized.
- It does not seem clear to all users that a well-aligned code on the phone's screen improves the recognition. Some users held the phone too far from the code, resulting in the code being small on the screen. Others held the phone at a sharp angle to the code and remained in such positions. Comments in such situations included for example: "*He (the scanner) does not like that (code). But he (the code) looks so easy.*"
- If codes could not be recognized in the first few seconds, many users did not try to change the situation (e.g., by varying the distance to the code), but tried instead to hold the phone even more steadily (as indicated on some user interfaces). Related comments included: "*Yeah, I'm holding still man... do something!*"
- In the case of multiple or very thin bar codes on the image, users found it easier to align the phone if the user interface had a line-based viewfinder; especially in comparison with a rectangle-shaped viewfinder or no viewfinder elements at all.

5.2 Scanner Analysis

Complementary to our user study, we performed a detailed analysis of bar code scanners under controlled conditions. Results obtained in the user study presented in the previous section do not necessarily represent the raw per-

[23] Users who have used bar code scanning on their mobile phones before our user study.

formance of bar code scanning algorithms, but the "real-world" performance of complete systems. They are influenced by test users (e.g., novice or experienced users), hard to control environmental conditions (e.g., shadows and glare on codes, dependent on the scan position) and other imperfections (e.g., uncontrolled autofocus behavior).

In the scanner analysis, we investigated scanner features (e.g., the kind of bar code symbology they can read), as well as recognition accuracy and speed for different bar code types and lighting conditions. In contrast to the user study, in which scanners were tested in a realistic environment, we deliberately ruled out the human factor (the complete study was performed by a single, expert scanner) as well as the environmental factor (the analysis was performed under controlled conditions) in order to obtain a clearer picture of the strengths and limitations of scanners. In this section we present the set-up of our analysis and selected results in order to complement the findings of our user study.

5.2.1 Analysis Setup

5.2.1.1 Tested Bar Code Scanners

The scanners that were tested included all applications that had been tested in the user study in addition to five scanners that were not able to recognize codes in blurry images but which exhibited a good scan performance. We used the same mobile phones as in the user study. Out of all available applications, the applications tested were selected based on the following criteria:

- They were available in the iPhone App store at the time of our analysis (November 2010)
- They exhibited a good scan performance
- They could recognize arbitrary EAN13 and UPC12 codes. (For example, some applications were able to recognize only bar codes of books starting with "978...")

5.2.1.2 Analyzed Conditions

We analyzed recognition speed and accuracy in dependence on the following conditions:

- *Code size:* Ranging from very small codes to very large bar codes
- *Code color:* Including codes with varying bar and background colors
- *Code shape:* Ranging from straight codes to very crumpled codes

- *Difficult codes:* Including damaged bar codes or codes with text written on top of them
- *Lighting conditions:* Including glares, shadows and very dark environments

Figure 5.17 provides an overview of the used test products as well as lighting conditions for each test. Each scanner was tested in 31 situations. With each scanner and mobile phone, we performed five scans for each test situation, resulting in a total of 155 scans for each scanner. We placed each test product in our setup (see Figure 5.16) that allowed us to control the lighting conditions. For each scan run, we measured the time between removing the mobile phone from a defined start position until the bar code was recognized. We moved mobile phones as quickly as possible and always directly to an optimal position in front of the bar codes:

- We ensured that the perspective on the code is directly from above and not angular,
- The code had an optimal size on the phone's screen (tested beforehand for each scanner),
- The images were perfectly sharp (to account for scanners that cannot recognize codes in blurry images)

We again recorded the mobile phone's screen during the tests using the setup shown in Figure 5.16. Based on the recordings, we measured the time each scan required and whether a code has been recognized correctly or incorrectly. If a code could not be recognized after 20 seconds, we aborted.

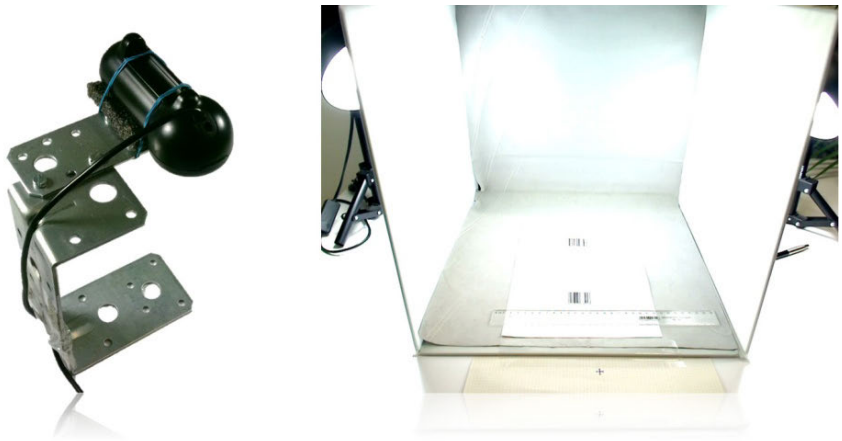


Figure 5.16 Mobile phone and camera holder used to record the phone's screen (left image). Set up of our scanner analysis (right image). We used two daylight lamps in combination with a photo tent for controlled lighting conditions.

Figure 5.17 Subset showing 20 out of the 31 tested bar codes and conditions in the scanner analysis.

Table 5.2 Features supported by scanner applications in the case of sharp images (on the iPhone 3GS and HTC Desire):

Scan Features	Red-Laser	Shop-Savvy	ScanDK	pic2 shop	i-nigma	Quick Mark	Bar-code Ninja	Scan-ner	Bar-code Plus	Google
SUPPORTED BAR CODE SYMBOLOGIES										
EAN13/UPC-A	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
EAN8	YES	YES	YES	NO	YES	YES	NO	YES	YES	YES
UPC-E	YES	NO	YES	NO	YES	NO	NO	NO	YES	YES
EAN128	NO	NO	YES	NO	YES	NO	NO	YES	YES	YES
Code39	NO	NO	YES	NO	NO	NO	NO	YES	YES	YES
GENERAL FEATURES										
Auto. Symbology Detection	PARTIALLY	YES	YES	YES	YES	YES	YES	YES	YES	YES
Code Position Highlighting	NO	NO	YES	NO	NO	NO	NO	NO	YES	YES
SCANNING OF ROTATED BAR CODES										
Upside-Down Codes	YES	YES	YES	NO	YES	YES	NO	YES	YES	YES
Codes in 90 Degree	NO	YES	YES	NO	YES	YES	NO	YES	YES	NO
Arbitrarily Oriented Codes	NO	NO	YES	NO	YES	NO	NO	NO	NO	NO

Table 5.3 Features supported by scanner applications in the case of blurry images (on the iPhone 3G):

Scan Features	RedLaser	ShopSavvy	ScanDK	pic2 shop
SUPPORTED BAR CODE SYMBOLOGIES				
EAN13/UPC-A	YES	YES	YES	YES
EAN8	YES	YES	YES	NO
UPC-E	YES	NO	YES	NO
EAN128	NO	NO	NO	NO
Code39	NO	NO	NO	NO

GENERAL FEATURES				
Auto. Symbology Detection	PARTIALLY	YES	YES	YES
Code Position Highlighting	NO	NO	YES	NO
SCANNING OF ROTATED BAR CODES				
Upside-Down Codes	YES	YES	YES	NO
Codes in 90 Degree	NO	YES	YES	NO
Arbitrarily Oriented Codes	NO	NO	YES	NO

5.2.2 Results of the Feature Analysis

We tested each bar code scanner application for the following features:

- *Supported bar code symbologies:* Not all available scanners are able to read all bar code symbologies relevant for mobile services (see Section 2.2 for details regarding the different symbologies).
- *Auto-symbology detection:* Not all scanners are able to automatically detect a bar code's symbology in the image. Some require users to manually set the symbology of certain bar codes that should be recognized in the application settings, or provide a switch on the scanner's user interface.
- *Code position highlighting:* Indicates whether a scanner highlights the recognized code visually after recognition. This feature reassures users that the correct code has been scanned when multiple bar codes are present in an image.
- *Upside-down code scanning:* Indicates whether a scanner is able to read codes positioned upside-down on the phone's screen.
- *Vertical code scanning:* Indicates whether a scanner is capable of reading codes positioned vertically on the phone's screen. This is often achieved by rotating one or multiple scan lines 90 degree and does not require a scanner to be able to read arbitrarily rotated bar codes.
- *Rotated code scanning:* Indicates whether a scanner is able to recognize codes arbitrarily rotated on the phone's screen.

Table 5.2 presents the supported features of scanners in the case of sharp images (e.g., on the iPhone 3GS and HTC Desire) and Table 5.3 shows the same for blurry images such as on the iPhone 3G. On the latter, several features are harder to achieve due to the blurry images obtained on this device.

Flexibility regarding the exact alignment of bar codes on the phone's screen is relevant in practice, especially in situations in which the exact alignment is difficult. For example, for heavy products, items on high shelves, or products locked in show-cases. Features supporting this flexibility are the possibility to scan upside-down, vertical or otherwise oriented bar codes on the screen. An-

other important factor is the largest and smallest allowed size of bar codes on the phone's screen for being still recognizable. We measured these values for each scanner and calculated the *dynamic range* of each scanner as follows: *dynamic range = lp – sp*, whereas *lp* corresponds to the largest still recognizable code size in percent of screen width and *sp* to the smallest still recognizable code size in percent of screen width. The larger the dynamic range, the more flexible scanners are with respect to the required distance to the code. Figure 9.3 in the Appendix shows the determined dynamic range for tested scanners on the iPhone 3GS.

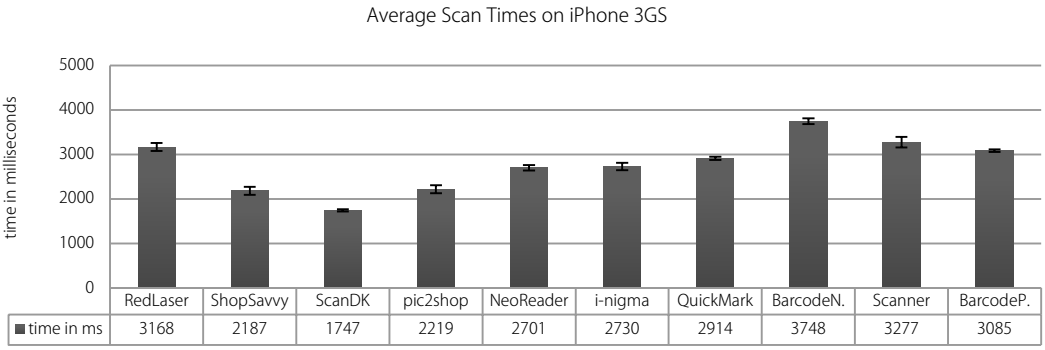
5.2.3 Quantitative Results

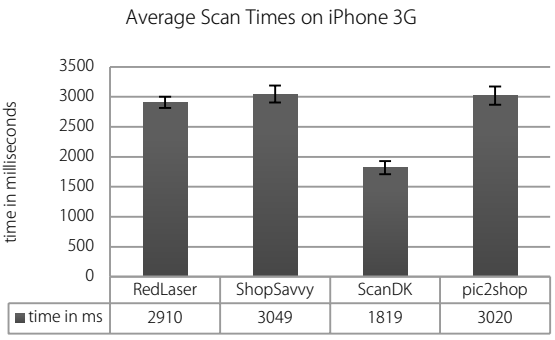
When comparing scan accuracy and the average time required for a single scan based on all 31 tests, the relative results between scanners correspond generally to results obtained in the user study. Figure 5.19 shows the measured scan accuracy. For the iPhone 3G the number of correctly recognized codes in the scanner analysis is in general higher and the differences among scanners are slighter. This might be attributable to the optimized scan behavior and controlled lighting conditions. Figure 5.18 presents the scan speed.

5.2.3.1 Speed and Accuracy Advantage of Blurry Code Recognition

When looking at the scan accuracy for very small bar codes (see Figure 5.20), it can be seen that the accuracy of most scanners that cannot recognize blurry bar codes on the iPhone 3GS is quite low. This is due to the fact that the iPhone's camera cannot focus on very close objects and the images remain slightly blurred.

Figure 5.22 shows the scan speed for easy-to-recognize bar codes. The iPhone 3GS scanners that are able to recognize codes in blurry images are (with the exception of RedLaser) usually faster than scanners requiring sharp images. This is the case, despite the fact that all scanners exhibit a good scan accuracy and the perfect positioning of all scanners during the test that ensured cameras could focus on the codes. This speed advantage is a result of the time the camera required to focus. Results on the HTC Desire are even more pronounced, with the Google scanner requiring nearly three times (2.72) as long for a scan than our solution.





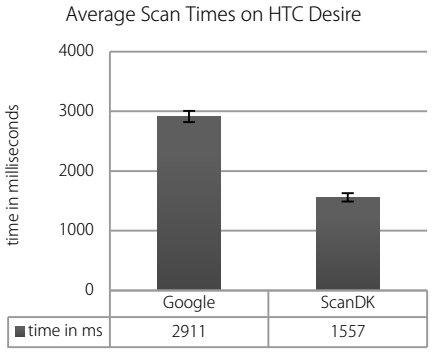
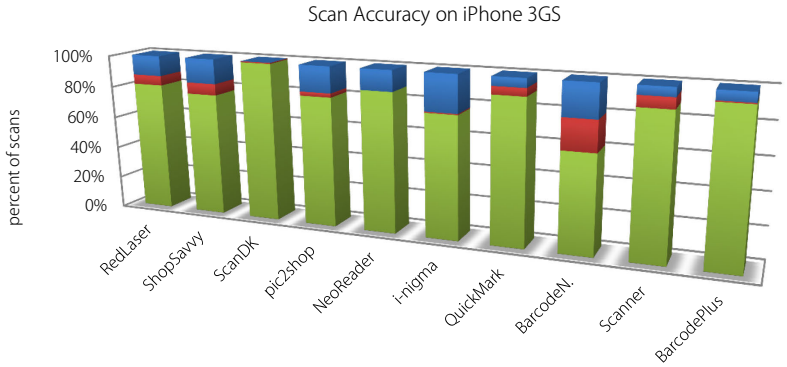
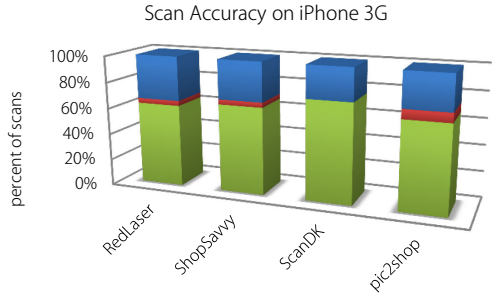
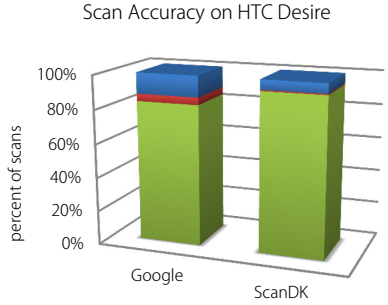


Figure 5.18 Average scan speed for a single scan, based on all 31 tests.



	RedLaser	ShopSavvy	ScanDK	pic2shop	NeoReader	i-nigma	QuickMark	BarcodeN.	Scanner	BarcodePlus
time-outs	12,73	15,57	0,00	16,35	12,90	23,08	5,52	20,42	5,42	5,77
false-positives	6,06	7,19	0,64	2,52	0,00	0,64	4,91	18,85	6,63	0,64
correct codes	81,21	77,25	99,36	81,13	87,10	76,28	89,57	60,73	87,95	93,59

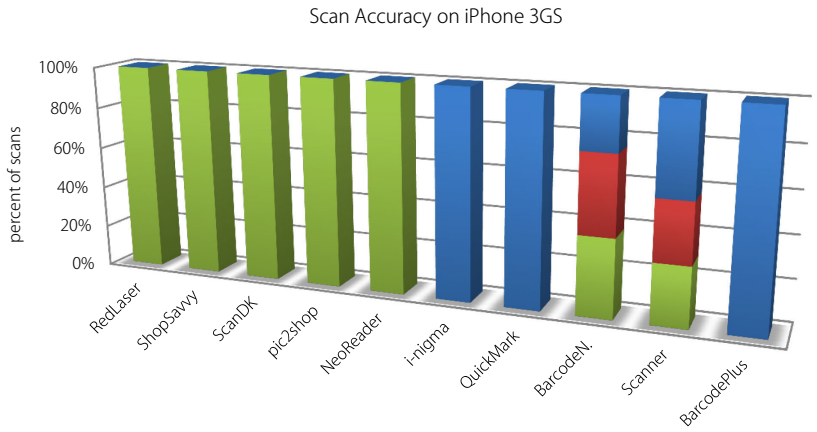




	RedLaser	ShopSavvy	ScanDK	pic2shop
time-outs	33,54	29,81	24,52	26,35
false-positives	3,73	3,73	0,00	7,19
correct codes	62,73	66,46	75,48	66,47

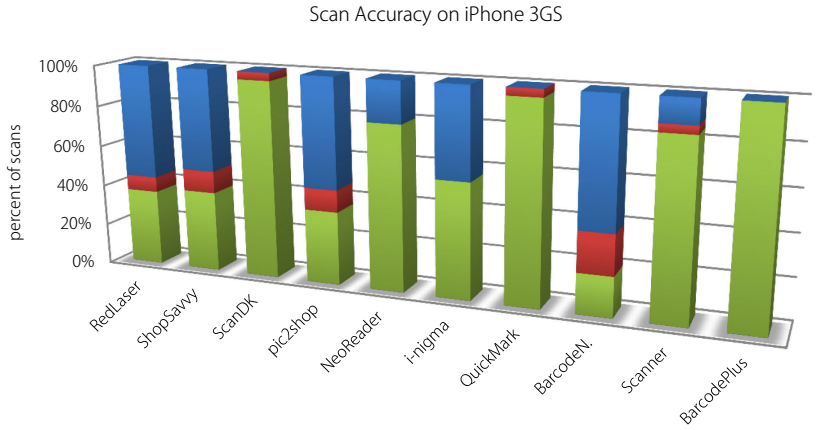
	Google	ScanDK
time-outs	12,35	5,77
false-positives	4,32	0,64
correct codes	83,33	93,59

Figure 5.19 Scan accuracy for all 31 test situations.



	RedLaser	ShopSavvy	ScanDK	pic2shop	NeoReader	i-nigma	QuickMark	BarcodeN.	Scanner	BarcodePlus
time-outs	0,00	0,00	0,00	0,00	0,00	100,00	100,00	25,00	42,86	100,00
false-positives	0,00	0,00	0,00	0,00	0,00	0,00	0,00	37,50	28,57	0,00
correct codes	100,00	100,00	100,00	100,00	100,00	0,00	0,00	37,50	28,57	0,00

Figure 5.20 Scan accuracy on the iPhone 3GS for a small bar code.



	RedLaser	ShopSavvy	ScanDK	pic2shop	NeoReader	i-nigma	QuickMark	BarcodeN.	Scanner	BarcodePlus
time-outs	55,56	50,00	0,00	53,57	20,00	44,00	0,00	61,29	11,54	0,00
false-positives	7,41	10,71	3,85	10,71	0,00	0,00	3,85	19,35	3,85	0,00
correct codes	37,04	39,29	96,15	35,71	80,00	56,00	96,15	19,35	84,62	100,00

Figure 5.21 Scan accuracy on the iPhone3GS for codes on a non-straight (e.g., crumpled or round) surface.

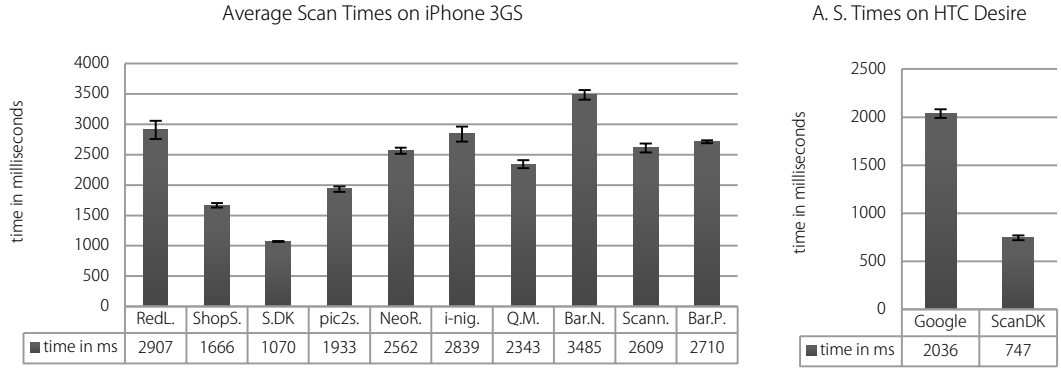
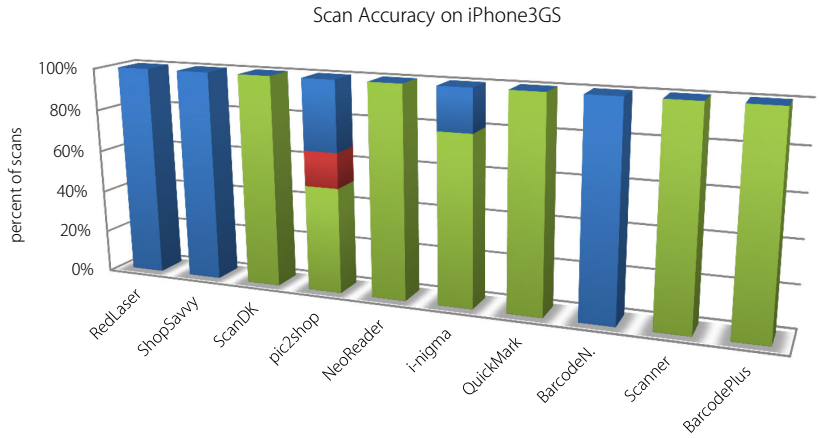
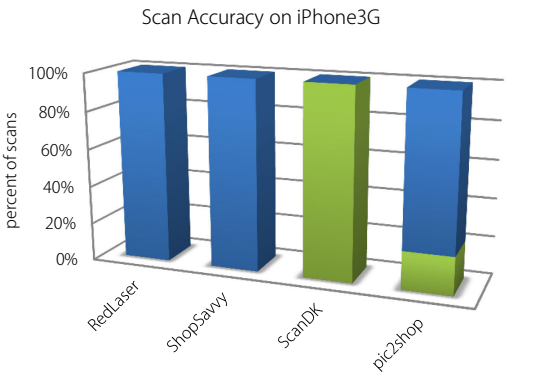


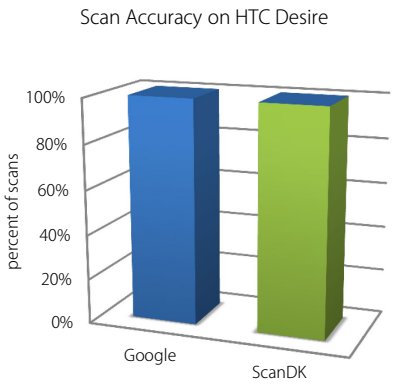
Figure 5.22 Speed results for easy-to-recognize bar codes.



	RedLaser	ShopSavvy	ScanDK	pic2shop	NeoReader	i-nigma	QuickMark	BarcodeN.	Scanner	BarcodePlus
time-outs	100,00	100,00	0,00	33,33	0,00	20,00	0,00	100,00	0,00	0,00
false-positives	0,00	0,00	0,00	16,67	0,00	0,00	0,00	0,00	0,00	0,00
correct codes	0,00	0,00	100,00	50,00	100,00	80,00	100,00	0,00	100,00	100,00



	RedLaser	ShopSavvy	ScanDK	pic2shop
time-outs	100,00	100,00	0,00	80,00
false-positives	0,00	0,00	0,00	0,00
correct codes	0,00	0,00	100,00	20,00



	Google	ScanDK
time-outs	100,00	0,00
false-positives	0,00	0,00
correct codes	0,00	100,00

Figure 5.23 Scan accuracy in the case of shadow on the bar code.

5.2.3.2 Limitations of Blurry Code Recognition

Despite its advantages in scan speed and accuracy for well-lit, straight bar codes, blurry code recognition has its limitations when the code geometry is round, crumpled or otherwise distorted and lighting is difficult. This can be seen in the case of very blurry images, but there was also a decline in scan performance among the scanners that were capable of recognizing blurry codes on the iPhone 3GS. Figure 5.21 shows the average scan accuracy on a set of difficult test runs. With the exception of our scanner, all scanners able to scan blurry bar codes exhibit a poor performance, while many scanners that are able to recognize only sharp images show much better results (except for BarcodeNinja, which performed poorly in general). We do not know what

methods scanners use to decode bar codes; however, the results indicate that the methods used by scanners that were capable of scanning blurry codes are less suited for difficult conditions. Our scanner performs well on all tests. This can likely be attributed to the two decoder approach (see Chapter 3), which supports our design decision. Figure 5.23 shows the scan accuracy in case of shadows on the bar code. Results confirm that our compensation for uneven lighting introduced in Section 3.3.4 performs well.

5.3 General Results and User Interface Guidelines

Based on our user study and scanner analysis, the implemented demonstrators as well as feedback from industry partners such as the Metro Future Store Initiative [12] or the Markant Group [13], we made several observations regarding the scan behaviors of users and limitations of the tested scanners. In this section, we summarize these observations (Section 5.3.1) and propose guidelines for designing user interfaces of bar code scanners for mobiles phones (Section 5.3.2).

5.3.1 General Results

5.3.1.1 Performance of Bar Code Scanners

Regarding the performance of bar code scanners, we made the following observations:

- Recognizing bar codes on devices without autofocus cameras works well for easy-to-recognize bar codes on flat surfaces, such as those typically found on books, DVDs, or electronics, but there are serious limitations in the case of irregular surfaces, such as crumpled codes often found on grocery items.
- Blurry images are still a problem even on devices with autofocus cameras. They complicate, or even prevent, the recognition of small bar codes and slow down the recognition process.
- The scan accuracy of most tested scanners for easy-to-recognize bar codes is good. However, there are large differences among scanners for difficult bar codes and poor lighting conditions (e.g., shadows).
- The same relative differences in scan speed and accuracy of scanners observed under controlled conditions and obtained with an expert user

(in the scanner analysis), are also observed in the case of realistic conditions and less experienced users (in the user study).
- False-positives occur frequently (up to 10% of all scans for some scanners). False-positives are most common with bar codes that are not flat, and when the phone is further away from the bar code, so that the bar code appears very small on the phone's screen but is still recognizable.

5.3.1.2 Scan Behavior of Users

We made the following observations about the scan behaviors of users:

- Most users try to orient bar codes horizontally on the phone's screen, if they can (e.g., when they have good access to products). Only few users tried to scan bar codes that are not horizontally aligned on the screen but otherwise oriented.
- Users will often not distinguish bar codes being upright from those that are upside-down on the screen.
- The most important criterion for users to choose a specific scanner is scan speed (given false-positives are not considered). They expect bar codes to be recognized within the first few seconds.
- It is difficult for users to distinguish among easy- and hard-to-recognize bar codes, especially on mobile phones without an autofocus camera and with the exception of obviously crumpled or damaged bar codes.
- Many users do not realize that even minor glares, an angular perspective on codes or the wrong distance to the bar code pose major problems for most scanners.
- If bar codes are not recognized after a few seconds, users generally do not know how to proceed in order to maximize their chances of having the scanner recognize the bar code.
- Most users prefer holding the phone in portrait orientation while scanning.
- Users prefer viewfinder elements that help them to align the bar code on the phone's screen. Even though users, according to our questionnaire, preferred rectangle- and line-shaped viewfinder elements alike, user feedback recorded during the scan runs indicated that interfaces that feature line-shaped viewfinders are perceived as easier; especially when multiple codes or thin bar codes are present in an image.
- The most common problems related to user behavior that prevented the recognition of codes were the scanning from an angular perspective, and holding the mobile phone too far from the code.

5.3.2 User Interface Guidelines

In practice, the user interface of a bar code scanner should serve multiple, sometimes even mutually exclusive purposes. Depending on the underlying application, it should allow both novice and expert users to scan bar codes as quickly and effortlessly as possible, entertain users, or communicate information like a brand name or logo, to name but a few goals. In terms of the following guidelines, we focus solely on optimizing the recognition process itself, and try to find a compromise that is suitable for experienced and inexperienced scan users alike.

The process of recognizing a bar code can be separated into different, potentially overlapping phases, based on the time passed:

1. The *alignment phase*: From starting the scanner application until the phone is positioned in front of the bar code that should be recognized and the code is aligned on the phone's screen
2. The *early recognition phase*: Starts after the bar code has been aligned and lasts for the first 3-5 seconds, or until the code has been recognized or the user aborted the recognition process
3. The *extended recognition phase*: Starts after the early recognition phase in the case no code could be recognized, and lasts either until a code has been recognized or the user aborted the recognition process
4. The *feedback phase*: Starts after a code has been recognized and lasts until the next, application-specific action starts (e.g., the focus switch from the recognition screen back to the application's main user interface)

Each phase has different goals that should be achieved in order to optimize the overall recognition process in terms of speed and ease-of-use:

1. In the alignment phase, users should be supported in aligning the bar code on the phone's screen in an optimal position. The optimal position depends on the scan technology used, but usually means that the code is horizontally aligned, has a size on the phone's screen that is optimal for the specific scanner (typically around 70% of the screen's width), and the perspective on the code is from straight above.
2. In the early recognition phase, feedback should be provided to the user in order to signal the presence of a bar code in the images and the operation of the bar code recognition process. In the case of easy-to-

recognize bar codes, the code is usually recognized immediately or after the first few seconds.

3. If the code could not be recognized in the early recognition phase, detailed feedback should be given to users about how to proceed. Preferably, this should be context-sensitive, based on the problem at hand. Users should be instructed to align the code in an optimal position (see details in alignment phase), to avoid glares and shadows on the code, and to ensure that images are sharp. If the code still cannot be recognized, users should slowly vary the distance to the bar code in order to maximize the odds for recognition. If a code is non-recognizable (such as when the code features an unsupported symbology or is damaged), this should be communicated to the user. For such situations, fallback solutions should be available that allow for the manual entering of a product's name or bar code number.
4. After the code has been recognized, direct feedback should be provided to users. The suitable form of feedback, e.g., optical, acoustic, or haptic through vibrations, depends on the user's preferences and context. In the case of multiple bar codes in the image, the recognized code should be highlighted.

Figure 5.24 Images illustrating the state of the user interface during the alignment phase (left image), extended recognition phase (middle image) and feedback phase (right image).

5.3.2.1 User Interface Concept

The following user interface (UI) concept complies with the above-presented guidelines and represents one possible UI implementation for scanners that addresses both novice users and scan experts. It serves as an example and has not been evaluated with test users (compare to screens shown in Figure 5.24):

1. *Alignment phase:* After starting the scan process, the UI is simple and contains only a scan line that indicates to users to place the scan line through the bar code (see left image in Figure 5.24). The length of the scan line indicates the approximate size that a bar code should have on the screen. Furthermore, the scan line helps users to align the phone in the case of thin bar codes or multiple codes correctly. Furthermore, advanced users can avoid local glare and damages on codes. No additional hints are provided at this phase. Our assumption is that expert users do not need further recognition tips, and novice users do usually start without first reading instructions, or sometimes do not understand them (e.g., when they are written in a foreign language that the user does not understand).
2. *Early recognition phase:* If the recognition algorithm detects a bar code, we change the color of the scan line to signal to users that the recognition engine is working. No additional feedback is provided, since experience shows that in most cases bar codes are either recognized immediately, or a longer phase requiring corrections from the user follows.
3. *Extended recognition phase*: If no code has been recognized, we provide feedback to the user in order to ensure that the code has the optimal size on the screen, and the perspective on the code is from straight above. This is done by drawing rectangles that indicate the optimal and the current code position, as well as arrows highlighting the difference between these two rectangles (see middle image in Figure 5.24). If the code still cannot be recognized after being close to the optimal position, we can vary the size of the target rectangle that indicates the optimal code position so that users can adjust the distance to the code. Highlighting the current bar code's position also ensures users that the correct code is processed. If the code still cannot be recognized, we advise the user that he or she should enter the code number manually.

4. *Feedback phase:* After a code has been recognized, we provide immediate feedback by flashing the screen, playing a brief sound or activating the phone's vibration alert. In addition, the color of the scan line is changed to green and the recognized bar code is highlighted on the screen.

At any time, the user can enter the code number or name manually and abort the recognition process by pressing a button.

5.4 Summary

Compared to earlier studies [77], all tested scanners showed much-increased recognition speed, accuracy, and performed well on easy-to-recognize codes. However, substantial differences between scanners remain. Our user study and the scanner analysis showed that our solution outperforms other solutions in terms of scan accuracy and recognition speed. The advantage of our scan engine compared to the other considered solutions is especially apparent in the case of difficult bar codes (e.g., codes on crumpled surfaces) and poor lighting conditions such as shadows.

In addition to the comparison of bar code recognition engines, we made some general observations about users' scan behaviors and gave guidelines for designing user interfaces for bar code scanners on mobile phones.

6 Mobile Services

While being a broadly applicable identification technology, one important use case for bar code recognition on mobile phones are consumer-oriented mobile applications that provide services and information to products. This section presents an overview of such "mobile services".

We provide an exemplary overview of application scenarios (Section 6.1) and analyze the strengths and weaknesses of different identification technologies for these use cases (Section 6.2). Section 6.3 concludes with a brief discussion of two additional components that affect the practicability of consumer-oriented mobile services. The first is mobile network performance and coverage (such as in stores), which is relevant when retrieving product-related data; the second deals with the availability of appropriate sources for such data.

With respect to product identification technologies, we argue that bar code recognition has certain advantages compared to 2D optical code recognition, NFC technology, and general image recognition for many consumer-oriented retail applications and that the bar code recognition is likely to remain relevant in the foreseeable future.

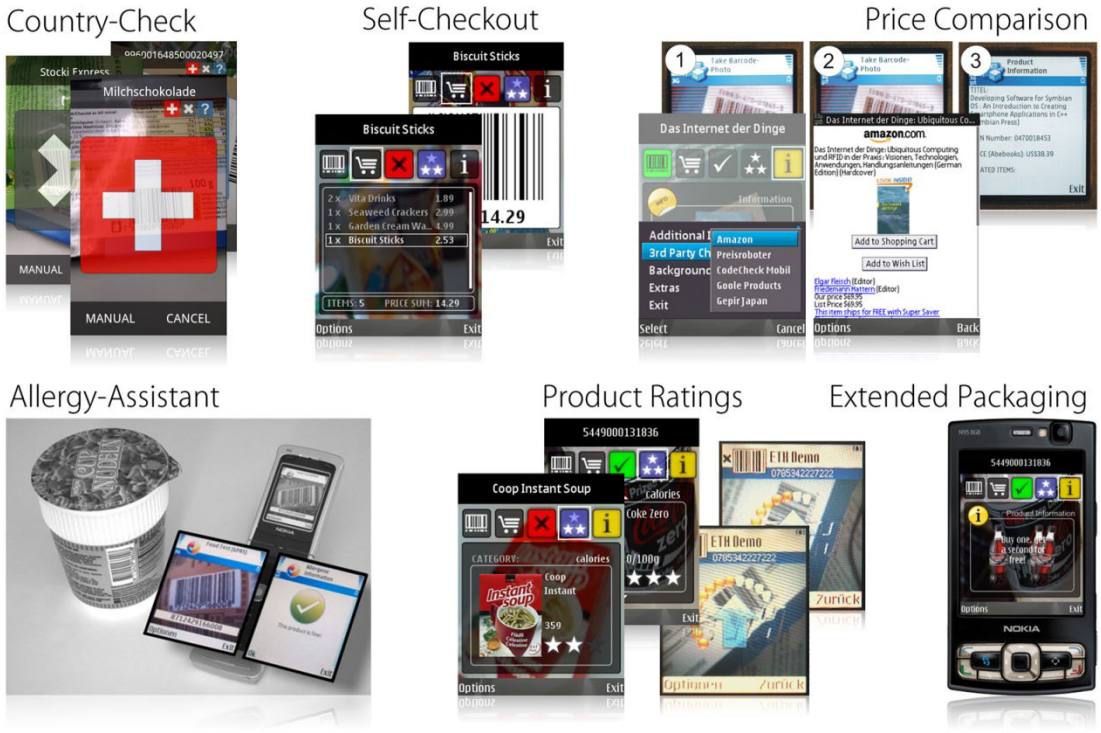


Figure 6.1 Screenshots from mobile phones illustrating some of the mobile services we prototypically implemented.

6.1 Application Scenarios

With the proliferation of smartphones that are capable of installing and running third-party applications in recent years, an increasing number of mobile applications have appeared that offer product-related services and information. Many of these applications would benefit from automated product recognition technology. To provide an overview of possible services, the following table includes brief descriptions of several consumer-oriented mobile services. Figure 6.1 shows screenshots of demonstrators that we implemented for some of these services.

Table 6.1 Example applications offering information and services to products:

Name	Description
Allergy-Assistant	After users defined a list containing all of the substances to which they are allergic (e.g., on the Web or directly on a mobile phone), the application tells them if a specific product is fine for them or not, after they scanned the product's bar code.
Price Comparison	Many online platforms offer up-to-date price information for specific sales items (e.g., electronics, books, or DVDs). After scanning a product's bar code, users can check directly in stores how much they would have to pay for the same product at another retailer.
Instant eBay-Auctions	When selling less valuable items, such as books or CDs on eBay, the required effort (e.g., manually creating a new eBay auction, writing a description of the item, and taking pictures) can easily outweigh the expected gain. After scanning a product's bar code with the "Instant eBay-Application", a description and images of the item are obtained from Amazon. Based on this information, a new eBay auction is generated automatically. Finally, the user is informed if the product is sold. This reduces the effort required when selling small items to scanning the item's bar code.
Product Ratings	Users can check online product ratings to obtain a second opinion in addition to the product infor-

	mation provided in the store and on the product's packaging. Ratings can be obtained from other stores (e.g., Amazon), or from user communities.
Inventory-Management	Even though a smartphone might never be able to compete with specialized identification equipment (e.g., professional, ruggedized laser scanners) in business environments, there are many use cases, in which smartphones suffice. They are capable of recognizing bar codes and can be used, for example, to scan and catalogue items in a warehouse or to generate an inventory of a user's private books or CDs.
Self-Checkout	Especially when buying only a few items, having to wait in line at the checkout can be cumbersome. Self-checkout applications allow users to scan the content of their shopping carts with their smartphones and check-out at automated terminals by means of their mobile phones (see the Metro MEA project [82]).
Extended Packaging	Space on the packaging of many items is limited for many reasons, including a small overall package size or legal regulations requiring certain information to be present. Especially with premium products, manufacturers would prefer to include more information about the product (e.g., how and where it was manufactured), dynamic content (e.g., videos), or interactive content (e.g., games or lotteries). Smartphones that are capable of recognizing products are well-suited platforms for delivering this additional content to consumers.
Country-Check	Users can check in which country a product was manufactured. Providing this information in a fast and transparent way to consumers enables them to make informed decisions while shopping. The same holds for other product-related data, e.g., information about specific product ingredients, such as genetically modified substances, which might be highly relevant for certain user groups.

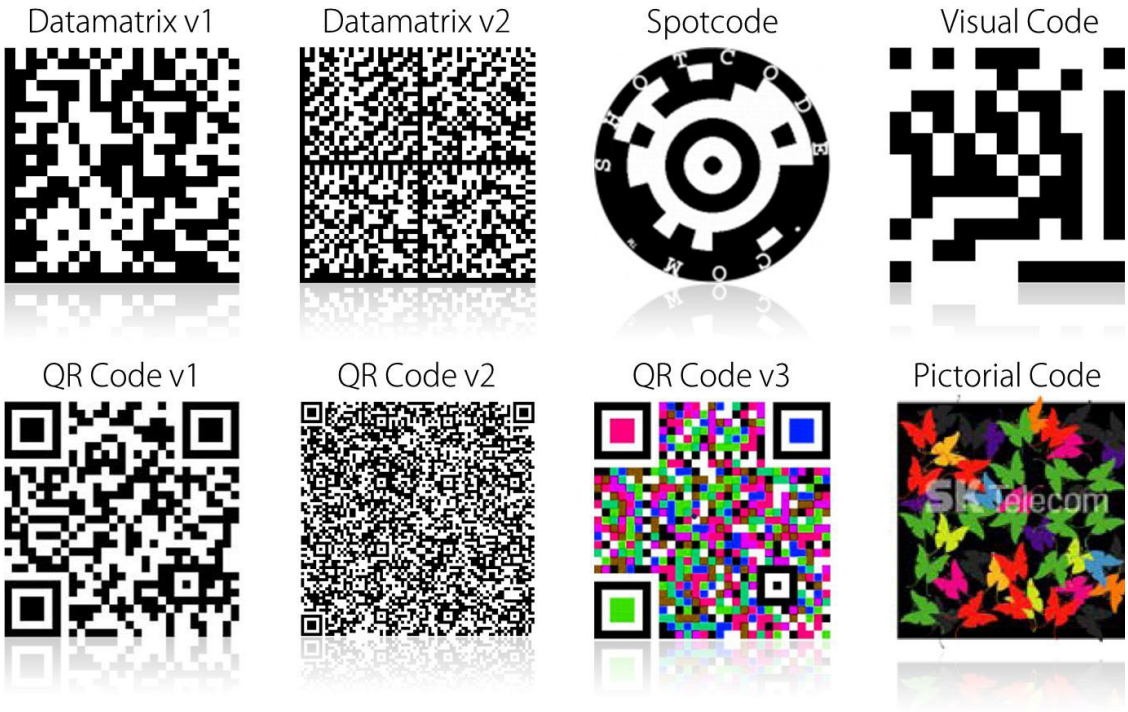


Figure 6.2 Examples of 2D code symbologies.

6.2 Product Identification Technologies

6.2.1 2D Codes

Compared to traditional bar codes, 2D codes also use the second dimension to encode data in patterns of varying size, shapes and colors. Depending upon the specific code symbology and pattern size, 2D codes have several advantages compared to bar codes:

- They can in general include more data than bar codes, especially if codes with many small features (e.g., black or white blocks) are used.
- Due to prominent guiding patterns present in many 2D codes, they are more robust against misalignments.
- If larger patterns are used, 2D codes are quite robust against low-quality and blurry images. Extensive built-in error correction mechanisms and error correction code words (e.g., in the case of the PDF417 code) further increase this robustness.

In particular the robustness against misalignments and low-quality, low-resolution images of certain 2D codes, for example, the Visual Code developed by Michael Rohs [10], allows for the optical recognition of 2D codes with older mobile phones. Therefore, many research projects used 2D codes to link real-world objects to virtual information [8, 83].

There is a wide variety of different 2D code symbologies available for different use cases and application scenarios (see Figure 6.2). Certain codes such as the VisualCode, ShotCode, or Pictorial Code[24] [84], were developed specifically with low-quality mobile phone cameras in mind, whereas others, like the Datamatrix or QR code, were also developed to improve data capacity for industrial applications [85]. Furthermore, certain codes provide additional features that enable new forms of interaction. For example, the VisualCode that features two prominent guiding bars and therefore allows for a robust detection of the camera's position in relation to a code. This information can be used to map arbitrary image coordinates to corresponding coordinates in the code's pane, which allows for example for the interaction with different elements printed on a magazine's page, while requiring only one code for the whole page [21]. The VisualCode developed by Rohs and the Pictorial Code from Tack-don Han are optical codes with limited data storage capabilities, but in turn their large patterns allow for the codes to be recognized even in the case of blurry images on fixed-focus camera phones.

QR codes [86] were developed in 1994 and have since then been widely used, especially in Japan. QR codes can be found on vending machines, in magazines, on the packaging of certain products, on street signs, and in various other places. This widespread adoption of QR codes is due to the fact that NTT DoCoMo [87] included QR code readers early on in many mobile handsets it sold. Standard QR codes have a maximum pattern size of 177x177 blocks and encode up to 2.953 bytes of data, which corresponds to 4.296 alphanumeric characters [88]. Whereas reading large QR codes, such as those printed on advertising billboards, is possible on devices without auto-focus cameras, smaller QR codes, particularly those containing a large amount of data, require auto-focus-enabled devices. Compared with other proprietary 2D code symbologies, QR codes have become an open international standard (ISO/IEC 18004 [89]) .

[24] The Pictorial code encodes 10 digits and features a matrix of 5x5 elements of varying shapes and four possible colors.

Figure 6.3 Application examples of 2D codes: posters (upper-left image), magazines (upper-right image), tagging of electronic devices (lower-left images), and for linking real-world devices to their Web-presence (lower-right image).

Despite the advantages of 2D codes, their use as universal identifiers for retail products is limited by the fact that bar codes are currently the standardized means for product identification. Bar codes are already present on virtually all items. Furthermore, hardware for scanning these codes is available throughout the supply chains world-wide. Therefore, the use of 2D codes is likely to be limited to application scenarios where either new codes have to be printed anyway or there are special requirements; for example, specific aesthetic considerations in magazines or advertisements, or in situations in which the encoding of more data is required than would fit in a standard bar code (e.g., for vouchers [90]). Increasingly, 2D codes are also being used to provide direct Web-links for objects and places, such as in the FavoritePlaces project launched by Google [91]. The latter encourages businesses to place an AR code in their windows. By scanning this code, customers can access additional information about the store (such as user reviews) or obtain coupons. Figure 6.3 illustrates some typical use cases of 2D codes.

While not the predominant choice for product identification, 2D codes and their many variants, such as 3D or 4D codes, which add dimensions via colors [92] or patterns that change over time (for example, on displays like in work presented by Langlotz and Bimber [93]), offer a number of advantages with respect to the optical recognition and remain a particularly attractive choice for many other application areas.

Figure 6.4 NFC/RFID tag and the Nokia 3220 device with NFC support (left image), and the Google Nexus S device with NFC support (right image).

6.2.2 Near Field Communication (NFC) Technology

Near Field Communication (NFC) is a technology that allows for the short-range exchange of data between a device and an NFC/RFID tag. Communication is based on inductive-coupling and generally requires no line-of-sight between the reading device and the tag [94]. In addition to early prototypical mobile phones featuring NFC technology, such as the Nokia 3220 (see Figure 6.4), recently an increasing number of consumer devices with NFC technology are appearing on the market (e.g., the Google Nexus S).

NFC/RFID technology has several advantages compared to the optical recognition of bar codes. Most notably, the recognition process itself can be automated. Instead of scanning each item in a large palette of products separately, in theory, all items can be scanned in parallel. This saves both time and labor in industrial environments. RFID technology has therefore advantages in domains in which fully automated, high-throughput product identification is required. However, regarding the identification of single items by a consumer, RFID technology also has major advantages compared to bar codes. RFID tags can store more data, and on some tags, this data can also be changed dynamically. In combination with the Electronic Product Code (EPC) [95], sales items can be assigned a world-wide unique identification number. In contrast to bar codes, this unique number allows not only for the identification of a product type, but also the identification and tracking of product instances. These unique properties of the NFC/RFID technology enable services not possible using traditional bar codes, such as anti-counterfeit measures [96, 97].

Despite the benefits of RFID technology, an item-level rollout, such as having an RFID tag on every single supermarket product, is still expected to be several years away. Current tag costs, as well as interference issues with water or on metallic surfaces, at least for cheaper tags, make the widespread appearance of RFID tags on everyday supermarket items relatively unlikely anytime soon. RFID tag use will likely be restricted to specialized application areas, in which these problems are not particularly prominent, such as in tagging library books, clothing, or higher-priced items. Mobile payment [98] and ticketing [99] also represent typical application scenarios for NFC-enabled mobile phones.

Compared to optical recognition technologies, NFC technology promises a better user experience due to its reduced line-of-sight requirements and lower susceptibility to dirty or damaged codes and environmental conditions, such as lighting. However, studies indicate that the user experience is comparable to the optical recognition of codes on mobile phones, at least with current

technology. The results of a study from O'Neill et al. [100] that compared the ease of scanning in the case of 2D codes and NFC tags indicate that untrained users preferred the optical codes, whereas trained users were faster when recognizing the NFC tags. Reischach et al. [6] also compared the speed and ease of scanning technologies in a user study. They concluded that bar code recognition on mobile phones could be conducted nearly as quickly and conveniently as scanning RFID tags. Using NFC to identify a product in this study took users on average 3.3 seconds, as opposed to an average of 5.5 seconds with bar code recognition and an early version of our recognition software [22] on a Nokia N95 device (see Figure 6.5). In our study reported in Section 5.1, scanning bar codes on an iPhone3GS or Android HTC Desire mobile phone took users on average 2 seconds with the recognition engine presented in this thesis. However, results for NFC may change in the future with advancements in NFC technology itself and its improved integration into mobile devices.

6.2.3 Image Recognition

General image recognition techniques that do not rely on special markers attached to products, but instead identify a product based directly on one or multiple images are a promising technology. Various recognition methods have been proposed [101-103]. Commercial implementations include Google Goggles [76] and services provided by Kooaba [104].

One advantage of this technology is that products not featuring bar codes have the potential to be recognized. This enables, for example, post-sale services to products for which the bar codes are no longer available, e.g., because they were located on the packaging, or services for products in stores with missing bar codes.[25] However, for our specific use case of recognizing retail products, general image recognition technology has also several drawbacks:

- Identification of the enormous number of products available requires a large database of product images or product feature sets, which must be created and maintained. This process is currently not feasible for all products sold worldwide.

[25] Retailers sometimes remove bar codes from products in the sales floor in order to hinder comparability, e.g., in case of electronics.

- If present, such databases must be stored somewhere and searched in a reasonable time. Such tasks that are in general not feasible directly on mobile phones and therefore require a server-based approach, or combined approaches, such as those described in [105], in which objects are tracked on mobile devices but the object identification is performed on a remote server.
- The variety of products available combined with the degree of freedom users have in taking images of a product pose a serious challenge: Users can vary the angle or distance from which an image is taken, the lighting may differ, and other products on the shelf may appear in the background, to name but a few issues.

In spite of these challenges, which prevent the use of image recognition in identifying a wide range of general products on mobile phones, the technology offers a great deal of potential and advantages in situations in which the set of objects to be recognized can be limited and the degrees of freedom in taking images of the product can be reduced. Such situations encompass the recognition of book covers, the imprints on wine bottles, the facade of famous buildings, or movie posters.

6.2.4 Manual Code Entry

The most straightforward method by which to identify a product is to manually enter a product's code or name, particular as this method is available on all mobile phones, regardless of their capabilities. Reischach et al. investigated the advantages of automatic product identification versus manual product identification in a user study with 17 participants [6]. The means of product identification considered included:

- Manually entering a product's EAN13 bar code number
- Manually entering a search term (such as the product name)
- Optical bar code recognition
- NFC tag scanning
- EPC[26] tag scanning

[26] The Electronic Product Code (EPC) standard allows for RFID tags to be identified using UHF signals (860-960 MHz) from a distance of up to 5 m and is mainly intended for supply chain applications. The Nokia prototype E61i, featuring an integrated EPC UHF reader, was used for the study.

Results indicate that automatic recognition technologies have a major advantage over manual code entry or search term entry in terms of perceived ease-of-use and identification speed (see Figure 6.5). Scanning NFC tags turned out to be nearly eight times faster than the manual search term entry. Manually entering a 13-digit bar code took participants around 14.4 seconds. These findings indicate that while manual code or search term entry remains an important fall-back solution for phones without a camera or situations in which the automated identification fails, e.g., due to lacking or damaged labels, it is not an alternative equivalent to the automatic identification of products.

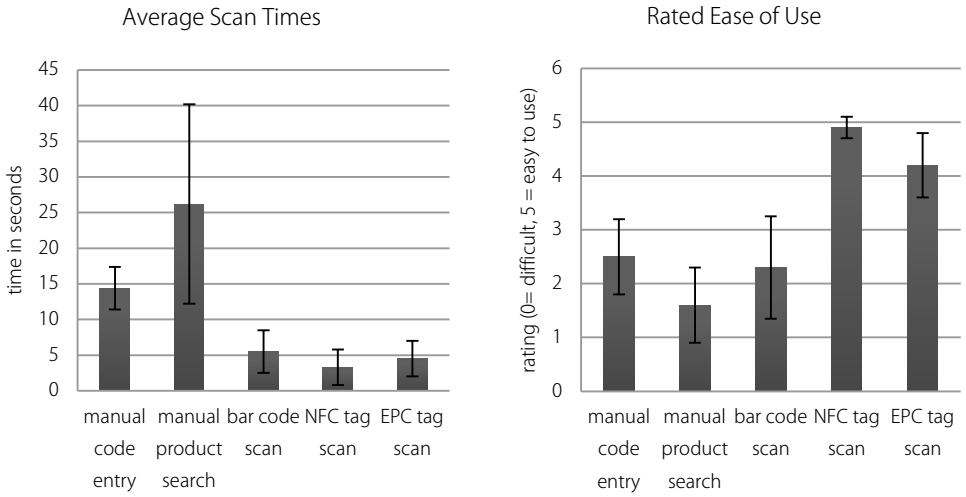


Figure 6.5 Average required scan time and rated ease-of-use for different interaction methods as determined by Reischach et al. [6].

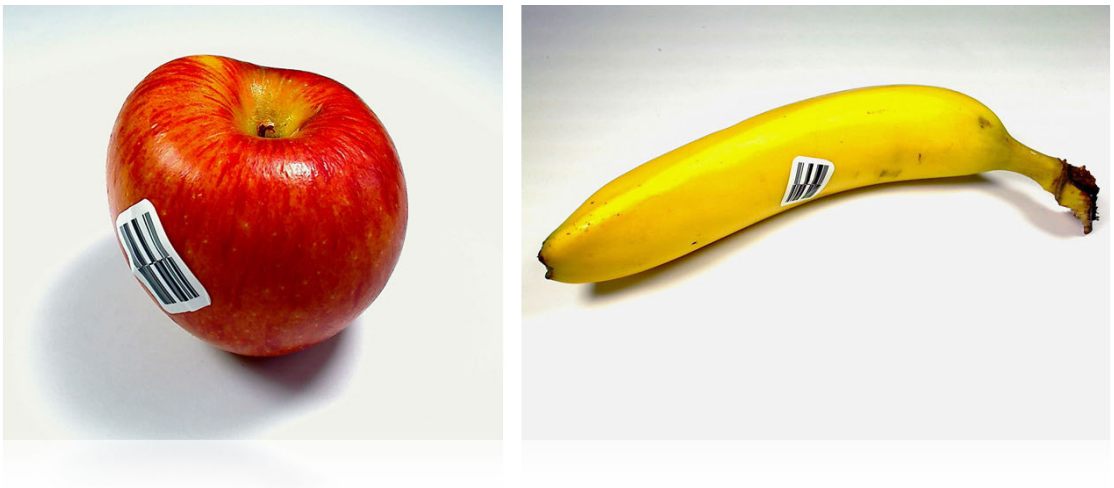


Figure 6.6 Examples of DataBar codes on fruit.

6.2.5 Bar Codes

Bar codes were introduced in the retail industry in 1974 [106] as a means of automatically identifying products. Today, they represent the standard method of uniquely identifying merchandise items worldwide. The bar code standard and the process of issuing unique numbers is managed by GS1 [107]. Although different bar code symbologies are available, the EAN/UPC code family is the one most widely used on products (see Figure 2.5).

The strongest argument for using bar codes to identify products by means of mobile phones is the fact that they are already an established standard and available on most products. A complete replacement of bar codes as the primary means of product identification, e.g., with RFID or 2D codes, remains unlikely for the foreseeable future. The requirement of replacing existing scanning hardware used in the industry in such a case is a substantial argument on its own. Furthermore, in 2010, GS1 introduced the successor of the codes that are currently used, namely the GS1 DataBar[27] code [108]. This code is another bar code and is intended to supplement codes that are currently used, specifically in areas in which existing EAN/UPC codes have their limitations [106] and until technologies such as RFID are ready for an item-level tagging:

- *Small and hard-to-mark goods*: DataBar codes can be printed smaller and offer the option of being printed in a stacked form with two bar codes on top of one another, which is more suitable for round surfaces. Due to this flexibility, items can be tagged that are currently not labeled, such as fruits, making the manual weighting process obsolete (see Figure 6.6).
- *Couponing*: DataBar codes can store a larger number of digits and therefore enable additional services, such as couponing.[28] [29] [109]
- *Variable measure products*: Pilot tests are being conducted with variable measure products, such as meat, seafood, or cheese. In such cases,

[27] Formerly referred to as Reduced Space Symbology (RSS)

[28] "*In the US, coupons are a significant business. Unfortunately, it has been limited by constraints around old coupon guidelines and bar code structures. GS1 DataBar will revitalize and provide the potential to greatly improve the coupon industry.*" Doug Naal, Kraft [163].

[29] Since January 2011, all manufacturer coupons in the United States and Canada have transitioned to the GS1 DataBar symbology [108].

information such as "best before date", the country of origin, or product weight can be encoded in the bar code.

New scanning equipment sold in the recent years is already able to scan DataBar codes, and existing systems have been upgraded where possible [110]. Our recognition method can be extended easily for the recognition of such DataBar codes.

The main drawback of bar codes, compared with 2D codes and NFC, is the limited amount of information that can be encoded. Bar codes can identify product types, but not single items. While this limitation excludes certain services, such as anti-counterfeiting, being able to identify the product type is sufficient for most mobile services presented in Section 6.1. In spite of their limitations, bar codes represent a good solution for product identification in many consumer goods applications today and for the foreseeable future.

6.3 Practicability of Mobile Services

Aside from product identification, two other aspects influence the practicability of mobile services, namely access to and the availability of product-related data.

6.3.1 Data Access via Cellular Networks

After product recognition, many mobile services rely on remote access to product-related information. In practice, information might be stored on the mobile device itself or can be locally accessed using WLAN or Bluetooth infrastructures where available. However, the most prominent technology for enabling a ubiquitous connectivity in the case of mobile phones are cellular networks. GSM (Global System for Mobile Communication) and EDGE (Enhanced Data Rates for GSM Evolution) networks are already deployed in most countries worldwide[30] [111]. Faster 3G technology networks, such as HSPA (High Speed Packet Access) networks, are also increasingly available[31] [112], and the next generation LTE (Long Term Evolution) already approaches[32] [113]. Access to product-related data and services requires sufficient network

[30] As of January 2011, 531 EDGE networks exist in 196 countries [111].

[31] In January 2011, 416 HSPA networks were launched in 161 countries [112].

[32] 128 LTE network deployments are planned or in progress in 52 countries, including 17 systems that have already been launched commercially [114].

bandwidth, short communication delays, and network coverage in relevant places, such as stores.

Bandwidth: While data rates on GPRS (General Packet Radio Service) networks (up to 114 kBit/s) and EDGE networks (up to 384 kBit/s) are already sufficient for many services that require only a limited amount of data to be transmitted to the mobile phone, the data rates of 3G (such as UMTS, HSPA, HSPA+) or 4G (LTE) networks are even sufficient for multi-media content streaming. Already 65% of HSPA networks support a 7.2 MBit/s or higher peak downlink capability, and even the HSPA+ networks (HSPA Evolution, commonly referred to as HSPA+) that support up to 42 MBit/s peak downlink data speed are market ready. Some operators are already preparing to introduce the next evolutionary step at 84 MBit/s.[33] Targets for LTE include peak downlink data rates of at least 100 MBit/s and uplink peak data rates of 50 MBit/s [114].

Table 6.2 Measured round-trip-times in 2007 of cellular networks in milliseconds based on 1000 runs:

	GPRS	HSDPA
Average	383.2 ms	215.2 ms
Minimum	186 ms	171 ms
Maximum	9429 ms	1844 ms

Latency: Generally even more critical than bandwidth are network delays. According to results from Stuckmann, Ehlers, and Wouters [115] from 2002, with the TCP protocol one can expect round-trip times of 500ms – 2s for GPRS/EDGE networks and 100ms-200ms for UMTS/HSDPA networks. However, latency on mobile phone networks is influenced by several factors that are difficult to predict, such as the network operator, mobile phone model, and the quality of reception in certain locations. Reliable results are therefore hard to produce. We performed our own spot test by measuring the round-trip-times between a client application written in J2ME on a Nokia N95 mobile phone and a Java-based server on a regular PC, both at the same location in 2007. Table 6.2 lists the measured times for messages sent on a GPRS and

[33]:"*HSPA+ is now mainstream. This has been achieved in only 23 months since the first HSPA+ system was commercially launched. The recent trend was for operators introducing mobile broadband services to launch with HSPA. Most operators entering the market today are going straight to HSPA+.*", Alan Hadden, President of the GSA [112].

an HSDPA network. Despite the lack of reliability in such measurements, the results indicate, that for mobile services that require data only once after a product has been scanned, the delays currently found on mobile phone networks are already within an acceptable range. Furthermore, the situation is set to improve with technologies like LTE that promise delays below 5ms [116].

Coverage: As of 2011, the network coverage of most mobile phone operators in densely populated areas can be considered to be sufficient. GSM coverage maps provided by Mobile World Live [117] show nearly 100% coverage in urban areas in Europe and the United States. Furthermore. tests we performed in 2011 in various stores in Switzerland and Germany indicated a good mobile phone reception in most locations.

6.3.2 Availability of Product-Related Information

With a robust and fast identification technology, such as the bar code recognition, and sufficiently fast cellular networks, most services presented in Section 6.1 can be realized from a technical point of view. However, one central component of these services has not been addressed so far: The availability of product-related data.

Whereas some services listed in Table 6.1 do not require previously available data about products to be useful (e.g., the Inventory-Management application) or only information about a limited set of products (e.g., price information about the products sold in a specific store for self-checkout), many consumer-oriented applications require data about a large number of products (e.g., the Allergy-Assistant or Product-Rating services). Users are likely to lose interest if the service they use provides information only for every tenth product they scan. In addition, services, such as the Allergy-Assistant, require not only detailed information about ingredients for a large number of products, but this information should be reliable, up-to-date, and comprehensive. In the following, we list different sources of product-related information as they are available of today.

One possible source of information about products are companies, such as Amazon [118], BestBuy [119], Google [120], and others that offer already Web-based, product-related services, including shopping, price comparisons, and product ratings. This information is in general available for a large set of products, up-to-date, and available to third-party developers. While access to this data is often free for prototype creation and testing, commercial use of the provided data is regulated, and there are some forms of legal or technical

limitations in place, e.g., the maximum amount of requests to the Web-services offered is limited to a certain number of requests per day.

In addition to the aforementioned companies, there are numerous associations and consumer watch groups that have product-related data. Examples include the Alliance for Justice (AFJ) [121], Consumer Action [122], MaxHavelar [123], Ktipp [124], ÖkoTest [125], and others. While this data is usually available only for a limited set of products and is not always accessible online, the information, such as in-depth test reports, is in general quite detailed and highly relevant for certain user groups.

Another option to obtain product-related data involves community- and crowd sourcing-based approaches. There are several projects that allow users to enter and manage product-related data, e.g., WikiFood [126] and Codecheck [127]. While these projects are a potential source of otherwise unavailable knowledge, such as user comments and personal opinions, the information usually does not cover all available products and might be too unreliable for some applications (e.g., the Allergy-Check application).

Traditionally, product-related data that is relevant throughout the supply chain has been managed and provided in the industry by data pools, such as SINFOS, 1SYNC, iTrageNetwork. GXS, and others. Information about these sources as well as a complete list of currently certified data pools can be found in [128]. These data pools include information like package size and product weight. For many food items, there is also a list of ingredients available. To allow for a centralized access to this information, the data pools are synchronized and connected with the Global Data Synchronization Network (GDSN) provided by GS1 [129]. However, even though these data pools contain detailed information in certain product categories, the information is in general not freely available or complete, given the abundance of items sold in typical stores.

6.4 Summary

There is a large variety of product-related mobile applications imaginable that enable consumers to access information and services to retail products. In contrast to traditional, product-related websites consumed in front of a stationary personal computer, many of these applications are most useful in a mobile scenario, such as when people are standing in a supermarket aisle and considering buying a product. In such a scenario, a streamlined user interaction, including a fast and convenient way to identify products, is essential.

In this chapter, we presented examples of consumer-oriented mobile services and discussed relevant product identification technologies. Besides the recognition of bar codes, we covered the optical recognition of 2D codes, RFID/NFC technology, general image recognition, as well as the manual input of product numbers or names. While each technology has its use cases and many have specific advantages compared to the recognition of bar codes, bar code recognition technology is especially suited for enabling mobile services today and in the foreseeable future. All components required for such services are already available. Most sales items have bar codes printed on them, many users have camera-equipped mobile phones, and mobile data plans are becoming more common. Furthermore, using the existing bar code numbers as primary means for product identification has the advantage that already existing, product-related information can be leveraged.

7 Rapid Prototyping of Mobile Services

Over the past few years, mobile phones have evolved into attractive platforms for novel types of applications. However, compared to the design and prototyping of desktop software, mobile phone development still requires programmers to have a high level of expertise in both phone architectures and low-level programming languages. In this chapter, we analyze common difficulties in mobile phone programming and present SPARK, a rapid prototyping environment for Symbian smartphones that allows non-experts to create advanced mobile services in a fast and easy way. We also present the results of two case studies in which SPARK has been used: a graduate course on distributed systems in which 73 students used SPARK to develop mobile applications, and the development of a mobile phone-based product information platform. Information about SPARK contained in this chapter was already published in [130].

7.1 Motivation

Mobile phones are attractive development platforms: They are ubiquitous, highly mobile, provide significant computing power, and increasingly also offer an abundance of built-in sensors. Concrete projects use mobile camera phones to recognize bar codes or 2D codes and offer services to retail products, such as those presented in the previous Chapter 6, or use the phone's built-in GSM, GPS or WLAN modules for location-based services [131]. Others perform "reality mining" through Bluetooth sightings in order to map social networks and everyday activities [132] or generate noise-maps of environments with the help of the phone's built-in microphone [133, 134].

Despite today's abundance of feature-rich mobile phone hardware and powerful software platforms, creating applications that leverage the platforms' potential is still a time-consuming process that challenges non-expert developers by requiring in-depth know-how [135]. This is especially true for applications that require full control of the device and its sensors. JavaME [136], the Java runtime available on many mobile phones, is, due to its limited APIs, severely restricted when it comes to full phone control. This forces

many application designers to delve into low-level programming languages, such as C++ Symbian [137] or Objective C [138]. While a number of scripting languages are available for mobile phones (e.g., Lua [139], Ruby [140], Flash-Lite [141] or Hecl [142]), they come with drawbacks. Either they are still in the early development stages (e.g., Lua and Ruby), run on top of the phone's Java runtime and thus fare no better than Java ME (e.g., Hecl), or allow only limited device control (e.g., FlashLite).

One notable exception is the Nokia-initiated Python language for S60 (PyS60) [143, 144], as it builds on an easy-to-learn scripting language, is implemented in native Symbian C++ and therefore offers direct access to most available phone functions. Furthermore, it is extensible through C++ modules. However, PyS60 development is not without difficulties. Its Symbian heritage requires programmers to be comfortable with the typical Symbian development process, e.g., how to package an application for distribution or sign an application in order to gain access to sensors like GPS. The latter requires knowledge about how to obtain the appropriate certificates and of the complex Symbian Signed [71] process. Last but not least, any application development on Symbian phones, be it low-level, Java-based, or scripted, faces additional challenges when it comes to general development issues: Developers must repeatedly upload, debug, and update their software on the actual devices, a process that is frequently time-consuming and fraught with errors on Symbian devices. Furthermore, they have to ensure that the software runs on different device types or update already deployed applications, e.g., in order to fix bugs.

The goal of our system is to provide an easy and fast to use development environment for mobile phone applications, particularly for developers unfamiliar with mobile phone programming. This is achieved by providing a rapid prototyping environment that leverages the strengths of the existing PyS60 eco system and systematically addresses the problems of mobile phone prototype development.

We structured the rest of this chapter as follows: Section 7.2 describes the remaining difficulties with the PyS60 development process in particular and of application design for mobile phones in general. Drawing from these challenges, Section 7.3 then presents the SPARK architecture and its implementation. Section 7.4 compares SPARK with existing solutions and discusses alternative application development options. SPARK has been used in a number of our research projects, in teaching, and in several larger projects with industry partners. Sections 7.5 and 7.6 discuss two example case studies: a student

class on distributed systems and the "Product Advisor", a customizable platform for mobile services.

7.2 Challenges

Based on several PyS60 projects conducted in the past, we identified a number of general areas that make the application development process challenging. These areas are especially problematic for beginners, who face a steep learning curve even when using PyS60 instead of C++ Symbian. Further issues arise in application scenarios that target large user deployments and frequent code updates, e.g., as part of an early adopter rollout or during long-term user studies.

7.2.1 Beginners' Challenges

When teaching colleagues and students on how to use PyS60, we noted three main obstacles for beginners: application signing, application packaging, and setting up a working development environment:

7.2.1.1 Application Signing

With Symbian OS v9, Symbian introduced a security system that is based on application capabilities[34] and certificates [71]. Gaining access to certain features (e.g., the GPS module or GSM cell information) requires users to choose or compile the Python interpreter and PyS60 modules with the required capabilities and sign them with a certificate covering these capabilities. Getting to know the complex "Symbian Signed"-processes, obtaining software modules that have been compiled with the appropriate capabilities, as well as obtaining certificates that grant more capabilities, is a complicated and time consuming task.

7.2.1.2 Application Packaging

To distribute an application on multiple devices, all application files must be packaged as a Symbian Installation System (SIS)[35] file. This requires not only

[34] Capabilities control access to various sensitive features of the phone (17 in total), for example, the user's current location. If a developer wants a PyS60 program to query the current location, its calling shell must have been registered with this "capability" and subsequently digitally signed with a capability-enabling certificate. Some capabilities can be "self-signed", others can only be signed by the phone manufacturer.

[35] SIS files package application files for installation.

performing the above-mentioned application signing but also choosing a matching application identifier (which must be in a particular range based on the certificate used for signing), building the SIS file, and providing an appropriate application icon in a special SVGT format [145].

7.2.1.3 Development Environment Setup

Compared to other programming alternatives (especially C++ Symbian), PyS60 drastically simplifies the process of programming. However, even though PyS60 comes with a range of tools to support development, getting to know what components and tools are required and available for what kind of Symbian version (e.g., the 2nd and 3rd Symbian S60 editions differ significantly), as well as setting them up, is non-trivial and may differ across desktop operating systems. Preparing, for example, a homework assignment involving PyS60 that covers all possible combinations of student hardware and software is quite time-consuming.

7.2.2 General Challenges

Even after users became familiar with PyS60 programming, three general issues continued to make the application development process tedious: the need for on-device testing, the variety of available devices, and software update management:

7.2.2.1 On-Device Testing

Emulators are no substitution for on-device testing of applications. They do not behave exactly like the real hardware (e.g., regarding stack size, handling of low-memory situation or timing), and usually do not support all features (e.g., camera support). This requires the developer to ultimately test the application on actual devices, including the time-consuming task of copying and executing all updated application files on the phone, which results in long and tedious “code, edit, execute”-cycles.

7.2.2.2 Multi-Device Support

Despite common programming platforms, such as Symbian S60 or even Java ME, every phone model is different. Therefore, mobile phone applications need to be tested practically on every single device (including different firmware revisions) on which they will be deployed to ensure proper operation. Given the above-mentioned lengthy “code, edit, execute”-cycles, testing each program modification on all supported devices is a time-consuming task.

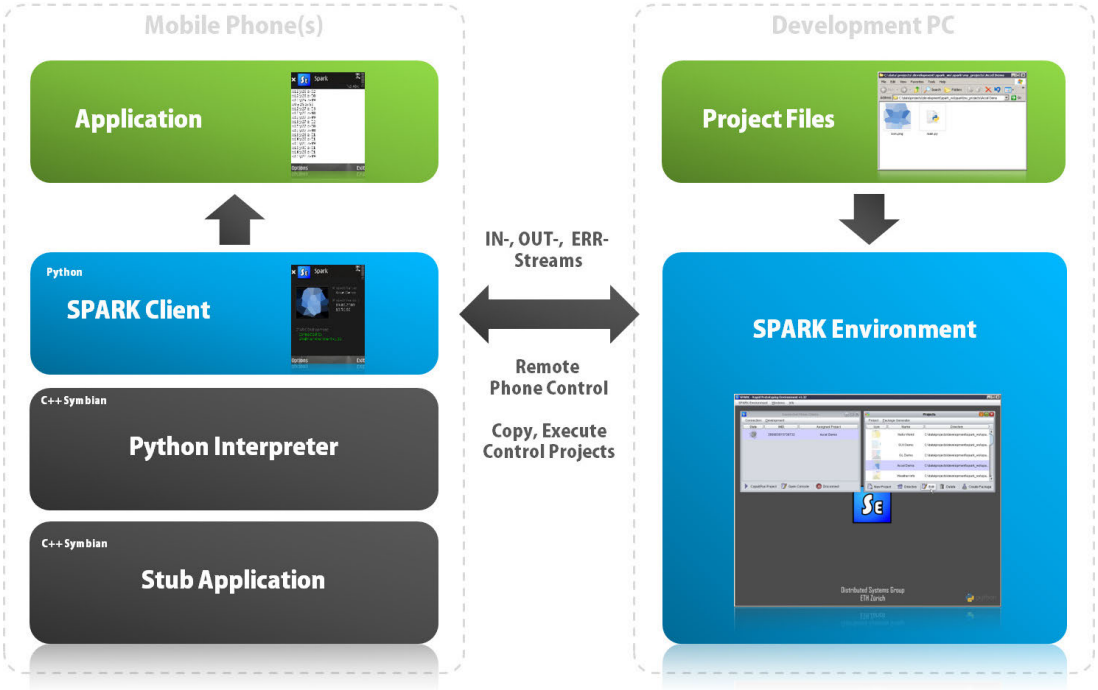


Figure 7.1 SPARK system architecture consisting of a client application running on mobile phones (SPARK Client) and a development environment (SPARK Environment) on a PC.

7.2.2.3 Remote Application Updates

Large-scale and long-term application deployments invariably come to the point when critical updates or missing features need to be rolled-out to all handsets. A deployment such as the Metro Mobile Shopping (MEA) application [82] used in the Metro future store, which allows users to scan products with their phone to perform self-checkout, has 100 registered users and around 30 in-store devices. Fixing a bug or adding new features requires all users to bring in their mobile phone for servicing, as typical users would not be able to easily install a provided SIS file on their mobile phones. Developers would be equally challenged by manually updating over 100 devices with a new software version.

7.3 Rapid Prototyping with SPARK

In the following, we will present the SPARK architecture and describe its implementation. SPARK specifically focuses on removing the above-mentioned obstacles from PyS60 programming. It lowers the entry barrier for programmers unfamiliar with mobile phone architectures and simplifies the general development cycles in order to support the development and deployment of large-scale, real-world applications.

7.3.1 General Architecture

The SPARK rapid prototyping environment is an OS-independent software that facilitates the simple and fast creation, testing, and deployment of arbitrary PyS60 applications. It builds on the available PyS60 resources [143] and extends these with features that address the six previously identified problems (Section 7.2). The environment comes in an easy-to-install single package and requires no prior knowledge about mobile phone programming or PyS60. With only a basic knowledge of Python, developers can immediately start creating applications. Specifically, the design of SPARK followed the four principles below:

1. *Low entry barrier:* Application creation should require no prior knowledge of mobile phone programming or PyS60, as well as minimum time and effort.
2. *No application restrictions:* No restrictions should be placed on the type of applications that can be created.
3. *Quick application creation:* Setup should be fast, "code, edit, execute"-cycles quick and application deployment simple.
4. *Ease of use:* Non-experts should be able to create powerful applications by abstracting from complex tasks.

As shown in Figure 7.1, the SPARK system consists of two parts: A mobile phone application (SPARK client) and a desktop Java application (SPARK environment). Each mobile phone application is represented as a *project* in the SPARK environment. A project is simply a directory on the user's PC that contains all application files. Application files include typically one or more Python source code files and additional resources, such as images or sound files.

The SPARK environment allows users to manage (create, copy, and delete) projects, install and execute projects on connected phones, and to package projects into distributable SIS files. The SPARK clients act as containers on the mobile phones that provide a connected SPARK environment with remote access to the device. Multiple SPARK clients (and therefore multiple mobile phones) can be connected to a single SPARK environment (see Figure 7.2). SPARK supports USB, Bluetooth, WLAN, and cellular connections.

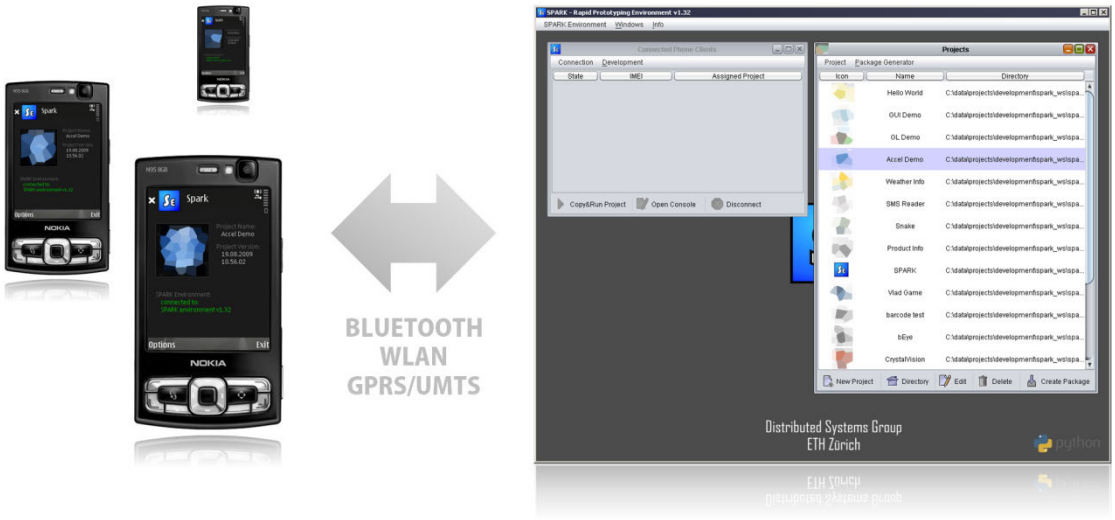


Figure 7.2 Multiple mobile phones with the SPARK client software can be connected to a SPARK environment instance.

To allow new users to quickly begin with development, the installation process has been designed to be simple: A prepackaged SIS file must be installed on each development phone, and a single installer file (e.g., EXE for PCs or an OS-independent JAR) needs to be installed on the desktop computer. The SPARK client's SIS file contains all resources needed for PyS60 development on Symbian 9 2nd and 3rd edition smartphones, such as the Nokia N73, N95/8g, or business phones like the E61, and thus requires no additional software. Specifically, the SPARK client SIS file contains the following components:

- C++ Symbian stub application starting the PyS60 interpreter
- PyS60 interpreter
- SPARK client application written in PyS60
- Additional C++ Symbian modules for common tasks not covered by the used PyS60 implementation (version 1.4.5), including Bluetooth scanning, XML parsing, or a module for the recognition of bar codes.

The SPARK environment installer contains the components listed below:

- A Java Runtime Environment (JRE 6)
- The SPARK environment application written in Java
- A collection of example programs and useful links to PyS60 documentation

7.3.2 SPARK Environment

The SPARK environment provides several services that apply to the currently connected mobile phones (SPARK clients) and the list of managed projects. For each connected phone, users can open a window providing remote access to this device. A remote control window displays basic information about the phone, such as the phone model and IMEI[36] number, provides a remote Python console with syntax highlighting, and allows the execution of selected projects on the phone by simply pressing a button (see Figure 7.3). The remote console provides users with a direct and interactive way to explore PyS60, test code found on the Web, e.g., by simply pasting it into the console, or to inspect parts of running applications.

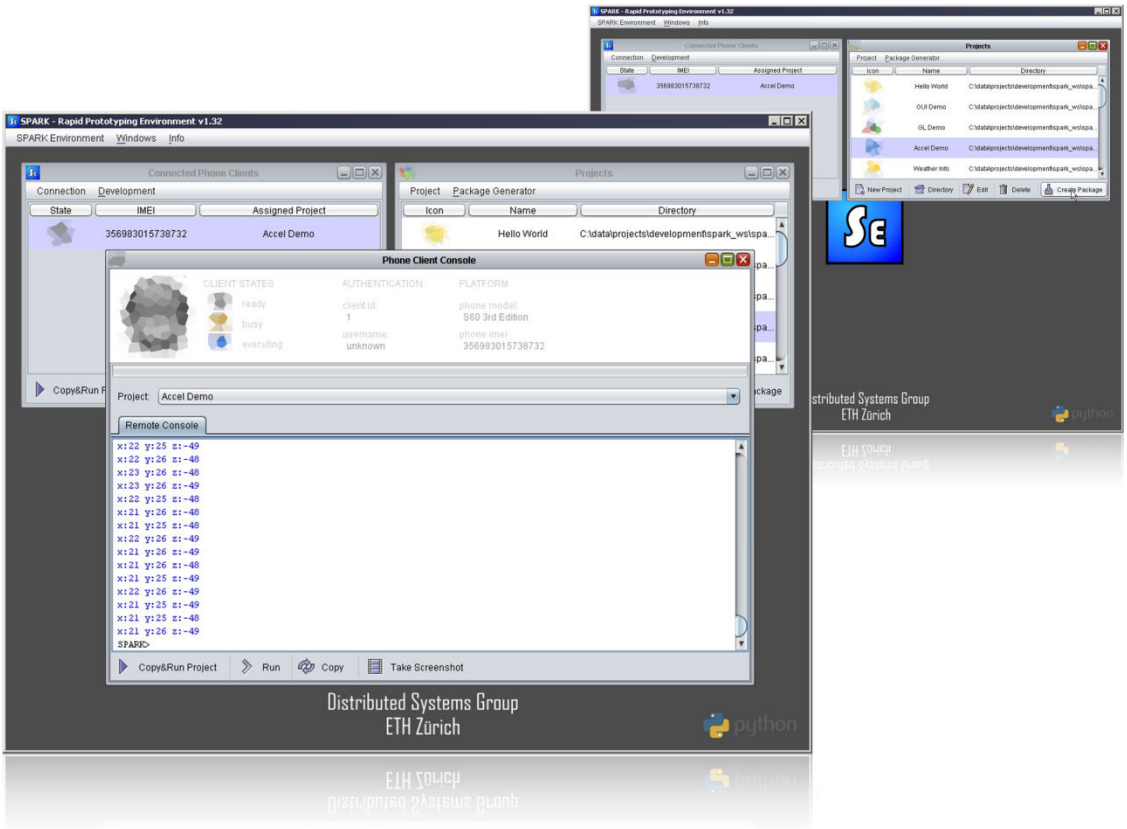


Figure 7.3 Screenshots of the SPARK environment showing three open windows: The left window shows all currently connected mobile phones, the right the list of projects, and the middle the remote control window for a connected phone displaying the current program output.

[36] The IMEI "International Mobile Equipment Identity" number is a unique number for every mobile handset sold world-wide.

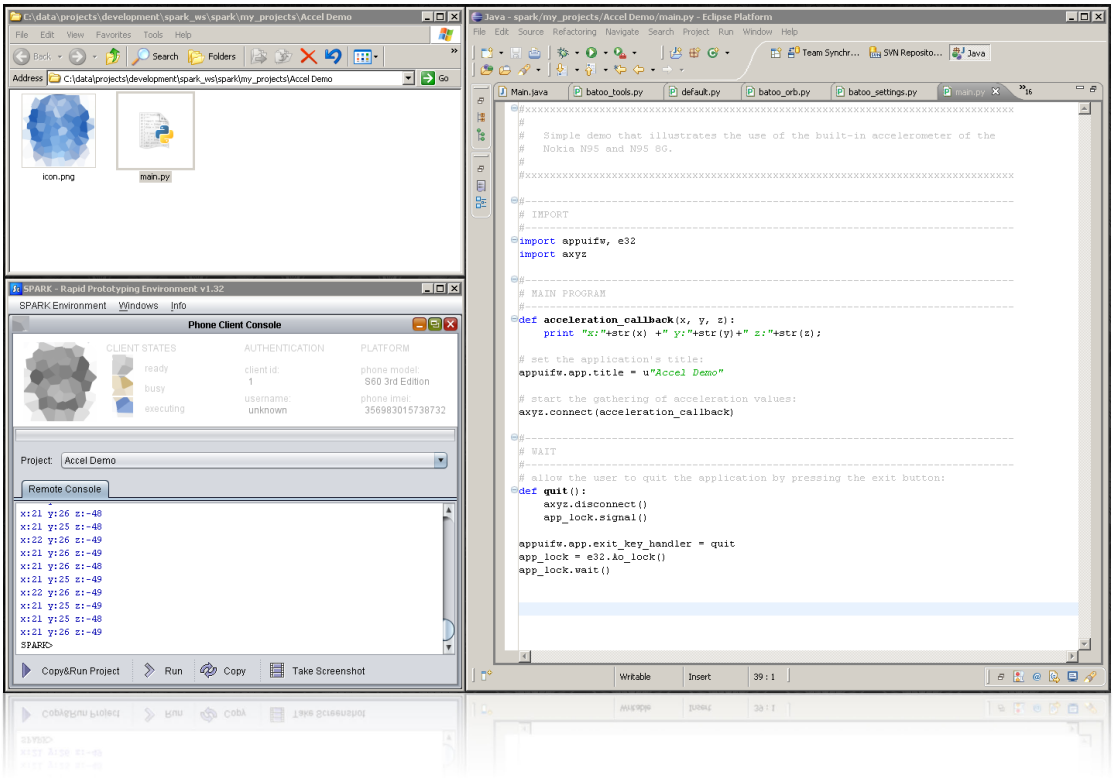


Figure 7.4 Typical development setup with a maximized remote control window for a connected mobile phone (lower left), the project files (upper left), and a Python editor showing the source code of the application (right).

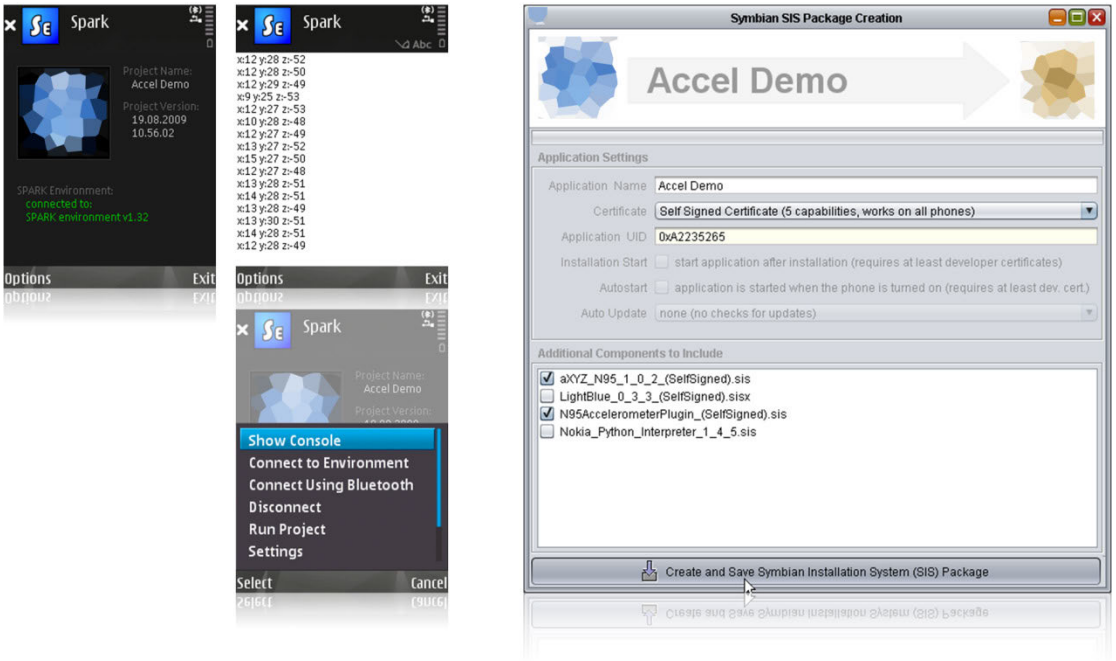


Figure 7.5 Screenshots from the SPARK client application (left three images) and the "application packaging" dialog (right image).

If multiple, potentially different mobile phones are connected to the SPARK environment, users can execute projects in parallel on multiple selected phones and monitor their real-time output, allowing for the simple detection and correction of application problems occurring on some phone models.

Creating new projects is supported by a wizard dialog, allowing users, for example, to use application templates as a starting point for their own applications. Available templates can easily be changed or extended, e.g., with user generated ones. Project management in the SPARK environment is very lightweight. The environment contains no internal Python editor or file manager and allows users therefore to choose their preferred tools instead. The SPARK environment simply keeps track of the main project directories, offering various actions for each of them. Figure 7.4 shows a typical development workspace, with a file manager listing the project files, the SPARK environment with a maximized remote console for a connected phone, and a third-party Python editor for source code modifications. A typical "code, edit, test"-cycle is as follows: The user makes some changes to one or more project files, e.g., code or images. After saving the changes, she presses the "Copy & Run" button on the SPARK remote console window. The SPARK environment will determine what files have changed, copy only the changed files to the mobile phone, and then run the updated project through the SPARK client. The application's output can then be monitored on the remote console. This allows for agile development, as saving changes and then pressing the "Copy & Run"-button is all it takes. The following automatic process of compiling and starting an application on the phone is usually also fast, e.g., on the N95 8g less than a second for smaller projects and only a few seconds for large applications.

Once an application is ready to be deployed, i.e., installed as a standalone application, the "application packaging" dialog (see Figure 7.5) provides a simple way of doing this. It allows for the generation of SIS files that contain all necessary project files, in addition to any software that might be required for the application to run, such as the Python interpreter or required C++ extension modules. The SPARK environment also greatly simplifies the complex application signing process by allowing users to choose between two different application signing options: The "SelfSigned" option allows the software to be installed on all devices, but with only limited capabilities, while the "DeveloperCertificates" option can grant more rights, but (depending on the actual certificate type) this usually limits application deployment to mobile phones with registered IMEI numbers. Depending on the user's choice, unique application identifiers (UIDs) from the correct range are generated automatically

and the correct versions of all extension modules available for inclusion in the final SIS package are selected. Furthermore, a code parser inspects the project's Python source code files and recommends the inclusion of required extension modules in the final SIS package. Miscellaneous tasks, such as scaling the application's icon to the required size and converting it to a special subset of the SVGT format for S60 devices, are also automated.

The auto-update mechanism of SPARK is targeted specifically at large-scale deployments. Developers can configure the generated standalone mobile phone applications in such a way that they will periodically monitor for changes in their original project files and automatically update themselves without the need for end-user interaction. This allows for subsequent changes to already deployed applications.

The SPARK environment has been designed in an open and extensible way, allowing developers to easily add their own application templates, Python extension modules, or signing options (including custom certificates). The SPARK environment also uses external software: The BlueCove Bluetooth stack [146] for Bluetooth communication and the Ensymble [147] tool for building SIS files. Created SIS files can be distributed and installed like any other mobile phone application – they do not require any additional software to be present on the phone prior to installation.

7.3.3 SPARK Client

The SPARK client software allows the SPARK environment to remotely access a device. It is a stand-alone application written in PyS60 and acts as a container for projects that should be executed on the mobile phone.

Figure 7.5 depicts screenshots from the SPARK client application showing the main screen, the options dialog, and a console window. The main screen displays information about the currently contained project, such as the project's name, icon, and version information, and indicates whether the SPARK client is currently connected to a SPARK environment instance. Connections to an environment can be established using Bluetooth, WiFi, or cellular networks. While cellular network connections also tend to be responsive enough for many tasks, we have found Bluetooth to provide the shortest delays when working with the interactive console. Standard output and error messages from executed applications can be viewed not only remotely on an SPARK environment, but also locally on the phone in a console window. Once a project has been copied to a SPARK client, the project can also be started directly from the mobile phone without the need for a connected SPARK environment.

7.4 Related Work

A number of projects have recognized the need for providing rapid prototyping tools for mobile phone development. In this section, we will cover prior work on rapid prototyping platforms for mobile phones (Section 7.4.1), discuss alternative options for mobile phone programming (Section 7.4.2), and contrast SPARK with existing tools that specifically target PyS60 programming (Section 7.4.3).

7.4.1 Rapid Prototyping Platforms

Holleis and Schmidt's MakeIt [148] system allows for the creation of application interfaces for mobile phones using (state, action)-graphs. MakeIt specifically targets the gap between Integrated Development Environments (IDEs) and paper prototyping. One of its key features is the fact that created applications can be simulated and analyzed according to the Keystroke-Level Model (KLM). Its main difference from SPARK is the fact that the system's focus is on interface creation and that it allows only for the creation of Java ME templates. This requires developers to use a Java ME development environment and limits the prototypes' functionality to what Java ME supports.

Campaignr [149] is a C++ Symbian software for mobile phones that supports data collection from a large set of sensors. It can be used to get both continuous and triggered audio, video, and location data from the device, allowing also non-experts in C++ Symbian to easily access and use this information. In a similar manner, the MyExperience [150] system also supports the collection of data from users, with the specific feature of allowing users to enrich the data gathered with subjective information that focuses on the user's activities (e.g., "working" or "biking"). Compared to SPARK, both Campaignr and MyExperience are specific applications that can be configured using XML files. They are specialized for the creation of customized data-collection applications and do not support the development of other application types.

Li and Landy's ActivityDesigner [151] allows users to create activity-driven prototypes using a visual designer and to test and deploy these applications on multiple devices, including mobile phones. This platform is similar to SPARK in so far as it targets non-experts, and allows for the simple and fast creation of applications suitable for prolonged, real-world usage. However, ActivityDesigner focuses more on the design process itself, and supports mo-

bile phone deployment only as part of a JavaScript-based Web application that runs in a mobile phone's Web browser.

ContextPhone [152] is a prototyping platform consisting of C++ Symbian modules that can be customized to sense and store data on a mobile phone and to communicate data between a phone and a "sink" application. While ContextPhone can significantly ease and accelerate application creation for mobile sensing applications, it is not an out-of-the-box solution, but a collection of C++ Symbian components that require users to be familiar with C++ Symbian development. This renders it unusable for non-expert users [135]. The same holds for the rapid application development system presented in [153], which provides a framework for C++ Symbian development that eases GUI creation, data access, and communication on this platform.

7.4.2 Mobile Phone Programming Options

Today's smartphones support an abundance of different programming options. We focused our work on the Symbian S60 platform, as this platform was the prominent development platform when starting the project in 2008. It still features the most available devices [154] and is currently the only platform with a scripting environment that includes bindings for most system functions.

Whereas the iPhone has attracted significant interest and momentum as an open development environment, it requires the use of Objective C for native application development. While being a powerful language, Objective C is not suited for prototype creation by non-experts. Scripting languages, on the other hand, are currently not supported on the iPhone due to license restrictions from Apple.

Java ME [136] is in general a good programming option for non-experts, featuring extensive tool support and documentation. There is a large set of frameworks available that simplify and accelerate application development, such as J2ME Polish [155]. The BaToo toolkit [14] we created falls into the same category, being mainly a Java ME framework that supports the creation of services to retail products, based on bar code recognition. However, as we already pointed out, Java ME offers only a limited set of APIs and lacks extension options, such as the Java Native Interface (JNI) in MIDP [136], which severely restricts application capabilities. Examples of restrictions include the lack of support for newer sensors, such as 3D accelerometers, lack of support for accessing video images from the camera for image recognition tasks (e.g.,

allowing only access to single images), or the lack of in-depth control of Bluetooth scanning processes.

C++ Symbian offers speed and full control of devices, but it features a very steep learning curve, mainly due to the sub-optimal tool support and lack of documentation. Furthermore, unique Symbian language concepts, such as ActiveObjects, Descriptors, or Resource files are complicated and thus difficult to use [135]. For the rapid prototype creation by non-experts, C++ Symbian is no option.

One additional platform that became attractive and is rapidly evolving is Android [156]. Especially since the availability of the Android Native Development Kit (NDK), which allows developers to write applications both in an easy-to-learn Java dialect for the Dalvik JMV [157], and outsource performance-critical code in C / C++ modules. However, despite its ease of use and well-documented resources, programming Android devices still requires more time and is less interactive compared to a dynamic scripting language executed on the phone. Nevertheless, projects like SL4A[37] (Scripting Layer for Android) [158] have recently emerged, which aim not only at allowing scripting languages to be interactively executed on Android devices, but also provide a set of bindings for several existing APIs.

7.4.3 PyS60-Related Tools

Python for S60 has been introduced by Nokia in 2006 and has since then gained a lot of attention by developers, which is due to several reasons: It is simple to learn and use, it features a large set of APIs (especially compared to Java ME), it is extensible through C++ modules, it features an open source license, and an abundance of demo applications for numerous tasks is already available [159, 160].

Due to PyS60's popularity there are already tools available that address some of the shortcomings that we identified in Section 7.2. PUTools [147] and the PythonShell application contained in the standard PyS60 Python distribution [143] feature a remote Bluetooth console that allows users to remotely execute commands on the mobile phone. The SPARK implementation of this feature offers an easier setup by completely encapsulating the Bluetooth connection setup – a process that needs to be done manually by the developer when using PUTools or the PythonShell. SPARK also supports WiFi and cellu-

[37] As of February 2011 the project is still in Alpha status.

lar networks for remote connections, e.g., when the desktop machine has no Bluetooth hardware, or when access to the phone is required “in the field”. The latter use case is complemented by allowing SPARK developers, for example, to inspect devices by taking remote screenshots. Furthermore, SPARK provides syntax highlighting and persistent logging, and the Bluetooth console is integrated into the remote console dialog described in Section 7.3.2 that offers additional services.

PUTools also provides a command line tool for copying files to the mobile phone. In contrast, the feature offered by SPARK is seamlessly integrated into the SPARK environment, allowing the developer to copy and execute project files using a single button press. SPARK also simplifies application execution, which is not covered at all by the PUTools. Whenever a new version is uploaded to the phone, SPARK automatically ensures that the old module version is unloaded so that the new version will be considered upon execution. Using SPARK, the developer is also relieved from micro management tasks, such as deciding where on the phone to store the application files during development and deployment. SPARK ensures that all Python source code files and resources are always found and properly managed.

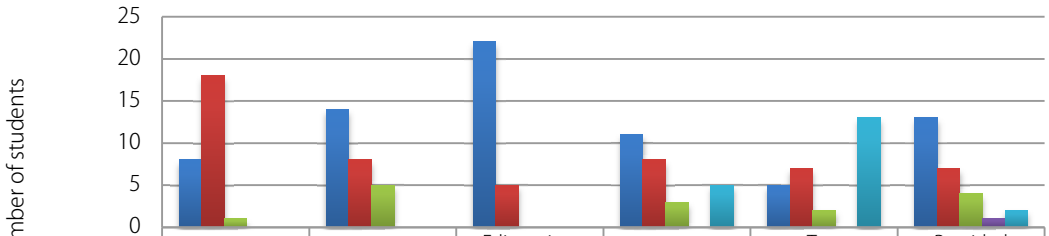
Ensymble [147] is a development project offering a command line tool for packaging PyS60 files into standalone SIS applications. SPARK uses the Ensymble tool and extends it with additional features to reduce the amount of knowledge required by the developer. In addition to providing a GUI for Ensymble, SPARK adds the following features: The ability to automatically choose modules that have the appropriate capabilities and have been signed using the correct certificate, the automatic creation of a suitable application UID, icon creation, a code parser for recommending external modules to include, and an auto-update mechanism.

7.5 Use Case 1: Use in Teaching

7.5.1 Setup

We used the SPARK rapid prototyping environment in the fall of 2008 as part of a practical course accompanying a lecture on distributed systems. The course had 73 enrolled graduate students, all from computer science. The topic "mobile phone programming" was not covered in the lecture, and students were not required to have had prior courses or knowledge in mobile phone programming. The idea was to have students explore some of the concepts of distributed systems in a hands-on manner by programming applica-

tions on a mobile phone. Students had three weeks to design and implement a project using PyS60. They were given access to the SPARK environment, but were free to use it or not (and instead rely on the standard PyS60 tools). Due to the limited time available, students were given only a 30 minute introduction to PyS60 and SPARK. They were then asked to form groups of two to work on the projects. We had 50 Nokia N95/8g devices at our disposal[38], so each group was given one or two phones. The students were free to take the devices home during the exercise period. Feedback was gained in multiple ways: At the end of the project, students were asked to present their applications to their peers and to us. We also encouraged students to approach us with questions during the exercise and asked them to fill out an online questionnaire with 26 questions about PyS60 and the SPARK rapid prototyping environment at the end of the course. We received 30 answers to the anonymous questionnaire (a return rate of 41%). The SPARK environments by default logged all system events and user interactions with the software into a simple text file. Participants were informed about this logging, had the chance to review the data, and were asked to provide us with this file on a voluntary basis. Twenty-five students sent us the log files.



	Easy setup (phone and PC software)	Having a remote console	Edit project files on PC and execute them on the phone	Package files and create distributable SIS files	Test programs in parallel on different phones	Provided extension modules (e.g. bar code recognition)
Highly useful	8	14	22	11	5	13
Useful	18	8	5	8	7	7
Somewhat useful	1	5	0	3	2	4
Not useful	0	0	0	0	0	1
N/A	0	0	0	5	13	2

Figure 7.6 Given answers to the question "How useful do you think are the following features of the SPARK environment?"

[38] Thirty devices were graciously provided by Nokia.

7.5.2 Results

All 30 respondents ended up using SPARK, and only 2 students also used alternative methods to develop their PyS60 application.

Entry barriers: The students were asked how easy it was for them to get started with PyS60 development using SPARK. The majority found it very easy (51.9%) or easy (44.4%). We also asked students to tell us about any installation problems they experienced with the SPARK environment. Fifteen answered this optional question, with 13 stating they had no problems and 2 stating that they had Bluetooth issues on Windows Vista and Linux.

SPARK features: Figure 7.6 shows how useful students rated the various features of SPARK for their project. Support for automated SIS package generation and for concurrent development on different device types were rated as being not that important. Since these features address problems occurring mainly in real-world deployments, this is not surprising. Asked to tell us what kind of features they missed, most students stated that they wished they had had access a real PyS60 debugger (77.8%).

General Feedback: The majority of students found the SPARK environment highly useful (66.7%) or useful (25.9%) for realizing their project. All students indicated that they would use the software again and that they would recommend it to colleagues. Answers to the open question about what they liked most about the software included: "Easy to use; good looking; worked instantly", "It makes the development very easy and fast.", "The time you need to deploy and test the code is very short.", "Easy, simple, fast, reliable, free to use". Things that students disliked about SPARK were: "Bluetooth problems" and "Missing built-in help".

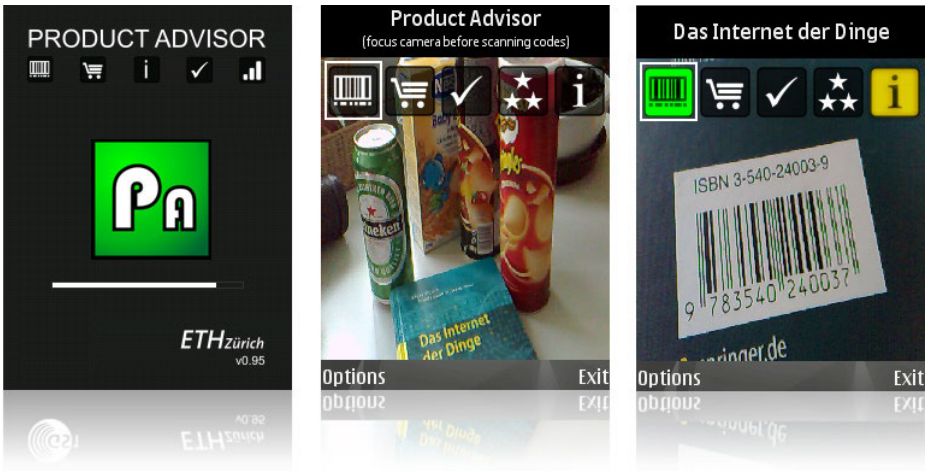


Figure 7.7 Screenshots from the Product Advisor application showing the start screen (left image) and general layout with the camera images in the background and the different plug-in icons on top (middle and right image).

7.6 Use Case 2: Product Advisor

The "Product Advisor" application is a proof-of-concept demonstrating that SPARK and PyS60 can be used to build complex and time-sensitive mobile services. The application represents a mobile phone-based information platform for retail products. It allows users to automatically recognize products using a PyS60 extension module we have written in C++ Symbian that encapsulates a version of our sharp bar code decoder. The application itself acts as a container in which different information "plug-ins" (arbitrary PyS60 applications) can be executed. For example, one such plug-in could compare product ingredients against a user-provided allergy list to alert consumers to allergic reactions, similar to the already presented "Allergy Check" application.

The basic application layout (see Figure 7.7) is optimized to provide users with a quick overview of a product and allows them to easily access more detailed information if required, e.g., why the product might be dangerous for them. The application background always shows the camera images, thus allowing the user to keep the product in sight and scan its bar code. Each plug-in is represented by a separate icon at the top of the screen. After a product has been identified by scanning its bar code, the icons change and signal through their shape and color the most relevant information about this item. This provides the user with a quick overview of the product. If the user is interested in further details, he or she can select a plug-in using the phone's "left" and "right" keys. If selected, a plug-in shows more information or provides additional services (see Figure 7.8, Figure 7.9 and Figure 7.10).

Plug-ins are regular Python programs that inherit from a provided abstract class, which allows them to get informed by the Product Advisor framework about relevant events, such as recognized bar codes, key presses or gestures performed by the user. On devices like the N95/8g, which feature a 3D accelerometer, gesture support is included, allowing users, for example, to reset the application by shaking the phone. Users can either choose what kind of plug-ins they would like to have active or, due to the dynamic nature of Python, plug-ins can also be loaded dynamically at runtime. Depending on what kind of product has been recognized, different plug-ins could be loaded. Product-related data is either stored on the device (e.g., for offline demos) or can be retrieved online using a simple REST [161] interface (HTTP requests and JSON). The application includes a robust logging system that allows us to monitor both user interactions and system events, such as low battery or network connectivity.

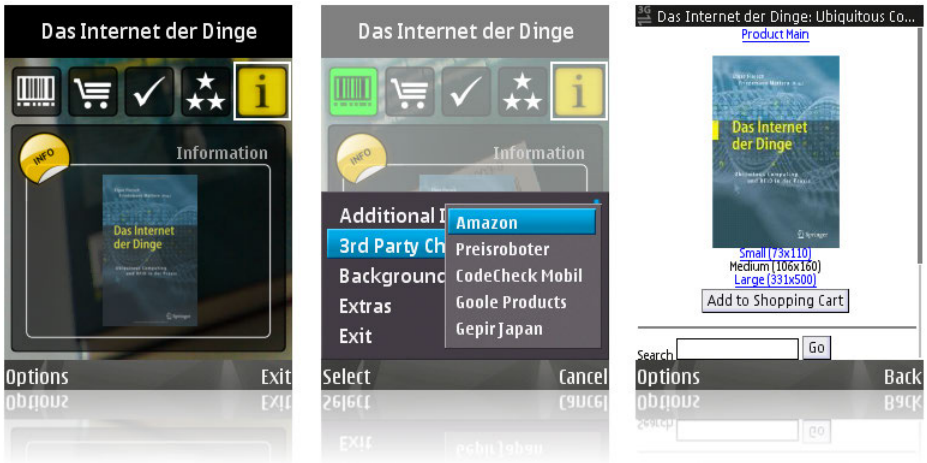


Figure 7.8 Screenshots of the "General Information"-plugin that displays additional information about a product and provides access to further information sources, such as Amazon, price-comparison websites, and others.

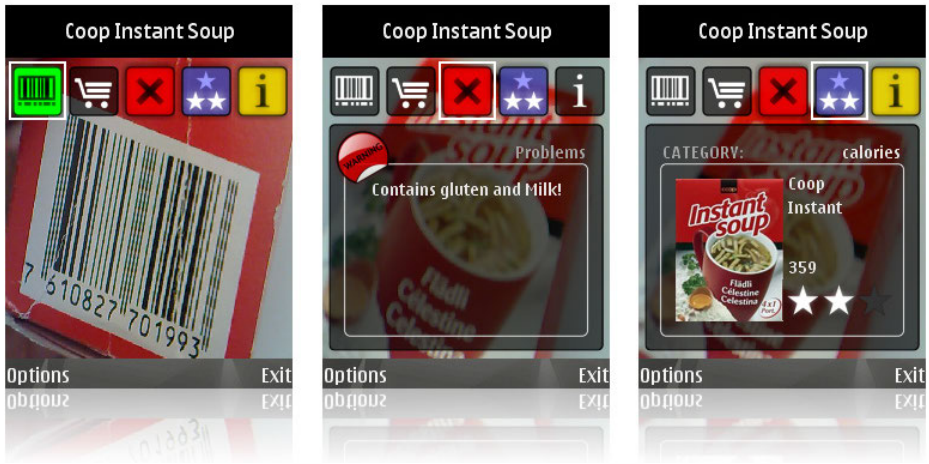


Figure 7.9 Screenshots of the "Allergy Check"-plugin as well as the "Product Rating"-plugin.

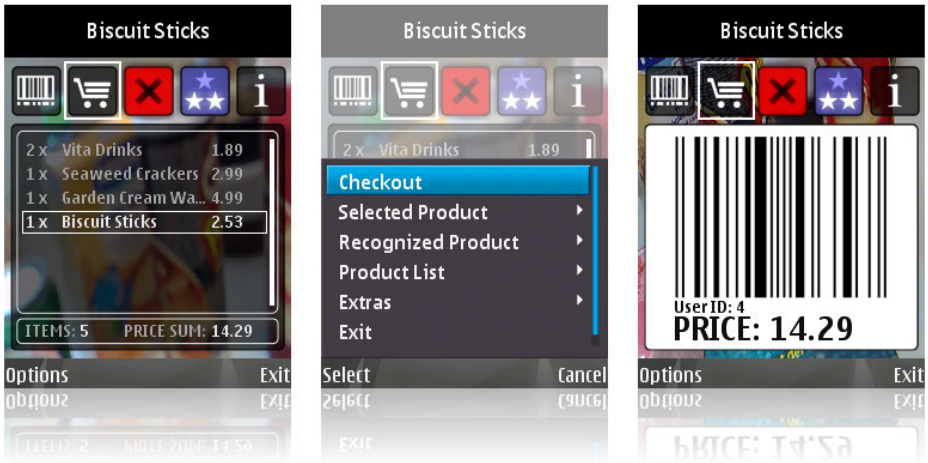


Figure 7.10 Screenshots showing the "Self-Checkout"-plugin, which allows users to manage scanned products on a list and displays a reference bar code that can be presented by users at the POS (point of sale) for self-checkout.

The project consists of 64 files, including 15 Python source code files with a total of approximately 6000 lines of code (including comments). The time required to come up with a fully functioning and tested application using SPARK was around 10 days, full time (the PyS60 module for the bar code recognition was already implemented). The final SIS file that includes all required resources has a size of 804 kB (233 kB without the Python interpreter). The overall performance of the application on the N95/8g mobile phone is good, resulting in an average of 16 frames per second with the bar code recognition running in the background. This indicates that the use of PyS60 and C++ is an attractive combination, in which time critical tasks can be performed in C++ Symbian and the application logic and the user interface (UI) are implemented using PyS60.

During development, several features of SPARK have proven helpful: The possibility to copy end execute files with a button-press on the phone was valuable not only for code generation, but also during UI graphics design. To see how a particular design would integrate into the application, we simply saved an updated version of an image from our desktop image processing tool and pressed the "Copy & Run" button. In this way, changes were applied in seconds. The fact that only changed files are uploaded to the phone also became relevant once the project increased in size. The simple creation of distributable SIS files was used several times to send demos of the application to different people wanting to see the current state of development. Finally, when implementing the optional accelerometer-based gesture support, we also took advantage of the possibility to develop on two different phone models in parallel, one with a built-in accelerometer (N95/8g) and one without it (N73), to ensure that the application behaves correctly in both cases.

7.7 Summary

In this chapter, we presented a comprehensive overview of current rapid prototyping tools for mobile phones and identified PyS60 as an attractive choice that combines ease-of-use with flexibility. However, PyS60 development still pose considerable barriers for the rapid prototyping of applications, especially for new users and when targeting real-world deployments. We discussed these barriers and presented the SPARK environment that targets non-expert users and supports the fast and simple application development and monitoring. In particular, SPARK offers the following:

- Low entry barriers, providing an OS-independent out-of-the box solution for development
- Support for rapid development, e.g., remote device control console and one-click upload and execution
- Abstraction from complex tasks, e.g., application signing and SIS file creation
- Support for large deployments, e.g., simple application deployment and remote monitoring, as well as support for parallel development on different device types

We also presented the results of two case studies in which SPARK has been used: a graduate course on distributed systems, in which 73 students used SPARK to develop mobile applications, as well as the development of a mobile phone-based product information platform. SPARK is available for download [162].

8 Conclusions

In this chapter, we conclude with a summary of the thesis and the contributions made, outline the main limitations of the work presented, and discuss promising topics for future work.

8.1 Summary

Linking products with information and services using mobile phones is a specific use case of the well-known concept of linking real-world objects with virtual information. It is interesting, as there are many beneficial applications possible that have the potential to enable consumers to access personalized, otherwise not available information about products when and where this information is required most (e.g., in stores). Furthermore, many components required for the large-scale deployment of according applications and services are already in place. World-wide, sales items are labeled with a bar code, camera-equipped mobile phones are ubiquitous, and there is an abundance of product-related information available. A fast, reliable and robust bar code recognition method is a central enabling technology for such consumer-oriented mobile services.

We started this thesis by presenting challenges inherent to the recognition of bar codes with mobile phones and provided an overview of bar code symbologies relevant in the context of consumer-oriented mobile services. In order to establish the required background knowledge for the description of the bar code recognition method, we covered the most commonly used symbology (EAN13/UPC-A) in detail.

Chapter 3 described the developed recognition algorithm and detailed our approach for the recognition of bar codes in blurry images. We compared the algorithm to related work and concluded that it primarily differs in two aspects from other approaches. First, it is capable of recognizing bar codes in very blurry images and relies on pre-computed patterns in order to achieve a high recognition speed and accuracy. Second, it combines two different decoder architectures with different strengths and weaknesses in order to address the various recognition challenges.

In order to confirm the practicability of our approach, proof-of-concept implementations on three major mobile phone platforms are provided: iOS, Android, and C++ Symbian. Chapter 4 covered relevant details regarding these

implementations and presented the tools created and used to optimize the recognition method for different mobile phone models.

Chapter 5 compared our recognition method to existing bar code scanning solutions for mobile phones. We conducted a user study with 16 participants and compared the scan speed and accuracy of scanners under realistic conditions. In addition, we analyzed the scan performance and features provided by scanners under controlled conditions. Results of the user study and scanner analysis show that our recognition method outperforms other solutions in terms of scan accuracy and recognition speed. Furthermore, it shows that our approach for recognizing bar codes in blurry images works in practice and provides advantages regarding the required scan-time and the recognition of small bar codes, even on devices with autofocus-cameras. Based on the findings of the study, we derived general observations about users' scan behaviors and presented user interface guidelines for mobile phone-based bar code scanners.

Chapter 6 concretized the concept of mobile services and discussed alternative product identification technologies for this use case, including the recognition of 2D codes, RFID/NFC technology, general image recognition, and the manual entry of a product's bar code number or name. It shows that while each technology has its use cases and specific advantages, the optical recognition of bar codes is the most suitable one for enabling mobile services today and in the near future. Most sales items already have bar codes printed on them, and many users have camera-equipped mobile phones capable of reading these codes.

Finally, Chapter 7 presented SPARK, a rapid prototyping environment targeted specifically at novice users that allows for the fast and easy creation of mobile applications on C++ Symbian devices. We provided an overview of existing rapid prototyping tools for mobile phones, covered the design and implementation of SPARK, and presented the results of two case studies in which it has been used: a graduate course on distributed systems, in which more than 70 students used SPARK to develop mobile applications, as well as the development of a mobile phone-based product information platform.

8.2 Contributions

The overall goal of this thesis is to foster the development of mobile phone-based applications that link information and services to products. In particular, the contributions can be summarized as follows:

- The main contribution of this thesis is a *method for the recognition of bar codes on mobile phones* that is capable of recognizing bar codes in blurry images and that outperforms existing systems in terms of recognition speed and accuracy. With the presented algorithm and proof-of-concept implementations, we hope to contribute to the advancement of this enabling technology for mobile services. Furthermore, we hope that our analysis of the recognition challenges and the findings of the user study and scanner analysis support application developers in making informed decisions when using this technology.
- A *rapid prototyping environment that eases the development of mobile applications,* targeted specifically at novice users. By enabling non-experts to quickly prototype novel application ideas, we hope to foster the creation of mobile services.

8.3 Limitations and Future Work

While the presented recognition method works in general very well, several issues remain that could not be addressed in this thesis, some of which could be the subject of future work.

One existing limitation is related to the required optimization process of the recognition algorithm for different mobile phone models. In the case of devices with autofocus or phone models with fixed-focus cameras that produce only moderately blurry images, the recognition method will usually work well without a dedicated optimization process for specific phone models. However, for mobile phones with fixed-focus cameras that result in very blurry images, a one-time, beforehand optimization process for each new phone model is required. This optimization process adjusts the recognition parameters to the camera optics of a specific phone model in order to maximize the recognition accuracy on this device. This approach has two limitations. First, the optimization process must be performed for each new mobile phone model. Despite being feasible, this is an additional effort and a phone has to be physically accessible in order to take test images. Second, variations in the camera modules sometimes exist, even in the case of the same mobile phone model. While the recognition still works on non-optimized devices, the recognition performance is not optimal.

Future work could consist of the development of a solution that automatically calibrates our recognition method with respect to the specifics of the built-in camera module found in a phone model. Ideally, the calibration should happen continuously and directly on the device, during regular use of the

recognition engine. The automatic calibration for each individual phone could result in a more accurate recognition of codes in blurry images and would allow for the bar code recognition to scale well with the increasing number of different camera phones on the market.

Another limitation is the large file size of the recognition tables that store pre-computed patterns used by the blurry decoder (see Section 3.3). Each table file requires less than one megabyte of space, and two to three recognition tables are usually sufficient to guarantee a good recognition rate. Despite these facts, the recognition table files are by far the largest part of the complete distributable of the bar code recognition (see Figure 4.5). Particularly, in case the recognition tables for several phone models have to be included in a distributable packet, the file size rapidly increases.

In order to overcome this problem, the pre-calculated patterns and values stored in recognition table files could be generated directly on the mobile phone. This process is time-consuming, but it might be performed either when the bar code recognition is used for the first time, or successively in the background while the recognition is already working. The latter will result in a situation, in which bar codes in sharp images can be recognized immediately, and the recognition performance on blurry images will improve over time, once the recognition tables are computed.

A final, promising topic for future work might be the development of a server-side, real-time recognition of bar codes for mobile devices. Cellular networks are evolving fast and with the next generation of networks, LTE (Long Term Evolution), on the horizon, network delays are expected to drop. A thin-client approach that consists of a basic application on the mobile phone for accessing the camera images and transmitting the data to a server, in combination with server-based bar code recognition, might provide several benefits compared to the recognition of codes directly on the phone:

1. The increased performance available on the server can be used to improve the recognition accuracy.
2. All applications that include the bar code recognition instantly benefit from an upgrade of the recognition method on the server; there is no need to update software on the mobile phone.
3. Mobile phones with minimal resources and those featuring only a Java virtual machine and no support for native code can be supported.
4. Despite the required data transmission, it might be possible to reduce the overall energy consumption on the mobile phone because expensive calculations can be performed on the server.

9 Appendices

9.1 Implementation Details

9.1.1 Recognition Table Layout

Figure 9.1shows the detailed layout of information stored in a recognition table file.

9.1.2 Recognition Table Set Optimization

Below is an excerpt from a protocol file that contains the results when recognizing all images in a given set of test images with a specific recognition table. The excerpt shows 5 out of the 1941 results the original file contains.

```
<PROTOCOL VERSION>:2.0
<TABLE
FILE>:C:\external_data\table_set\new_table_set\basic_set\table_26_11_6
<PATH TO TEST IMAGES>:C:\test_images\iphone3gs
<NUMBER TEST IMAGES>:1941

<IMAGE
FILE>:C:\test_images\iphone3gs\large_codes\0051122160226\1271317526823_16.pn
g <RESULT>:NOT RECOGNIZED<CODE NUMBER>:6654424160426<CODE CONFI-
DENCE>:21821<ESTIMATED PSF>:153<NUMBER EP>:3<STD>:-1.0

<IMAGE
FILE>:C:\test_images\iphone3gs\large_codes\0051122160226\1271317526823_17.pn
g <RESULT>:RECOGNIZED<CODE NUMBER>:0051122160226<CODE CONFI-
DENCE>:439<ESTIMATED PSF>:85<NUMBER EP>:59<STD>:-1.0

<IMAGE
FILE>:C:\test_images\iphone3gs\large_codes\0051122160226\1271317526823_18.pn
g <RESULT>: RECOGNIZED<CODE NUMBER>:0051122160226<CODE CONFI-
DENCE>:469<ESTIMATED PSF>:85<NUMBER EP>:59<STD>:-1.0

...

<IMAGE
FILE>:C:\test_images\iphone3gs\small_codes\9789736370540\1270114904145_38.pn
g <RESULT>:RECOGNIZED<CODE NUMBER>:9789736370540<CODE CONFI-
DENCE>:5344<ESTIMATED PSF>:153<NUMBER EP>:35<STD>:-1.0

<IMAGE
FILE>:C:\test_images\iphone3gs\small_codes\9789736370540\1270114904145_39.pn
g <RESULT>:RECOGNIZED<CODE NUMBER>:9789736370540<CODE CONFI-
DENCE>:4298<ESTIMATED PSF>:154<NUMBER EP>:37<STD>:-1.0

<CORRECTLY RECOGNIZED>:85.41988
<WRONGLY RECOGNIZED>:12.261721
<NOT RECOGNIZED>:2.3183975
```

Layout Recognition Table v5

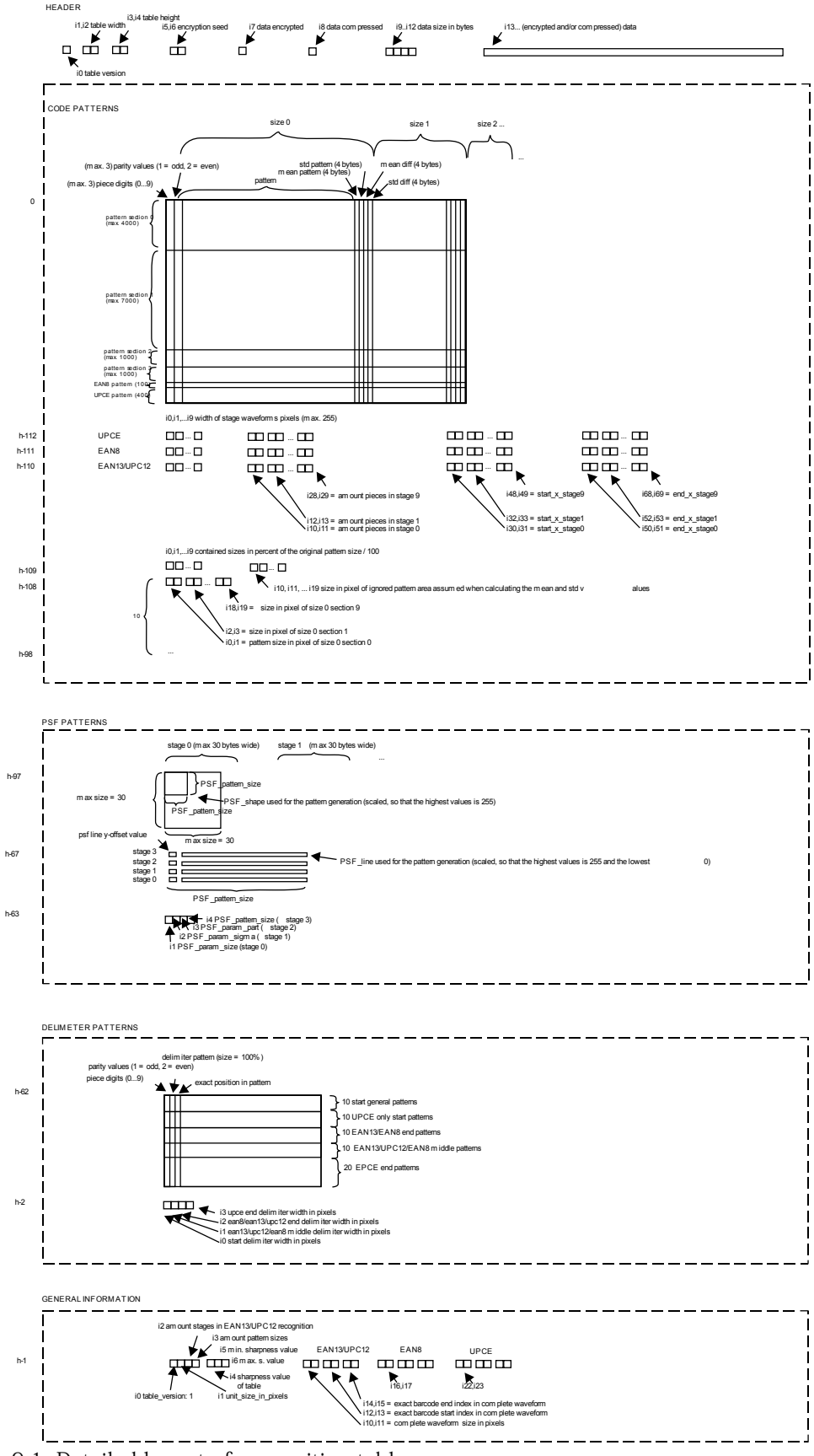
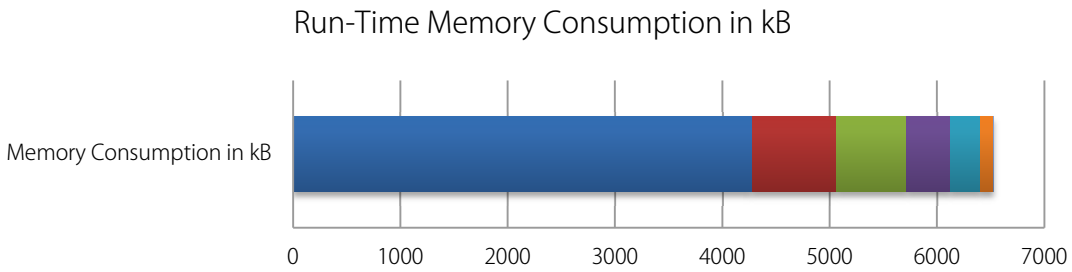


Figure 9.1 Detailed layout of recognition table.

9.1.3 Memory Consumption



	Memory Consumption in kB
Two Recognition Tables	4279,48
Help Functions	787,03
Scan Line Data	650,39
Orientation Detection	405,94
Blurry Decoder	281,38
Offset Maps	120,94
Sharp Decoder	5,02

Figure 9.2 Runtime memory requirements of the recognition engine.

Figure 9.2 shows the run-time memory consumption of our implementation of the recognition algorithm.

9.2 Details on the User Study

9.2.1 Scanner Order

Table 9.1 Order in which test users tested different scanner applications on the different mobile phones:

Test Person	RedLaser	ShopSavvy	ScanDK	pic2sop	i-nigma	Google
iPhone3GS						
1	1	2	3	4	5	-
2	5	3	2	1	4	-
3	5	2	4	3	1	-
4	3	4	1	2	5	-
5	5	4	3	1	2	-
6	4	1	3	2	5	-
7	2	4	5	3	1	-
8	4	1	3	5	2	-
9	3	1	4	2	5	-
10	1	2	3	5	4	-

11	5	1	3	4	2	-
12	5	1	3	4	2	-
13	1	3	5	2	4	-
14	4	5	2	1	3	-
15	2	3	1	4	5	-
16	5	2	4	3	1	-
iPhone3G						
1	1	2	3	4	-	-
2	1	4	2	3	-	-
3	3	1	4	2	-	-
4	2	1	4	3	-	-
5	2	4	1	3	-	-
6	3	2	4	1	-	-
7	1	3	4	2	-	-
8	1	3	2	4	-	-
9	3	4	1	2	-	-
10	4	3	1	2	-	-
11	2	1	4	3	-	-
12	2	1	4	3	-	-
13	1	2	3	4	-	-
14	3	1	4	2	-	-
15	1	4	3	2	-	-
16	2	4	3	1	-	-
HTC Desire						
1	-	-	1	-	-	2
2	-	-	1	-	-	2
3	-	-	2	-	-	1
4	-	-	2	-	-	1
5	-	-	1	-	-	2
6	-	-	2	-	-	1
7	-	-	2	-	-	1
8	-	-	2	-	-	1
9	-	-	1	-	-	2
10	-	-	2	-	-	1
11	-	-	2	-	-	1
12	-	-	2	-	-	1
13	-	-	1	-	-	2

14	-	-	2	-	-	1
15	-	-	2	-	-	1
16	-	-	1	-	-	2

Table 9.2 Order in which the three different phones have been used by each test person:

Test Person	iPhone 3GS	iPhone 3G	HTC Desire
1	1	2	3
2	1	2	3
3	2	3	1
4	1	3	2
5	3	1	2
6	1	2	3
7	3	2	1
8	1	3	2
9	2	3	1
10	1	3	2
11	1	2	3
12	1	2	3
13	2	1	3
14	3	2	1
15	3	1	2
16	1	3	2

9.2.2 Original German User Comments

Table 9.3 German user comments recorded during the user study and their translations:

Translation	German Comment
"The good thing about the red line is that he tells me how to hold it (the phone)"	"Das ist das Gute an der roten Linie, er sagt mir, wie ich es halten muss"
"very user friendly"	"super bedienfreundlich"
"I don't know what this red or green means, it is completely random. It showed me already so many green lines, the code should be definitely recognized by now."	"Ich hab keine Ahnung, was das mit dem Rot und Grün soll. Es ist komplett random. Er hat mir schon so viele grüne Balken gezeigt, der müsste schon längst erkannt sein."

"I don't get it, you have no clue about what is happening."	"Ich durchschau den nicht. Man weiss gar nicht was er macht"
"I'm not sure I'm too stupid or the scanner"	"Ich frag mich, ob ich zu doof bin, oder er."
"I wondered a few times why certain codes are recognized so fast and others not at all. How can this be?"	"Ein paar Mal habe ich mich gewundert, dass die einen so schnell, und die anderen überhaupt nicht erkannt werden, das kann doch irgendwie nicht sein, oder?"
"What is so difficult about this code?"	"Was ist denn an dem so schwer?"
"Why do they (scanners) have problems with this code?"	"Warum haben die mit dem immer so Probleme?"
"He (the scanner) does not like that (code). But he (the code) looks so easy."	"Den will er jetzt nicht. Dabei sieht er so einfach aus."
"Why is it (this code) hard (to recognize)?"	"wieso ist der schwer?"

9.2.3 Dynamic Range of Tested Scanners

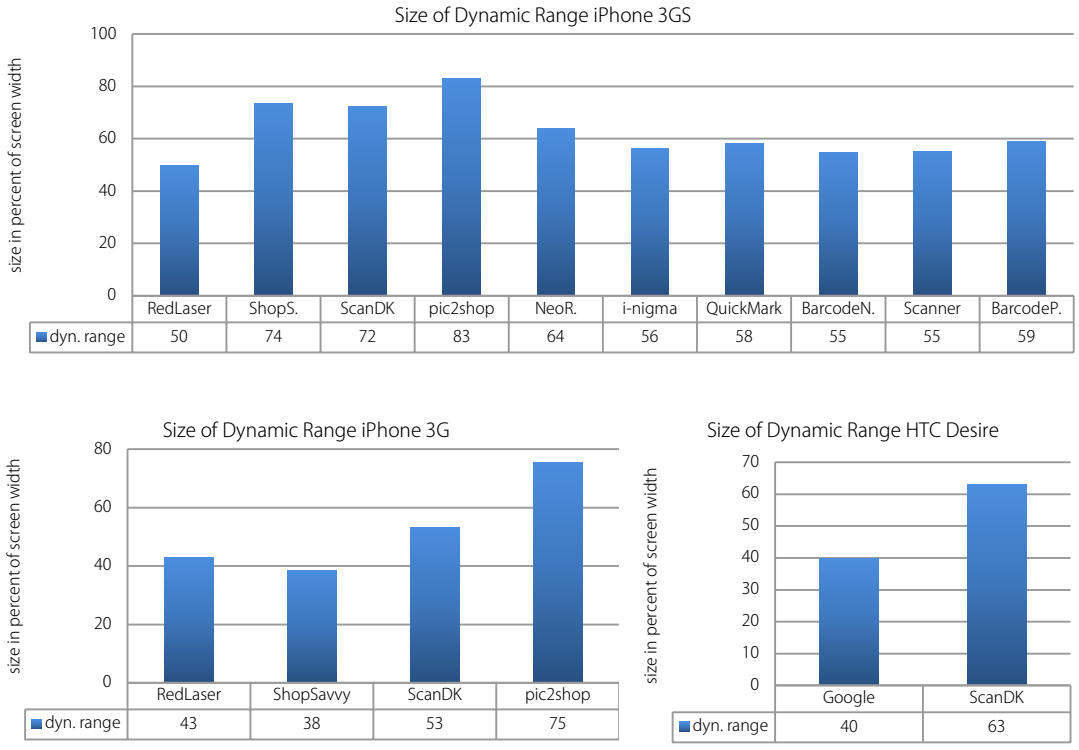


Figure 9.3 Dynamic range of tested bar code scanners.

10 Bibliography

[1] P. D. Wellner, “Interacting with paper on the DigitalDesk,” *Communications of the ACM,* vol. 36, no. 7, pp. 87-96, 1993.

[2] R. Barrett, and P. P. Maglio, “Informative things: how to attach information to the real world,” in Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology, San Francisco, California, United States, 1998, pp. 81-88.

[3] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic, “People, Places, Things: Web Presence for the Real World,” *Mobile Networks and Applications,* vol. 7, no. 5, pp. 365-376, 2002.

[4] F. Siegemund, C. Floerkemeier, and H. Vogt, “The value of handhelds in smart environments,” *Personal Ubiquitous Computing,* vol. 9, no. 2, pp. 69-80, 2005.

[5] R. Ballagas, J. Borchers, M. Rohs, and J. G. Sheridan, “The smart phone: a ubiquitous input device,” *Pervasive Computing, IEEE,* vol. 5, no. 1, pp. 70-77, 2006.

[6] F. Reischach, F. Michahelles, D. Guinard, R. Adelmann, E. Fleisch, and A. Schmidt, “An Evaluation of Product Identification Techniques for Mobile Phones,” in Proceedings of the 12th International Conference on Human-Computer Interaction - Volume I, Uppsala, Sweden, 2009, pp. 804-816.

[7] R. Want, K. P. Fishkin, A. Gujar, and B. L. Harrison, “Bridging physical and virtual worlds with electronic tags,” in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Pittsburgh, Pennsylvania, United States, 1999, pp. 370-377.

[8] L. Pohjanheimo, H. Kernen, and H. Ailisto, “Implementing touchme paradigm with a mobile phone,” in Proceedings of the Joint Conference on Smart Objects and Ambient Intelligence, Grenoble, France, 2005, pp. 87-92.

[9] R. Angeles, "Rfid Technologies: Supply-Chain Applications and Implementation Issues," *Information Systems Management,* vol. 22, no. 1, pp. 51-65, 2005.

[10] M. Rohs, "Real-world interaction with camera-phones," in Proceedings of the 2nd International Symposium on Ubiquitous Computing Systems (UCS), Tokyo, Japan, 2004, pp. 39-48.

[11] Z. Chunhui, W. Jian, H. Shi, Y. Mo, and Z. Zhengyou, "Automatic Real-Time Barcode Localization in Complex Scenes," in Proceedings of the IEEE International Conference on Image Processing, 2006, pp. 497-500.

[12] Metro Future Store Innitiative. February 2nd, 2011; www.future-store.org/fsi-internet/html/en/459/index.html.

[13] Markant AG. February 2nd, 2011; www.markant.com.

[14] R. Adelmann, M. Langheinrich, and C. Floerkemeier, "A Toolkit for Bar Code Recognition and Resolving on Camera Phones – Jump Starting the Internet of Things," in Proceedings of the Workshop on Mobile and Embedded Interactive Systems (MEIS) at Informatik 2006, Dresden, Germany, 2006.

[15] C. Myung-Jin, and S. Sung-Yong, "Development of compact auto focus actuator for camera phone by applying new electromagnetic configuration," *Journal of Mechanical Science and Technology,* vol. 20, no. 12, pp. 2087-2093, December, 2006.

[16] C. Myung-Jin, Y. Yang-Hee, and A. Woo-Hyun, "Development of compact camera module having auto focus actuator and mechanical shutter system for mobile phone," in Proceedings of the International Conference on Control, Automation and Systems, 2007, pp. 2319-2322.

[17] P. Moran, S. Dharmatilleke, A. Khaw, K. Tan, M. Chan, and I. Rodriguez, "Fluidic lenses with variable focal length," *Applied Physics Letters,* vol. 88, no. 4, 2006.

[18] H. Yang, C.-Y. Yang, and M.-S. Yeh, "Miniaturized variable-focus lens fabrication using liquid filling technique," *Microsystems Technology,* vol. 14, no. 7, pp. 1067-1072, 2008.

[19] ISO/IEC Standard. "EAN/UPC bar code symbology specification," www.iso.org/iso/catalogue_detail.htm?csnumber=46143.

[20] J. Swartz, and Y. P. Wang, "Fundamentals of Bar Code Information Theory," *Computer,* vol. 23, no. 4, pp. 74-86, 1990.

[21] M. Rohs, "Visual Code Widgets for Marker-Based Interaction," in Proceedings of the 5th International Workshop on Smart Appliances and Wearable Computing - Volume 5, 2005, pp. 506-513.

[22] R. Adelmann, "Mobile Phone Based Interaction with Everyday Products - On the Go," in International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST), Cardiff, Wales, UK, 2007, pp. 63-69.

[23] P. Hough, *Methods and means for recognizing complex patterns.*, US Patent 3,069,654, 1962.

[24] R. Duda, and P. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM,* vol. 15, no. 1, pp. 11-15, 1972.

[25] G. Olmo, and E. Magli, "All-integer Hough transform: performance evaluation," *Image Processing,* vol. 3, pp. 338-341, 2001.

[26] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal,* vol. 4, no. 1, pp. 25-30, 1965.

[27] P. D. Wellner, *Adaptive Thresholding for the DigitalDesk*, EuroPARC Technical Report EPC, 1993.

[28] R. Gonzalez, and R. Woods, *Digital Image Processing*: Addison-Wesley Longman Publishing Co., Inc., 2001.

[29] H. Schweitzer, J. Bell, and F. Wu, "Very Fast Template Matching," *Computer Vision — ECCV 2002*, Lecture Notes in Computer Science, A. Heyden, G. Sparr, M. Nielsen *et al.*, eds., pp. 145-148: Springer Berlin / Heidelberg, 2006.

[30] W. Turin, and R. A. Boie, "Bar code recovery via the EM algorithm," *IEEE Transactions on Signal Processing,* vol. 46, no. 2, pp. 354-363, 1998.

[31] T. Wittman, "Lost in the Supermarket: Decoding Blurry Barcodes," *SiamNews*, 37 (7), 2004.

[32] Axtel - Machine Vision. March 1st, 2011; www.axtel.com.

[33] CharacTell - Image Recognition Software. March 1st, 2011; www.charactell.com.

[34] C. Viard-Gaudin, N. Normand, and D. Barba, "A bar code location algorithm using a two-dimensional approach," in Proceedings of the 2nd International Conference on Document Analysis and Recognition, 1993, pp. 45-48.

[35] M. Kuroki, T. Yoneoka, T. Satou, Y. Takagi, T. Kitamura, and N. Kayamori, "Bar-code recognition system using image processing," in Proceedings of the 6th International Conference on Emerging Technologies and Factory Automation Proceedings (ETFA), 1997, pp. 568-572.

[36] F. Xianyong, W. Fuli, L. Bin, Z. Haifeng, and W. Peng, "Automatic Recognition of Noisy Code-39 Barcode," in Proceedings of the 16th International Conference on Artificial Reality and Telexistence Workshops (ICAT), 2006, pp. 79-82.

[37] H. Hee Il, and J. Joung Koo, "Implementation of algorithm to decode two-dimensional barcode PDF-417," in Proceedings of the 6th International Conference on Signal Processing, 2002, pp. 1791-1794 vol.2.

[38] K. Aas, and L. Eikvil, *Decoding Bar Codes from Human-Readable Characters*, Amsterdam: Elsevier, 1997.

[39] Grabba Bar Code Scanner Attachment for Mobile Phones. February 2nd, 2011; http://grabba.com/portal/index.php.

[40] M. Rohs, and J. Bohn, "Entry Points into a Smart Campus Environment - Overview of the ETHOC System," in Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003, pp. 260.

[41] S. M. Youssef, and R. M. Salem, "Automated barcode recognition for smart identification and inspection automation," *Expert System Applications,* vol. 33, no. 4, pp. 968-977, 2007.

[42] R. Muniz, L. Junco, and A. Otero, "A robust software barcode reader using the Hough transform," in Proceedings of the International

Conference on Information Intelligence and Systems, 1999, pp. 313-319.

[43] L. Shu-Jen, L. Hong-Yuan, C. Liang-Hua, T. Hsiao-Rong, and H. Jun-Wei, "Camera-based bar code recognition system using neural net," in Proceedings of the International Joint Conference on Neural Networks (IJCNN), Nagoya, Japan, 1993, pp. 1301-1305.

[44] A. K. Jain, and Y. Chen, "Bar code localization using texture analysis," in Proceedings of the 2nd International Conference on Document Analysis and Recognition, 1993, pp. 41-44.

[45] R. J. Howlett, S. Berthier, and G. J. Awcock, *Determining the location of industrial bar codes using neural networks*, London: Institution of Electrical Engineers, 1997.

[46] D. Chai, and F. Hock, "Locating and Decoding EAN-13 Barcodes from Images Captured by Digital Cameras," in Proceedings of the 5th International Conference on Information, Communications and Signal Processing, 2005, pp. 1595-1599.

[47] Y. Huijuan, J. Xudong, and A. C. Kot, "Accurate localization of four extreme corners for barcode images captured by mobile phones," in Proceedings of the 17th IEEE International Conference on Image Processing (ICIP), 2010, pp. 3897-3900.

[48] X. Feng, and G. M. J., "Locating barcodes using JPEG 2000 compressed data," in Proceedings of Visual Communications and Image Processing, Beijing, China, 2005, pp. 1-9.

[49] A. Tropf, and D. Chai, "Locating 1-D Bar Codes in Dct-Domain," in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2006, pp. II-II.

[50] E. Ohbuchi, H. Hanaizumi, and L. A. Hock, "Barcode readers using the camera device in mobile phones," in International Conference on Cyberworlds, 2004, pp. 260-265.

[51] Y. Hu, J. Huang, and Z. Ma, "A low cost barcode recognition method," in Proceedings ot the 6th International Symposium on Instrumentation and Control Technology, Beijing, China, 2006, pp. 1-5.

[52] X. Wu, L. Qiao, and J. Deng, "A New Method for Bar Code Localization and Recognition," in Proceedings of 2nd International Congress on Image and Signal Processing (CISP '09), 2009, pp. 1-6.

[53] L. YeMin, and Z. Li, "Research and application of the EAN-13 barcode recognition on iphone," in Proceedings of the International Conference on Future Information Technology and Management Engineering (FITME), 2010, pp. 92-95.

[54] R. Puetter, T. Gosnell, and A. Yahil, "Digital image reconstruction: Deblurring and denoising," *Astronomy and Astrophysics,* vol. 43, no. 1, pp. 139, 2005.

[55] H. C. Andrews, and B. R. Hunt, *Digital Image Restoration*: Prentice Hall Professional Technical Reference, 1977.

[56] R. G. Lane, "Methods for maximum-likelihood deconvolution," *Journal of the Optical Society of America,* vol. 13, no. 10, pp. 1992-1998, 1996.

[57] M. A. T. Figueiredo, and R. D. Nowak, "An EM algorithm for wavelet-based image restoration," *IEEE Transactions on Image Processing,* vol. 12, no. 8, pp. 906-916, 2003.

[58] E. Y. Lam, "Blind bi-level image restoration with iterated quadratic programming," *IEEE Transactions on Circuits and Systems,* vol. 54, no. 1, pp. 52-56, Jan, 2007.

[59] L. Ta-Hsin, and L. Ke-Shin, "Deblurring two-tone images by a joint estimation approach using higher-order statistics," in Proceedings of the IEEE Signal Processing Workshop on Higher-Order Statistics, 1997, pp. 108-111.

[60] D. Kundur, and D. Hatzinakos, "Blind image deconvolution," *IEEE Signal Processing Magazine,* vol. 13, no. 3, pp. 43-64, 1996.

[61] Matlab Imaging Toolbox - Lucy-Richardsson Method for Deconvolution. March 10th, 2011; www.mathworks.com/help/toolbox/images/ref/deconvlucy.html.

[62] E. Joseph, and T. Pavlidis, "Bar code waveform recognition using peak locations," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 16, no. 6, pp. 630-640, 1994.

[63] E. Joseph, and T. Pavlidis, "Waveform recognition with application to bar codes," in Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 1991, pp. 129-134 vol.1.

[64] E. Selim, "Blind deconvolution of bar code signals," *Inverse Problems,* vol. 20, no. 1, pp. 121, 2004.

[65] K. Q. Wang, Y. M. Zou, and H. Wang, "1D bar code reading on camera phones," *International Journal of Image Graphics,* vol. 7, no. 3, pp. 529-550, 2007.

[66] R. Shams, and P. Sadeghi, "Bar Code Recognition in Highly Distorted and Low Resolution Images," in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2007, pp. I-737-I-740.

[67] J. C. Rocholl, S. Klenk, and G. Heidemann, "Robust 1D Barcode Recognition on Mobile Devices," in Proceedings of the 20th International Conference on Pattern Recognition, Istanbul, Turkey, 2010, pp. 2712-2715.

[68] J. C. Rocholl, "Robust 1D Barcode Recognition on Mobile Devices," Diploma Thesis, Institute of Visualization and Interactive Systems, Department of Intelligent Systems, University of Stuttgart, Stuttgart, Germany, 2009.

[69] iOS Development Center. February 7th, 2011; http://developer.apple.com/devcenter/ios/index.action.

[70] Android. February 7th, 2011; www.android.com.

[71] Symbian Ltd. Symbian Signed User Guide. February 6th, 2011; www.symbiansigned.com/app/page.

[72] RedLaser Bar Code Scanner. February 7th, 2011; http://redlaser.com.

[73] ShopSavvy Bar Code Scanner. February 7th, 2011; http://shopsavvy.mobi.

[74] pic2shop Bar Code Scanner. February 7th, 2011; www.pic2shop.com.

[75] Matlab - The Language of Technical Computing. 7th March, 2011; www.mathworks.com.

[76] Google Goggles Project. February 1st, 2011; www.google.com/mobile/goggles.

[77] F. von Reischach, S. Karpischek, F. Michahelles, and R. Adelmann, “Evaluation of 1D barcode scanning on mobile phones,” in Proceedings of the Internet of Things (IOT), 2010, pp. 1-5.

[78] iPhone 3GS Device Specifications. February 7th, 2011; www.apple.com/iphone/iphone-3gs/specs.html.

[79] iPhone 3G Device Specifications. February 7th, 2011; www.gsmarena.com/apple_iphone_3g-2424.php.

[80] HTC Desire Device Specifications. February 7th, 2011; www.gsmarena.com/htc_desire-3077.php.

[81] i-nigma Bar Code Scanner. February 7th, 2011; www.i-nigma.com/i-nigmahp.html.

[82] Metro Group MEA Application. February 6th, 2011; www.future-store.org/fsi-internet/html/en/7568/index.html.

[83] J. Rekimoto, and Y. Ayatsuka, “CyberCode: designing augmented reality environments with visual tags,” in Proceedings of DARE 2000 on Designing Augmented Reality Environments, Elsinore, Denmark, 2000, pp. 1-10.

[84] C. Cheong, T.-D. Han, J.-Y. Kim, T.-J. Kim, K. Lee, S.-Y. Lee, A. Itoh, Y. Asada, and C. Craney, “Pictorial Image Code: A Color Vision-based Automatic Identification Interface for Mobile Computing Environments,” in Proceedings of the 8th IEEE Workshop on Mobile Computing Systems and Applications, 2007, pp. 23-28.

[85] H. Kato, and K. T. Tan, “Pervasive 2D Barcodes for Camera Phone Applications,” *IEEE Pervasive Computing,* vol. 6, no. 4, pp. 76-85, 2007.

[86] Denso Wave Incorporated. February 1st, 2011; www.qrcode.com.

[87] NTT DoCoMo. January 2nd, 2011; www.nttdocomo.com.

[88] Denso Wave Incorporated. "QR Code Features," February 2nd, 2011; www.qrcode.com/qrfeature-e.html.

[89] ISO/IEC Standard. "Bar code symbology - QR Code," www.iso.org/iso/catalogue_detail.htm?csnumber=30789.

[90] M. Ebling, and R. Caceres, “Bar Codes Everywhere You Look,” *IEEE Pervasive Computing,* vol. 9, no. 2, pp. 4-5, 2010.

[91] Google Favorite Places Project. February 5th, 2011; www.google.com/help/maps/favoriteplaces/business/barcode.htmt.

[92] K. A. H. Nurwono, and R. Kosala, "Color quick response code for mobile content distribution," in Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia, Kuala Lumpur, Malaysia, 2009, pp. 267-271.

[93] T. Langlotz, and O. Bimber, "Unsynchronized 4D barcodes: coding and decoding time-multiplexed 2D colorcodes," in Proceedings of the 3rd International Conference on Advances in Visual Computing - Volume I, Lake Tahoe, NV, USA, 2007, pp. 363-374.

[94] NFC Forum. April 3rd, 2011; www.nfc-forum.org.

[95] EPC Global. February 1st, 2011; www.epcglobalus.org.

[96] T. Staake, F. Thiesse, and E. Fleisch, "Extending the EPC network: the potential of RFID in anti-counterfeiting," in Proceedings of the ACM Symposium on Applied Computing, Santa Fe, New Mexico, 2005, pp. 1607-1612.

[97] S. Sarma, D. Brock, and K. Ashton, *The networked physical world*, Massachussets Institute of Technology, Auto-ID Center White Paper.

[98] J. Ondrus, and Y. Pigneur, "An Assessment of NFC for Future Mobile Payment Systems," in Proceedings of the International Conference on the Management of Mobile Business (ICMB), 2007, pp. 43-43.

[99] A. Geven, P. Strassl, B. Ferro, M. Tscheligi, and H. Schwab, "Experiencing real-world interaction: results from a NFC user experience field trial," in Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services, Singapore, 2007, pp. 234-237.

[100] E. O'Neill, P. Thompson, S. Garzonis, and A. Warr, "Reach out and touch: using NFC and 2D barcodes for service discovery and interaction with mobile devices," in Proceedings of the 5th International Conference on Pervasive Computing, Toronto, Canada, 2007, pp. 19-36.

[101] B. Erol, E. Antunez, and J. J. Hull, "HOTPAPER: multimedia interaction with paper using mobile phones," in Proceedings of the 16th ACM

International Conference on Multimedia, Vancouver, British Columbia, Canada, 2008, pp. 399-408.

[102] T. Quack, H. Bay, and L. V. Gool, “Object recognition for the internet of things,” in Proceedings of the 1st International Conference on the Internet of Things, Zurich, Switzerland, 2008, pp. 230-246.

[103] D. Nister, and H. Stewenius, “Scalable Recognition with a Vocabulary Tree,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2, 2006, pp. 2161-2168.

[104] Kooaba AG. February 1st, 2011; www.kooaba.com.

[105] S. Gammeter, A. Gassmann, L. Bossard, T. Quack, and L. Van Gool, “Server-side object recognition and client-side object tracking for mobile augmented reality,” in Proceedings of the Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2010, pp. 1-8.

[106] GS1. "DataBar Revolution Brochure," February 2nd, 2011; www.gs1.org/sites/default/files/docs/barcodes/databar/GS1_DataBar_Revolution_Brochure.pdf.

[107] GS1. February 1st, 2011; www.gs1.org.

[108] GS1. "DataBar Transition Whitepaper," February 2nd, 2011; www.ncr.com/documents/gs1_databar_trans_wp.pdf.

[109] GS1. "North American Coupon Application Guidline Using GS1 DataBar (RSS) Expanded Symbols," February 2nd, 2011; www.gmaonline.org/filemanager/Events/North_American_Coupon_Application_Guideline.pdf.

[110] GS1. "DataBar Readiness Report," February 2nd, 2011; www.gs1.org/barcodes/databar/readiness.

[111] GSA - The Global Mobile Suppliers Associaton. "GSA EDGE Fact Sheet," January 2nd, 2011; www.gsacom.com/gsm_3g/edge_databank.php4#EDGE_Fact_Sheet.

[112] GSA - The Global Mobile Suppliers Associaton. "GSA Global HSPA+ Network Commitments Report 2011," January 2nd, 2011; www.gsacom.com.

[113] GSA - The Global Mobile Suppliers Association. "GSA Evolution to LTE Report," January 2nd, 2011; www.gsacom.com.

[114] GSA - The Global Mobile Suppliers Association. "GSA Evolution to LTE Overview," January 2nd, 2011; www.gsacom.com.

[115] P. Stuckmann, N. Ehlers, and B. Wouters, “GPRS traffic performance measurements,” in Proceedings of the Vehicular Technology Conference (VTC), 2002, pp. 1289-1293.

[116] D. Astely, E. Dahlman, A. Furuskar, Y. Jading, M. Lindstrom, and S. Parkvall, “LTE: the evolution of mobile broadband,” *Communications Magazine, IEEE,* vol. 47, no. 4, pp. 44-51, 2009.

[117] Mobile World Live. "GSM coverage maps," January 2nd, 2011; http://maps.mobileworldlive.com.

[118] Amazon. February 2nd, 2011; www.amazon.com.

[119] BestBuy. January 2nd, 2011; www.bestbuy.com.

[120] Google. January 2nd, 2011; www.google.com.

[121] Alliance for Justice (AFJ). January 2nd, 2011; www.afj.org.

[122] Consumer Action. January 2nd, 2011; www.consumer-action.org.

[123] Max Havelaar. February 2nd, 2011; www.maxhavelaar.ch/en.

[124] Ktipp. January 2nd, 2011; www.ktipp.ch.

[125] ÖkoTest. January 2nd, 2011; www.oekotest.de.

[126] WikiFood - The Wiki for foodstuff. January 2nd, 2011; www.wikifood.lu.

[127] CodeCheck. January 2nd, 2011; www.codecheck.info.

[128] GS1. "GDSN Certified Data Pools," January 2nd, 2011; www.gs1.org/docs/gdsn/gdsn_certified_data_pools.pdf.

[129] GS1. "GDSN Global Data Synchronization Network Whitepaper," January 2nd, 2011; www.gs1.org/sites/default/files/docs/gdsn/gdsn_brochure.pdf.

[130] R. Adelmann, and M. Langheinrich, "SPARK Rapid Prototyping Environment – Mobile Phone Development Made Easy," *Intelligent Interactive Assistance and Mobile Multimedia Computing*, Communications in Computer and Information Science, D. Tavangarian,

T. Kirste, D. Timmermann *et al.*, eds., pp. 225-237: Springer Berlin Heidelberg, 2009.

[131] A. Lamarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit, "Place Lab: Device Positioning Using Radio Beacons in the Wild," in Proceedings of the 3rd International Conference on Pervasive Computing, 2005.

[132] T. Nicolai, E. Yoneki, N. Behrens, and H. Kenn, "Exploring Social Context with the Wireless Rope," *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, R. Meersman, Z. Tari and P. Herrero, eds., pp. 874-883: Springer Berlin Heidelberg, 2006.

[133] N. Maisonneuve, M. Stevens, M. E. Niessen, P. Hanappe, and L. Steels, "Citizen noise pollution monitoring," in Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government, 2009, pp. 96-103.

[134] S. Santini, B. Ostermaier, and R. Adelmann, "On the use of sensor nodes and mobile phones for the assessment of noise pollution levels in urban environments," in Proceedings of the 6th International Conference on Networked Sensing Systems, Pittsburgh, Pennsylvania, USA, 2009, pp. 31-38.

[135] M. Huebscher, N. Pryce, N. Dulay, and P. Thompson, "Issues in Developing Ubicomp Applications on Symbian Phones," in Proceedings of the International Workshop on System Support for Future Mobile Computing Applications, 2006, pp. 51-56.

[136] J2ME Java 2 Micro Edition. February 6th, 2011; http://java.sun.com/javame/index.jsp.

[137] H. Richard, and N. Phil, *Symbian OS C++ for Mobile Phones*: John Wiley & Sons, Inc., 2003.

[138] Objective C Programming Language. February 6th, 2011; http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC.

[139] Lua Scripting Language for S60 Devices. February 6th, 2011; http://luaforge.net/projects/luas60.

[140] Ruby for S60 Devices. February 6th, 2011; http://ruby-symbian.rubyforge.org.

[141] FlashLite. February 6th, 2011; www.adobe.com/products/flashlite.

[142] Hecl – The Mobile Scripting Language. February 6th, 2011; www.hecl.org.

[143] Python for S60 Open Source Project. February 6th, 2011; http://sourceforge.net/projects/pys60.

[144] J. Laurila, V. Tuulos, and R. MacLaverty, "Scripting Environment for Pervasive Application Exploration on Mobile Phones," in Proceedings of the 4th International Conference on Pervasive Computing, Dublin, Ireland, 2006.

[145] W3C, "Scalable Vector Graphics (SVG) Tiny 1.2 Specification," February 7th, 2008.

[146] BlueCove Library for Bluetooth (JSR-82) Implementation. February 6th, 2011; www.bluecove.org.

[147] Ensymble Developer Utilities for Symbian OS. February 6th, 2011; http://code.google.com/p/ensymble.

[148] P. Holleis, and A. Schmidt, "MakeIt: Integrate User Interaction Times in the Design Process of Mobile Applications," in Proceedings of the 6th International Conference on Pervasive Computing, Sydney, Australia, 2008, pp. 56-74.

[149] A. Joki, J. A. Burke, and D. Estrin, *Campaignr: A Framework for Participatory Data Collection on Mobile Phones,* Technical Report 770, UC Los Angeles: Center for Embedded Network Sensing, 2007.

[150] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay, "MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones," in Proceedings of the 5th International Conference on Mobile Systems, Applications and Services, San Juan, Puerto Rico, 2007, pp. 57-70.

[151] Y. Li, and J. A. Landay, "Activity-based prototyping of ubicomp applications for long-lived, everyday human activities," in Proceedings

of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems, Florence, Italy, 2008, pp. 1303-1312.

[152] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen, "ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications," *IEEE Pervasive Computing,* vol. 4, no. 2, pp. 51-59, 2005.

[153] S. Long, R. Kooper, G. D. Abowd, and C. G. Atkeson, "Rapid prototyping of mobile context-aware applications: the Cyberguide case study," in Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking, Rye, New York, United States, 1996, pp. 97-107.

[154] Gartner Report on Smartphone Sales, *Market Share: Smartphones, Worldwide, 3Q08*, 2008.

[155] J2ME Polish. Febraury 6th, 2011; www.j2mepolish.org/cms.

[156] M. Butler, "Android: Changing the Mobile Landscape," *IEEE Pervasive Computing,* vol. 10, no. 1, pp. 4-7, 2011.

[157] Dalvik Virtual Machine. January 7th, 2011; www.dalvikvm.com.

[158] SL4A Scripting Languages for Android Project. February 7th, 2011; http://code.google.com/p/android-scripting.

[159] Forum Nokia Python Resources. February 6th, 2011; http://wiki.forum.nokia.com/index.php/Category:Python.

[160] J. Scheible, and V. Tuulos, *Mobile Python: Rapid prototyping of applications on the mobile platform*: Wiley Publishing, 2007.

[161] L. Richardson, and S. Ruby, *RESTful web services*: O'Reilly, 2007.

[162] SPARK - Rapid Prototyping Environment for Mobile Services. February 7th, 2011; http://people.inf.ethz.ch/adelmanr/spark.

[163] GS1. "DataBar Business Cases," April 3rd, 2011; www.gs1.org/docs/barcodes/databar/GS1_DataBar_Business_Case_Complete.pdf.