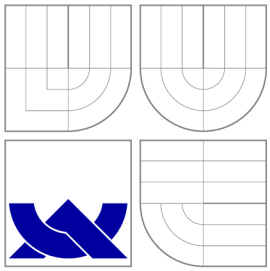
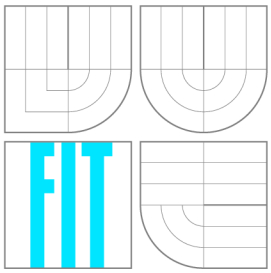


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KNIHOVNA PRO ČTENÍ ČÁROVÝCH KÓDŮ

BARCODES READING LIBRARY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR NOVOHRADSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK KUBÍČEK
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce se zabývá dekódováním různých druhů čárových kódů v obraze. Vybrané druhy čárových kódů podrobně popisuje. Představuje přístup k dekódování lineárních i maticových čárových kódů. Praktická část se zabývá návrhem a implementací programové knihovny pro čtení vybraných typů čárových kódů v jazyce Java.

Abstract

This work deals with the decoding of different types of barcodes in the image. Selected types of bar codes are described in detail. Presents an approach to decode linear and matrix bar codes. The practical part deals with the design and implementation of software library for reading selected types of bar codes in Java.

Klíčová slova

čárový kód, UPC, EAN-13, Code 25, Code 128, QR Code, PDF417, Data Matrix, dekódování čárového kódu

Keywords

barcode, UPC, EAN-13, Code 25, Code 128, QR Code, PDF417, Data Matrix, barcode decoding

Citace

Petr Novohradský: Knihovna pro čtení čárových kódů, bakalářská práce, Brno, FIT VUT v Brně, 2011

Knihovna pro čtení čárových kódů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Kubíčka.

.....
Petr Novohradský
18. května 2011

Poděkování

Děkuji vedoucímu práce Ing. Radku Kubíčkovi za cenné rady při tvorbě této práce.

Obsah

1 Úvod **3**

2 Čárový kód **4**
2.1 1D čárové kódy 4
2.1.1 Code 25 5
2.1.2 Code 128 6
2.1.3 EAN-13 7
2.1.4 UPC 7
2.2 2D čárové kódy 8
2.2.1 PDF417 8
2.2.2 Data Matrix 9
2.2.3 QR Code 9

3 Návrh **11**
3.1 Návrh čtení 1D kódů 11
3.2 Návrh čtení 2D kódů 12
3.3 Společné části knihovny 13
3.4 Ukázková aplikace 14
3.5 Implementační prostředky 14

4 Popis implementace **15**
4.1 Implementované typy kódů 15
4.2 Implementace čtení 1D kódů 15
4.2.1 Získání vzorku dat 16
4.2.2 Dekódování dat 17
4.2.3 Ověření kontrolního součtu 18
4.3 Implementace čtení 2D kódů 18
4.3.1 Přípravné činnosti 18
4.3.2 Čtení a dekódování dat 19
4.4 Implementace společných částí knihovny 21
4.5 Shrnutí požadavků na vstupní obraz 21
4.5.1 1D kódy 21
4.5.2 2D kódy 22
4.6 Implementace ukázkové aplikace 22

5 Dosažené výsledky **23**
5.1 Výsledky prahování 23
5.2 Výsledky čtení 1D kódů 24
5.3 Výsledky čtení 2D kódů 24

6 Závěr **25**

A Obsah CD **27**

B Uživatelský manuál **28**

Kapitola 1

Úvod

S čárovými kódy se setkáváme prakticky každý den. Lze je nalézt například na naprosté většině zboží u obchodníků. Čárové kódy jsou oblíbené všude tam, kde je třeba provádět rychlou a spolehlivou identifikaci předmětů. Může to být třeba v přepravní firmě, větším skladu, výrobě nebo pro účel inventarizace majetku. Jsou oblíbené zejména pro jednoduchost použití a nízké náklady na označení.

Čárový kód byl vytvořen jako strojově čitelný kód. Pokud tento kód vidí člověk, vidí pouze několik černých čar na bílém podkladu. Informace, která se v takovém kódu ukrývá, není pro člověka čitelná. Proto jsou některé typy čárových kódů doplněny také textem, který člověk může přečíst. Tento text pak lze použít, pokud se z nějakého důvodu nepodařilo čárový kód přečíst strojově.

Čárový kód v původní podobě je tvořen černými čarami na bílém podkladu. Informace je zakódována do šířky těchto čar a mezer mezi nimi. Černé čáry většinu dopadajícího světla pohltí, naopak bílé mezery většinu světla odrazí. Čtečka čárových kódů se pak obvykle skládá z laserového paprsku, který přejíždí napříč čárovým kódem, a prvku citlivého na světlo, který měří intenzitu odraženého světla. Toto však není jediný způsob strojového čtení. Jiný způsob využívá digitální kameru či fotoaparát k zachycení obrazu čárového kódu. Tento obraz je pak zpracováván programem. Vytvoření takového programu je cílem této práce.

Cílem této práce je navrhnout a realizovat programovou knihovnu, která bude schopna přečíst nejpoužívanějších typy čárových kódů ze statických obrázků. Součástí bude jednoduchá aplikace demonstrující schopnosti této knihovny. Práce si neklade za cíl lokalizaci čárového kódu v obraze. Předpokládá vstupní obraz takový, kde čárový kód zabírá naprostou většinu obrazu.

V kapitole 2 jsou popsány hlavní vlastnosti vybraných typů čárových kódů. V kapitole 3 je popsán návrh čtení a hlavní použité algoritmy. Je zde také vybrán implementační jazyk. V kapitole 4 je podrobně popsána implementace jednotlivých typů čárových kódů. Kapitola 5 pak zhodnotí výsledky dosažené knihovnou. Nakonec kapitola 6 celou práci shrnuje a jsou v ní navrženy možné směry pokračování této práce.

Kapitola 2

Čárový kód

Dnes běžně používané čárové kódy můžeme rozdělit do 2 hlavních skupin. První skupinou jsou *1D čárové kódy*, kde je čárový kód tvořen řadou svislých černých čar a mezerami mezi nimi. Ukázka typického 1D čárového kódu je na obrázku 2.1, kde se jedná o typ EAN-13. Takový kód se pak čte pouze podél jedné osy (odtud 1D, jednorozměrný).

Obrázek 2.1: 1D čárový kód, typ EAN-13.

Druhou skupinou jsou *2D čárové kódy*, kde je čárový kód tvořen maticí černých a bílých bodů. Čtení pak musí probíhat v ploše. Příklad 2D kódu typu QR Code je na obrázku 2.2. 2D kódy obvykle mohou uchovat větší objem informace. Existují i jiné druhy čárových kódů, například kruhové. Takové kódy se však používají pouze ve specifických oblastech a tato práce se jim nevěnuje.

Obrázek 2.2: 2D čárový kód, typ QR Code.

2.1 1D čárové kódy

Jednorozměrné kódy, nebo také lineární symboliky, jsou nejčastěji užívanou třídou čárových kódů. Nejrůznějších typů kódů existuje opravdu mnoho. Tato podkapitola podrobněji rozebere několik často používaných typů.

Nejužší dovolená šířka čáry v 1D kódu se nazývá modul a všechny rozměry jsou vždy dány celočíselným násobkem tohoto modulu. Následující text bude popisovat složení čáro-

vého kódu pomocí řetězce znaků 0 a 1, kde 0 znamená bílou mezeru a 1 znamená černou čáru. Pokud se vyskytuje více stejných znaků za sebou, například 111, znamená to černou čáru tlustou 3 moduly. Zápis 1011 potom bude vypadat jako černá čára, za ní stejně tlustá mezera, následovaná čárou tloušťky dvou modulů.

2.1.1 Code 25

Čárový kód typu Code 25 patří k nejstarším specifikacím čárových kódů. Dostupné popisy jsou třeba v [2] a [3]. Vyskytují se i názvy kódu „Standard 2 of 5“ a „Industrial 2 of 5“, jedná se však stále o tentýž kód.

Kód může uchovat pouze číslice (znaky 0–9), zato ale nemá pevně určenou šířku. V tomto kódu nesou informaci pouze černé čáry. Jednotlivé čáry jsou od sebe vždy odděleny mezerou šířky 1 modul. Čáry se mohou vyskytovat pouze 2 šířek. Úzká šířky 1 modul a široká šířky 3 moduly. Příklad kódu typu Code 25 je na obrázku 2.3.

Obrázek 2.3: Kód typu Code 25.

Symbolice Code 25 se někdy také říká kód 2 z 5 proto, že každá číslice je zakódována pěti čarami, kde vždy 2 jsou široké a 3 úzké. Jednotlivé číslice se zakódují podle tabulky 2.1. Zakódované číslice se pak doplní zleva o startovací sekvenci 11011010 a na konec se přidá ukončovací sekvence 1101011. Pokud takto vzniklý řetězec převedeme na čárový kód, vznikne platný kód Code 25.

Tabulka 2.1: Kódovací tabulka Code 25 [3].

číslice	**zakódovaný tvar**
0	10101110111010
1	11101010101110
2	10111010101110
3	11101110101010
4	10101110101110
5	11101011101010
6	10111011101010
7	10101011101110
8	11101010111010
9	10111010111010

Bohužel, ač se specifikace většinou shodují ve způsobu uvádění startovací a ukončovací sekvence, některé dostupné generátory kódu toto nectí. Některé používají šířku čar v poměru 2:1, jiné v poměru 3:1. Výše uvedené startovací a ukončovací sekvence jsou s poměrem čar 2:1 a ty také bude předpokládat navržená knihovna pro čtení kódu. Samozřejmě kódování číslic uvnitř kódu stále zůstává s poměrem šířek čar 3:1.

Kontrola správnosti přečtení je trochu problematická. Tento způsob zakódování má sice definován způsob výpočtu kontrolní číslice, nicméně její přítomnost na konci kódu není

povinná. Pokud tedy přečteme kód, který vyhovuje všem výše zmíněným pravidlům, ale poslední číslice nevyhovuje výpočtu kontrolního součtu, je kód buďto chybně přečtený, nebo kontrolní číslice nebyla použita. Provádět kontrolu čtení tedy pravděpodobně nebude možné. Nikdy není jisté, zda kontrolní číslice byla skutečně použita.

Protože informaci nesou pouze černé čáry, je výsledný kód poměrně veliký. Proto později vznikla nová verze. Nazývá se „Interleaved 2 of 5“. Kódovací tabulka zůstává zachována ale mění se způsob kódování. V prokládané verzi se pro kódování čar používá střídavě černá a bílá barva. Není tedy třeba oddělovat jednotlivé čáry od sebe. Tím se šetří místo zabrané kódem. Přestože jsou názvy kódů podobné, je důležité si je neplést. Dále v práci bude uvažována pouze neprokládaná varianta.

2.1.2 Code 128

Kód typu Code 128 přináší jednu zásadní změnu – může již přenášet číslice i znaky. Je pravděpodobně nejpoužívanějším typem 1D kódu používaného pro přenos znaků. Počet přenášených znaků není pevně určen. Dostupných popisů kódu je opět značné množství. Kvalitní popisy jsou například v [4] a [11]. Příklad kódu typu Code 128 je na obrázku 2.4.

Obrázek 2.4: Kód typu Code 128.

Kódovací tabulka je oproti ostatním 1D kódům řešena jiným způsobem. Kód 128 má 3 sady symbolů, každá sada obsahuje 103 znaků. Při kódování se lze mezi sadami přepínat a vybrat tak nejkratší možný tvar dané zprávy. Aby bylo jasné, jaká znaková sada se právě používá, existují 3 startovací sekvence, pro každou znakovou sadu jedna. Každá sada pak má v tabulce 2 speciální znaky pro indikaci přepnutí do zbývajících sad. Ukončovací sekvence je vždy stejná.

Jednotlivé sady znaků se označují A, B a C. *Sada A* obsahuje hlavně velká písmena, číslice a speciální znaky z dolní části ASCII[1] tabulky. *Sada B* obsahuje hlavně malá písmena, velká písmena a číslice. *Sada C* obsahuje číslice po dvojicích. Umožňuje tedy efektivně kódovat číselné údaje po dvojicích do jednoho znaku. V jednom kódu se lze mezi jednotlivými znakovými sadami libovolně přepínat. Tento princip je poměrně efektivní a umožňuje zvyšovat hustotu informace v čárovém kódu pomocí výběru nejvhodnější znakové sady. Tyto 3 znakové sady dohromady umožní zakódovat 128 znaků z ASCII tabulky. Kódovací tabulky jsou rozsáhlé, najdete je například v [4].

Čárový kód musí začínat jedním ze 3 startovacích znaků. Každý ze startovacích znaků určuje jednu znakovou sadu. Za startovacím znakem následuje překódovaný obsah včetně případných přepínání znakových sad. Za překódovaný text se doplní kontrolní součet, STOP znak a ukončovací čára (11).

Pro kontrolu správného přečtení slouží povinná kontrolní číslice. Tato kontrolní číslice se vkládá na konec kódovaného textu těsně před ukončovací sekvenci. Oproti jiným kódům

[1] ASCII – American Standard Code for Information Interchange. Tato kódovací tabulka tvoří základ naprosté většiny dalších kódování v počítačích.

(třeba EAN-13) se kontrolní číslice nepíše do textového zápisu pod kódem. Výpočet kontrolní číslice se počítá pomocí váhové sumy. Pro výslednou sumu se spočítá modulo[2] 103 a výsledné číslo se použije jako kontrolní znak. Bližší podrobnosti o výpočtu kontrolního součtu i s příkladem lze nalézt například v [11].

2.1.3 EAN-13

EAN-13 je zřejmě nejznámější typ čárového kódu. Používá se v obchodních domech na zboží. Přenáší pouze číslice, velikost kódu je pevná. Velmi podrobný popis kódu je například v [5] nebo [13].

Může přenášet právě 12 číslic, 13. číslice je povinný kontrolní součet. Aby nedošlo k situaci, kdy dva výrobci označí své zboží stejným kódem, používá se systém přidělování rozsahů čísel. První tří čísla v kódu tak obvykle znamenají zemi, další označují výrobce. Výrobce potom na své produkty tiskne pouze čísla z přiděleného rozsahu [7]. Příklad kódu typu EAN-13 je na obrázku 2.1 v úvodu kapitoly.

Tento kód je však používaný i v jiných oblastech, a tak existují i prefixy pro použití k zakódování ISBN[3] nebo ISSN[4] do čárového kódu.

Zajímavostí tohoto kódování je fakt, že ve vlastním čárovém kódu je uloženo jen 12 číslic. První číslice se totiž odvozuje ze způsobu zakódování číslic 2–7. Tento způsob byl pravděpodobně zvolen pro zpětnou kompatibilitu s kódem typu UPC (viz sekce 2.1.4).

Jak již bylo řečeno, první číslice se v čárovém kódu nevyskytuje. Zbývajících 12 číslic se dělí na levou a pravou polovinu po šesti číslicích. Levá polovina čísel používá dva typy zakódování jednotlivých číslic, s lichým nebo se sudým počtem jedniček. Pravá polovina pro všechny číslice používá inverzi liché varianty. Způsob kódování pro levou polovinu určuje první číslice. Z toho vyplývá že kódovací tabulky jsou 2. První tabulka přiřazuje způsob zakódování levé strany k první číslici, druhá tabulka již udává zakódované řetězce pro vstupní číslice.

Poslední číslice kódu je kontrolní součet, ten musí být vždy přítomen. Výpočet kontrolní číslice je založen na výpočtu váhové sumy jednotlivých čísel a operaci modulo 10 s touto sumou. Kódovací tabulky a vhodný příklad výpočtu kontrolního součtu je v [5].

Po kódování čísel se na začátek doplní startovací sekvence (101) a na konec ukončovací sekvence stejného tvaru. Mezi pravou a levou polovinou kódu je ještě vloženo rozdělení (01010).

2.1.4 UPC

Kód UPC je predchůdcem kódu EAN-13 (sekce 2.1.3) a dá se říci, že kód UPC lze chápat jako podmnožinu EAN-13. Umožňuje zakódovat právě 11 číslic a 1 číslici kontrolního součtu. Pokud jej chceme rozšířit na EAN-13, stačí na začátek doplnit 0. Pak je způsob zakódování shodný s kódováním EAN-13, včetně výpočtu kontrolního součtu. Kód UPC je na první pohled stejný jako EAN-13. Liší se pouze počtem číslic vytištených pod ním. Zatímco pod kódem UPC je 12 číslic, pod kódem EAN-13 je 13 číslic. V implementované knihovně tedy nebude třeba přidávat další způsob čtení, ale využije se algoritmus pro EAN-13.

[2] Zbytek po dělení.
[3] ISBN – International Standard Book Number. Mezinárodní číslování knih pro komerční účely.
[4] ISSN – International Standard Serial Number. Mezinárodní číslování periodických publikací.

2.2 2D čárové kódy

Zatímco 1D čárové kódy jsou reprezentované soustavou černých a bílých rovnoběžných čar, 2D čárové kódy jsou složitější. Jedná se již o čárový kód, který reprezentuje informaci ve dvou osách. Z tohoto důvodu také čárový kód vypadá jinak.

Jednou z možností, jak vytvořit 2D čárový kód, je tzv. typ "stacked barcode". Tento pojem by se dal přeložit například jako stohovaný čárový kód, nebo čárové kódy v patrech. Nejedná se o nic jiného než více jednorozměrných čárových kódů nad sebou. Zástupcem této kategorie je typ PDF417.

Druhou možností je tzv. maticový kód. V tomto případě jsou černé a bílé značky uspořádány do formy pravidelné mřížky, kde jednotlivá políčka mají nejčasteji čtvercový tvar. Zástupcem této kategorie je např. Data Matrix. Všechny zmíněné 2D kódy mají variabilní velikost. Lze tedy uložit různé objemy dat a ovlivnit tím velikost výsledného čárového kódu.

Také snímání 2D kódů se provádí jiným způsobem. Zde si již nevystačíme se snímáním laserovým paprskem. Ten umožňuje snímání pouze po přímce, kdežto pro 2D kódy je třeba snímání v ploše. Proto se používají čtečky na pricipu zpracování obrazu. Taková čtečka pak funguje následujícím způsobem. Obvykle CCD čip sejme obraz stejně jako v běžných digitálních fotoaparátech nebo kamerách. V další fázi je takto vytvořený obrázek zpracováván, obvykle pomocí programů, které vyhodnotí data uložená v čárovém kódu.

Objem přenášené informace je typicky mnohem větší než u 1D kódů, proto se používá jiný způsob zabezpečení informace. 1D čárové kódy používají obvykle jednu číslici kontrolního součtu. 2D kódy již vyžadují lepší zabezpečení a také opravu chybně načtené informace. Všechny zde zmíněné 2D kódy používají pro detekci a opravu chyb kódování typu "Reed-Solomon" [9].

Jedná se o zabezpečení informací pomocí vypočítané kontrolní informace. Kódování má samoopravnou funkci, to znamená, že pokud je část informace pozměněna nebo chybí, existuje matematický způsob jak načtenou informaci opravit do původní podoby. Samozřejmě množství chyb v načtených datech nesmí přesáhnout únosnou mez. Algoritmus pro svoji činnost využívá vlastnosti konečných těles tvořících část diskrétní matematiky. Podrobný popis funkce by překračoval rozsah a určení této práce. Více informací je dostupných třeba v [9].

2.2.1 PDF417

Kód PDF417 patří do kategorie "stacked". Jedná se tedy o více 1D čárových kódů posazených nad sebou tak, aby výsledek vytvořil obdélník. Jednotlivé moduly pak mají obdélníkový tvar, narozdíl od čtvercových modulů v maticových kódech. Kód je standardizován jako ISO 15438. Příklad kódu typu PDF417 je na obrázku 2.5.

Obrázek 2.5: Kód typu PDF417.

Každý řádek je opatřen číslem řádku a dalšími informacemi pro detekci a opravu chyb. Velikost se může pohybovat od 3 do 90 řádků a od 1 do 30 kódových slov na šířku. Všechny řádky musí mít stejný počet kódových slov. Každé kódové slovo se skládá ze 4 čar a 4 mezer. Kódovat lze číslice, znaky a obecná binární data [10].

Tento kód se používá na identifikačních kartách, například některé státy USA jej používají na řidičských průkazech. Umožňuje přenášet až 1,1 kB informací, může tedy obsahovat biometrické údaje. Další použití je například pro identifikaci pacientů v nemocnici včetně jejich diagnózy.

2.2.2 Data Matrix

Data Matrix je formálně definován v normě ISO 16022. Kód je patentován, ale patentová práva nejsou vykonávána. Patří do kategorie maticových kódů. Sám kód má čtvercový nebo obdélníkový tvar. Vnitřní moduly mohou být čtvercového nebo kulatého tvaru [8]. Příklad kódu typu Data Matrix je na obrázku 2.6.

Obrázek 2.6: Kód typu Data Matrix.

Kód umožňuje uložení textů a čísel. Používá se například v logistice na strojově čitelné označení příjemce. Další použití je v elektronice, kde někteří výrobci tímto kódem označují desky tištěných spojů nebo křemíkové čipy. Velmi často jsou těmito kódy značeny díly v leteckém průmyslu. Používá se také v nemocnicích či na letenkách.

Velikost kódu se může měnit v širokém rozsahu. Pohybuje se od velikosti 10×10 modulů po 144×144 modulů [8].

2.2.3 QR Code

Samotný výraz „QR Code“ je registrovaná obchodní známka společnosti DENSO WAVE INCORPORATED. QR kód je také patentován stejnou společností, nicméně patentová práva nejsou vykonávána. K užití a tvorbě QR kódu není třeba žádná licence. Kód je také standardizován jako standard ISO 18004 [1]. Příklad kódu typu QR Code je na obrázku 2.2 v úvodu kapitoly.

Je populární zvláště v Japonsku, rozšiřuje se však po celém světě. Nejznámější použití je na nejrůznějších reklamních materiálech, kde má sloužit ve spojení se čtecím programem pro mobilní telefon. Telefon s fotoaparátem za pomoci čtecího programu dekóduje data, kde je zapsána URL adresa stránky. Uživatel pak může tuto stránku rovnou načíst. Z ostatních způsobů použití se zatím rozšiřuje pouze pro značení výrobků.

Umožňuje uchovat číslice, znaky latinské abecedy a japonské znaky Kanji. Norma [1] připouští i režim ECI, kde je možno kódovat text za použití mnoha různých kódovacích tabulek. Režim ECI je však označen jako nepovinný, tudíž se na jeho implementaci v generátoru či čtečce obecně nelze spoléhat.

QR kód má variabilní velikost. Velikost QR kódu se vyjadřuje tzv. verzí. Verze může nabývat hodnot od 1 do 40, číslo verze pak určuje velikost v počtu modulů. Modul je označení pro velikost nejmenší jednotky kódu. Touto nejmenší jednotkou je bílý nebo černý čtverec. Jednotlivé moduly na sebe přímo navazují. Jsou-li tedy dva stejně barevné moduly vedle sebe, jeví se jako obdélník s poměrem stran 2:1. Z těchto modulů je pak složen celý kód, žádné jiné stavební kameny neobsahuje. QR kód je vždy čtvercový s velikostí hrany 21

modulů pro verzi 1, až po 177 modulů pro verzi 40. Velikost hrany narůstá o 4 pro každou verzi.

Jsou na něm nápadné velké zaměřovací značky ve třech rozích. Podle těchto značek je schopen uživatel rozpoznat, že se jedná o QR kód. Zároveň mají sloužit k automatickému vyhledání kódu v obraze z fotoaparátu či kamery a detekce jeho orientace. Při srovnání s ostatními zmíněnými 2D kódy je struktura QR kódu složitější. Data se umisťují v blocích po 8 bitech, přičemž jsou tyto bloky někdy rozděleny zaměřovacími značkami. Tím může dojít i k situaci, že jeden blok osmi bitů nebude v souvislé oblasti [1].

Pro detekci a opravu chyb je také možné použít jednu ze čtyř definovaných úrovní ochrany. Nejnižší úroveň umožní opravit asi 7 % dat, nejvyšší úroveň pak až 30 %. Norma [1] tyto 4 úrovně označuje jako L, M, Q a H. Vyšší úroveň ochrany také znamená více informací a tím větší kód. Vždy se jedná o samoopravný kód typu Reed-Solomon, pouze je pro jednotlivé úrovně různě nastavený.

Tento typ samoopravného kódu je uzpůsoben na opravy souvislých blokových chyb. Chyby v jednom nebo několika málo bitech rozprostřené pro celé šíři dat velmi snižují možnosti opravy. Při čtení QR kódu se však spíše vyskytnou blokové chyby. Mohou vzniknout například odtržením části kódu, nebo zakrytím menší oblasti nečistotou. Výběr ochranného kódu Reed-Solomon tedy má pro QR kód dobré opodstatnění.

Kódování jde však ještě dál. U větších verzí QR kódu je již objem kódovaných dat natolik velký, že se tato data dělí do bloků a při zápisu se jednotlivé bloky střídají. Každý blok pak má svůj vlastní kontrolní součet. V případě zakrytí části QR kódu je taková chyba rozložena mezi více samostatných bloků. To opět napomáhá větší pravděpodobnosti úspěšné rekonstrukce informace.

Kapitola 3

Návrh

Cílem této práce je vytvořit programovou knihovnu s funkcemi pro čtení čárových kódů. Návrh je tedy zaměřen čistě na návrh knihovny, bez jakéhokoliv uživatelského rozhraní. Provedení aplikace, která může použít tuto knihovnu, je ponecháno plně na uživateli knihovny. Použití knihovny by mělo být co nejjednoduší pro typické úlohy, měla by však být zachována možnost detailní konfigurace jednotlivých fází čtení. Knihovna bude všechny kódy číst z předaných obrázků. Původ obrázků je opět ponechán na uživateli knihovny. Může se tak jednat o prosté obrázky ze souborů či snímky z kamery. Jelikož se od sebe čtení 1D a 2D kódů velmi liší, bude návrh proveden zvlášť pro 1D kódy a 2D kódy.

3.1 Návrh čtení 1D kódů

1D kódy jsou uzpůsobeny pro čtení laserovým paprskem. Laserový paprsek u obvyklých hardwarových čteček přejíždí po úsečce a je jím zaznamenáván průběh intenzit jasu přes celou šířku kódu. Tato knihovna však bude číst kódy z předaných obrázků. V obrázcích je tedy vhodné do jisté míry napodobit princip laserových čteček. Čtení bude probíhat po jedné přímce pixelů vstupního obrázku. Jednotlivé fáze čtení 1D kódu budou v tomto pořadí:

1. *Separace čtecí linky* ve formě řady pixelů. Ze vstupního obrázku musí být vhodným způsobem vybrána posloupnost pixelů, která bude procházet přes všechny čáry kódu. V této fázi ještě není známa pozice začátku kódu. Bude tedy nejspíše nutné čtecí linku vést přes celou šírku obrázku. Pozice vlastního začátku kódu se najde později. Tato fáze bude pro všechny typy kódů stejná, implementaci lze sdílet.

2. *Převod na odstíny šedi.* Posloupnost pixelů získaná v předchozím kroku se nyní převede na odstíny šedi. Barevná informace není pro další zpracování potřeba. Převod na odstíny šedi lze realizovat minimálně dvěma způsoby. Jeden způsob respektuje vnímání jednotlivých barev lidským okem. Z barevných složek se tedy dělá vážený průměr, který smíchá složky podobným způsobem jako lidský mozek. Při tomto způsobu převodu se člověku zdá výsledek přirozený. Vzhledem k tomu, že výstup této fáze není určen člověku, ale dalšímu strojovému zpracování, bude postačovat druhý způsob. Ten provádí obyčejný aritmetický průměr barevných složek. Převod je mírně rychlejší a dalšímu zpracování plně postačuje. Tato fáze bude pro všechny typy kódů stejná, implementaci lze sdílet.

3. *Prahování.* Je třeba jednoznačně rozhodnout, zda se pixel bude zpracovávat jako černý nebo jako bílý. U prahování je nejobtížnější najít prahovou hodnotu, u které se mění barva. Obecně prahovou hodnotu nelze určit dopředu. Vstupní obrázek může být příliš tmavý nebo světlý, vhodná prahová hodnota je pak různá. Nabízí se zde řešení v podobě dynamického prahování. To rozhoduje o prahovací hodnotě s ohledem na okolní pixely. V této práci byl zvolen jiný způsob. Stačí si uvědomit, že prahování není nic jiného než klasifikace vstupu do dvou tříd. Úlohy na klasifikaci dat by se daly přímo použít pro prahování. Bylo tedy využito klasifikačního algoritmu *k-means* [12] v provedení pro jednorozměrný vstup s klasifikací na dvě třídy. Navrhovaná knihovna tedy bude sloužit i k ověření tohoto přístupu k problému prahování.

   Algoritmus k-means zpracuje celou vstupní posloupnost pixelů v odstínech šedi a podle výsledku zpracování vybere vhodnou prahovací hodnotu. Problém by mohl být v rychlosti zpracování. Algoritmus pracuje iterativně a není předem známé, kolik průchodů bude potřebovat na správné rozdělení vstupní posloupnosti. Počet iterací se dá snížit počátečním výběrem středních hodnot jednotlivých skupin. Čím přesnější bude počáteční volba, tím méně iterací bude třeba. Pro odhad tedy použijeme minimální a maximální hodnotu vstupní posloupnosti. Tato fáze bude pro všechny typy kódů stejná, implementaci lze sdílet.

4. *Nalezení startovací sekvence.* Hledání začátku kódu již musí být rozděleno podle typu kódu. Každý kód si definuje vlastní startovací a ukončovací sekvenci. Podle těchto sekvencí je možné automaticky rozpoznat o jaký typ kódu se jedná, nebude to však součástí navrhované knihovny. Uživatel knihovny musí vždy uvést jaký typ kódu je na obrázku. Obecně bude hledání startovací sekvence probíhat sledováním poměrů šířek černých a bílých oblastí. Například kód EAN-13 má definovanou startovací sekvenci 101 (viz sekce 2.1.3), takže se v jeho případě bude hledat poměr 1:1:1. Sekvence samozřejmě musí začínat černou čarou. Součástí hledání startovací sekvence bude také odhad velikosti modulu. Jakmile je nalezena startovací sekvence, lze vypočítat odhad šířky modulu prostým dělením.

5. *Dekódování obsahu.* Dekódování bude probíhat podle kódovacích tabulek jednotlivých typů kódů. V této fázi již je nalezený začátek kódu a je odhadnuta velikost modulu. Lze tedy určit, jak je daná oblast široká v násobcích modulu. Podle toho se již vyhledá význam znaku v tabulce.

6. *Ověření kontrolního součtu.* Některé 1D kódy obsahují kontrolní číslici. Definice výpočtu této kontrolní číslice se u jednotlivých typů liší. Obecně se však jedná pouze o kontrolu správného načtení, obvykle neexistuje možnost opravy informace. Pokud kontrolní číslice načtená z kódu nesouhlasí s vypočítanou, je kód chybně načten. Některé kódy nemají vůbec definovanou kontrolní číslici nebo je obsažena nepovinně (viz sekce 2.1.1).

Skutečná implementace čtení 1D kódů je popsána v sekci 4.2 na straně 15.

3.2 Návrh čtení 2D kódů

Čtení 2D kódů bude vyžadovat značně odlišný způsob. 2D kódy je třeba číst jako matici bodů a z této matice získat data. Už nebude stačit jedna čtecí linka, ale bude potřeba čtecí mřížka. Jednotlivé fáze čtení 2D kódů budou v tomto pořadí:

1. *Převod na stupně šedi.* Stejně jako u 1D kódů se musí barevné obrázky prevést na odstíny šedé. Převod bude probíhat stejným způsobem jako bylo popsáno u 1D kódů (sekce 3.1). Kvůli povaze 2D kódů je nutné zpracovat celý obrázek, ne jen jednu čtecí linku.

2. *Prahování.* Pro prahování stále platí problémy popsané v sekci 3.1. U 2D kódů už by byl objem dat zpracovávaných algoritmem k-means příliš velký a hledání prahové hodnoty by tudíž trvalo příliš dlouho. Vzhledem k faktu, že tato práce si neklade za cíl vyhledání kódu v obraze, a tedy předpokládá vstup přesných obrázků 2D kódů, je zvolena nejjednodušší možná implementace prahování. Prahovací hodnota bude zvolena pevně na základě provedených pokusů. Pokud by se v budoucnu tato knihovna rozšiřovala o detekci kódů v obraze, bude nutné implementovat i lepší způsob prahování.

3. *Nalezení hranic kódu.* Tato práce si neklade za cíl detekci kódu v obraze. Fáze tedy bude pracovat velmi jednoduchým způsobem. Za hranici kódu budou považovány první výskyty černých bodů (po prahování) ze všech 4 stran kódu. Čtení 2D kódů tímto přichází o možnost číst kód z reálných fotografií a omezuje se pouze na synteticky vytvořené přesné obrázky.

4. *Usazení čtecí mřížky.* Správné usazení čtecí mřížky je velmi důležité pro správné načtení kódu, je třeba mu věnovat zvláštní pozornost. Po nalezení hranic je potřeba zjistit, kolik řádků a sloupců bude mít čtecí mřížka. Různé kódy nabízejí různé způsoby, toto tedy bude závislé na typu čteného kódu.

5. *Zjištení metainformací.* Některé 2D kódy obsahují i metainformace potřebné pro korektní přečtení dat. Například QR kód obsahuje informaci o použité masce, nastavení algoritmu Reed-Solomon a někdy i o samotné velikosti kódu. Po načtení metainformací se ještě může zpřesnit usazení čtecí mřížky.

6. *Čtení dat.* Přečtení dat z datové oblasti kódu. Běžně by se během čtení prováděla zároveň oprava dat pomocí kontrolních součtů. Implementace dekódování samoopravného kódu Reed-Solomon už překračuje rozsah a cíle této práce, proto tato knihovna data z 2D kódů nekontroluje. Data je během čtení potřeba upravovat, například QR kód může mít data zpřeházená (viz sekce 2.2.3). Během této fáze je sestaven binární řetězec a ten je dále dekódován v následující fázi.

7. *Dekódování dat.* U 2D kódů je definováno několik způsobů jak lze data kódovat. Během této fáze se tedy načtená binární data dekódují na znaky podle definic jednotlivých kódů.

Skutečná implementace čtení 2D kódů je popsána v sekci 4.3 na straně 18.

3.3 Společné části knihovny

Hlavní hledisko při návrhu knihovny je možnost relativně snadné výměny kterékoliv fáze implementace. Struktura knihovny tak bude vypadat, jako by byla složena z jednotlivých modulů. Pro jednoduchost použití bude dostupná i společná část, které uživatel předá obrázek a typ kódu, tato funkce poskládá jednotlivé moduly doporučeným způsobem a daný typ kódu přečte. Zároveň bude uživateli knihovny ponechána možnost poskládat si moduly

samostatně, s možností lepší konfigurace. Modulární přístup ponechává i možnost přepsat implementaci některé části a její snadné zapojení do struktury knihovny. Případný uživatel knihovny si tedy může čtení kódu částečně upravit pro konkrétní situaci. Například by mělo být možné vyměnit implementaci prahování. Nicméně zásahy na úrovni výměny implementací velmi pravděpodobně budou vyžadovat detailní znalost struktury knihovny. Z toho důvodu pravděpodobně uživatelé nebudou příliš zasahovat do použitých implementací.

Knihovna by také měla umožňit získat výčet implementovaných typů kódů. Jejich implementace budou mít jednotné rozhraní, aby bylo usnadněno použití. Přidávání dalších typů čárových kódů by mělo být poměrně snadné s ohledem na případná budoucí vylepšení a rozšíření.

3.4 Ukázková aplikace

Mimo popisované knihovny, která je bezesporu hlavním cílem práce, je vhodné vytvořit také jednoduchou aplikaci. Aplikace bude používat vytvořenou knihovnu a demonstrovat její možnosti. Ukázková aplikace bude také sloužit k ladění a testování v průběhu vývoje knihovny. Aplikace by měla být velmi jednoduchá a měla by demonstrovat jednoduchost použití knihovny. Kód ukázkové aplikace by měl být přísně oddělen od kódu knihovny.

Aplikace by měla umožnit výběr a načtení obrázku z počítače a jeho zobrazení. Následně uživatel vybere typ čárového kódu a stisknutím tlačítka zahájí čtení. Aplikace pak předá obrázek a vybraný typ kódu knihovně a po přečtení zobrazí uživateli data z přečteného kódu. Pokud čtení skončí chybou, zobrazí jednoduché chybové hlášení, že se nepodařilo načíst kód z obrázku.

3.5 Implementační prostředky

Vzhledem k navržené variabilitě by bylo vhodné zvolit objektový programovací jazyk, aby bylo možné použít řadu návrhových vzorů [6]. Důležité také bude, zda základní knihovny jazyka umožňují snadnou práci s obrázky. Nutnost připojovat knihovny třetích stran by celé použití komplikovala.

Byly zvažovány dva vhodné jazyky. Prvním je C++, což je velmi rozšířený jazyk, je pro něj dostupných mnoho různých vývojových prostředí, podpora práce s obrázky je však na velmi nízké úrovni a vyžadovala by použití knihovny třetí strany. Proto byla nakonec zvolena druhá varianta, jazyk Java. Jedná se o jazyk s dobře propracovaným objektovým modelem a jsou pro něj zdarma dostupná velmi kvalitní vývojová prostředí. Podpora práce s obrázky je obsažena v základních knihovnách a umí číst i zapisovat nejrozšířenější formáty. Navíc lze do platformy jazyka zařadit podporu dalších formátů obrázků bez nutnosti nové kompilace programu. Jazyk má velmi dobrou přenostitelnost, je podporován na všech rozšířených platformách, například v prostředí Windows, Linux i Solaris. Pro přenos na jinou platformu není třeba nová kompilace zdrojových kódů. Nevýhodou může být podstata jazyka, jedná se o částečně interpretovaný jazyk, k běhu je tedy třeba mít instalované příslušné prostředí a může trpět menší rychlostí provádění kódu. Interprety jazyka Java jsou však již natolik rozšířené, že se setkáváme s problémy spíše výjimečně. Bude použita momentálně nejnovější stabilní verze jazyka Java SE 6[1]. Použití jazyka Java činí knihovnu nezávislou na operačním systému nebo typu procesoru.

[1] `http://www.oracle.com/technetwork/java/javase/overview/index.html`

Kapitola 4

Popis implementace

V této části bude popsána implementace knihovny a její výsledná struktura. Jak již bylo napsáno v sekci 3.5, je celá knihovna implementována v jazyce Java. Popis se zaměří na jednotlivé fáze činností definované v kapitole 3, s důrazem na zajímavé nebo netradičně řešené oblasti. Popisy tříd se budou zaměřovat na princip funkce. Více informací o metodách, konstruktorech a jejich parametrech lze najít v programové dokumentaci na přiloženém CD. Pokud v knihovně existuje více použitelných implementací, bude uvedeno, jaká třída je výchozí pro určitou fázi a typ kódu.

4.1 Implementované typy kódů

Z 1D kódů jsou implementovány typy UPC, EAN-13, Code 128 a Code 25. Kód EAN-13 obsahuje i korekce deformace kódu a hodí se tedy pro čtení ze syntetických obrázků i reálných fotografií. Kód UPC používá implementaci kódu EAN-13 (viz sekce 2.1.4), platí pro něj tedy stejné podmínky. Code 25 a Code 128 jsou na tom hůře. Korekce deformací nebyla implementována a čtení je tím předurčeno pouze pro čtení synteticky vytvořených obrázků. V případě Code 25 se navíc projevuje problém různých startovacích sekvencí, kódy z některých generátorů jsou tedy pro knihovnu nečitelné.

U 2D kódů je implementován jediný zástupce, je jím QR kód. Čtení kódu je opět zamýšleno pouze pro syntetické obrázky. U ostatních kódů se bohužel nepodařilo včas zajistit kvalitní materiály, podle kterých by se dalo čtení implementovat.

4.2 Implementace čtení 1D kódů

Implementace se drží návrhu (sekce 3.1), pouze došlo ke spojení některých fází do jedné třídy. Čtení je děleno na 3 hlavní celky, pro každý celek existuje rozhraní:

1. Získání vzorku dat – rozhraní `Sampler1D`.
2. Dekódování dat – rozhraní `Decoder1D`.
3. Ověření kontrolního součtu – rozhraní `Checksum`.

Jednotlivé celky spojuje dohromady třída `ReaderUniversal1D`. Ta vyžaduje v konstruktoru předání implementací uvedených rozhraní, ty pak používá pro čtení kódu z obrázku. Výjimkou je ověření kontrolního součtu. Některé kódy nemají kontrolní součet definován, a proto se tato implementace uvádět nemusí. Výsledek načtení pak není kontrolován.

Uživatel knihovny dostává také možnost dodat svoji implementaci některého z rozhraní a tím pozměnit nebo vylepšit funkci. Předpoklad ovšem je, že uživatel knihovny bude používat již vytvořené třídy dodané s knihovnou.

Nyní se blíže podíváme na jednotlivé části.

4.2.1 Získání vzorku dat

Z hlediska fází čtení definovaných v návrhu (sekce 3.1) pokrývá tato část fáze od separace čtecí linky až po prahování. Je definována rozhraním `Sampler1D`.

Jediná implementace dostupná v knihovně je třída `LineSampler`. Ta řeší získání vzorku dat výběrem jednoho řádku pixelů ze vstupního obrázku. Výběr řádku je závislý na počtu požadovaných vzorků. Pokud je požadován jeden vzorek, je výsledný řádek zvolen přesně v polovině obrázku. Pokud je zvoleno více vzorků, jsou rovnoměrně rozloženy.

Převod na odstíny šedi a prahování pak třída `LineSampler` přímo neřeší. Za tímto účelem požaduje předání implementace rozhraní `Binarizer1D`. Předané třídě pak svěří celý proces převodu na odstíny šedí i prahování. Zde existují dvě implementace tohoto rozhraní a tím dva způsoby prahování. Je doporučeno používat spíše třídu `kmeans`.

Třída `SimpleTreshold` provádí jednoduchý způsob prahování. Jednoduše v konstruktoru požaduje zadání pevné prahovací hodnoty a pomocí této hodnoty pak rozhoduje o výsledku prahování. Řešení je to velmi rychlé, ale nepohodlné. Vybraná prahovací hodnota může velmi ovlivnit výsledek a tím i úspěšnost čtení. Třída má sloužit pouze jako alternativa k následující implementaci prahování.

Třída `kmeans` sice stále provádí prahování jednou hodnotou stejnou pro celý obrázek, ale tuto hodnotu se snaží na počátku vhodně zvolit. Jak už bylo napsáno v návrhu prahování (sekce 3.1), používá k výběru hodnoty algoritmus k-means [12]. Zjednodušeně řečeno, algoritmus je definován tak, že na počátku vybere z dat dva náhodné vzorky. Tyto vzorky se stávají středy skupin. Ostatní vzorky se rozdělí do těchto skupin podle vzdálenosti k danému středu. V dalších iteracích se vždy spočítá střed nalezené skupiny a všechny vzorky se opět rozdělí do skupin. Ve chvíli, kdy rozdělení skončí dvakrát za sebou shodně, je rozdělení považováno za hotové.

Z popisu algoritmu k-means je jasné, že čím lépe se podaří vybrat počáteční středy, tím méně iterací bude třeba. Proto navržená implementace nepoužívá náhodně vybrané vzorky, ale pro počáteční body používá největší a nejmenší hodnotu ve vstupních datech. Praktické vyzkoušení tohoto výběru ukazuje akceptovatelný počet iterací výpočtu. U běžných obrázků se pohybuje v jednotkách iterací.

Drobná změna oproti definici je také ve výstupu algoritmu. Obvyklé by bylo výslednou prahovou hodnotu určit podle aritmetického průměru obou zjištěných středů. Jak se při praktických pokusech ukázalo, pro prahování za účelem čtení čárových kódů to není vhodné. Metodou postupných pokusů bylo zvoleno dělení vzdálenosti mezi středy v poměru 7:3, přičemž větší část patří bílé barvě. Je zajímavé, že tento poměr je podobný očekávanému poměru mezi bílou a černou barvou v kódu EAN-13, nicméně souvislost je pravděpodobně pouze náhodná. Praktické výsledky určení prahové hodnoty algoritmem k-means lze najít v části 5.1.

Část získání vzorku dat a jeho prahování je společná pro všechny typy 1D kódů. Prahování s využitím k-means je tedy dostupné pro všechny typy 1D kódů a ve výchozím nastavení knihovny se také používá pro všechny typy.

4.2.2 Dekódování dat

Dekódování dat vzhledem k návrhu (sekce 3.1) pokrývá fáze vyhledání startovací sekvence a dekódování. Dekodéry pro 1D kódy jsou dány rozhraním `Decoder1D`, každý typ kódu pak má vlastní implementaci dekodéru.

Hledání startovací sekvence fuguje přesně tak, jak bylo popsáno v návrhu. Ještě před vlastní hledání začátku kódu byla doplněna malá pomocná třída `PrevodnikDelek`, která funguje jako mezivrstva pro čtení vstupních dat. Původní vstupní data jsou tvořena sekvencemi 0 a 1. Pro dekodér je však vhodnější mít infomaci např. černá oblast pět pixelů dlouhá. Přesně takovou informaci má za úkol vytvářet tato pomocná třída. Pouze čte vstupní data a počítá šířky souvislých oblastí. Zároveň provádí i potlačení šumu, které ovšem není příliš propracované a zvládá odfiltrovat jen malou úroveň šumu.

Vlastní čtení dat již má informaci o šírce modulu od hledání startovací sekvence. Pak stačí jen číst šířky souvislých oblastí, dělit je velikostí modulu a v kódovacích tabulkách vyhledávat přečtené znaky. Pokud načtená kombinace čar není v kódovacích tabulkách, je kód prohlášen za nečitelný a zpracování končí výjimkou `BCLReadError`.

Bohužel při praktickém použití nastaly dva problémy. První problém je s rozmazanými fotografiemi, kdy se po prahování obvykle stávají černé oblasti širšími, než by měly být. Druhý problém je s defomovanými kódy, například na lahvích. Jakmile je kód vylepen na lahvi, získává tvar části válce. Při snímání takového kódu pak dojde k deformaci způsobené projekcí takového kódu do roviny. Vlivem deformace se fakticky mění velikost modulu v průběhu čtení kódu. Oba problémy se snaží řešit následující opatření.

Opatření proti deformacím kódu musí být velmi úzce propojeno s dekodérem. Implementováno je pouze v rámci čtení kódu EAN-13, do ostatních typů nebylo přeneseno. Pokud jsou po prahování černé oblasti příliš široké a bílé naopak příliš úzké, lze sledovat průběžnou chybu. Sledování průběžné chyby funguje tak, že se po vydělení velikostí modulu sleduje rozdíl mezi očekávanou šířkou a skutečnou šířkou. Tento rozdíl se ukládá jako průběžná chyba takovým způsobem, aby byl zachován vliv měření u predchozích čtení v rámci čtení jednoho kódu. Způsob výpočtu průběžné chyby vysvětluje rovnice 4.1, kde E_i je nová průběžná chyba, E_{i-1} je stávající průběžná chyba a R_i je aktuální rozdíl oproti očekávané hodnotě. Jak je vidět, vliv předchozích měření bude postupně slábnout a naopak jedna výrazná náhodná chyba nemůže průběžnou chybu příliš ovlivnit.

$$E_i = \frac{E_{i-1}}{2} + \frac{R_i}{2} \tag{4.1}$$

Průběžná chyba se sleduje zvlášť pro bílé oblasti a zvlášť pro černé oblasti. To umožňuje kompenzovat chyby vzniklé rozmazaným vstupním obrázkem. Uplatnění takto sledované průběžné chyby je už jednoduché. Ze změřené délky souvislé oblasti je odečtena sledovaná průběžná chyba a teprve potom je výsledek dělen velikostí modulu. Tato kompenzace deformací dobře zvládá deformace, které jsou plynulé, například kód nalepený na láhvi. Naopak není vhodná pro náhlé a krátké deformace vzniklé třeba zmačkáním podkladu. Opatření proti deformacím výrazně vylepšilo schopnosti čtení kódu EAN-13 z reálných fotografií, viz sekce 5.2. Korekce deformací do ostatních typů kódů nebyla přenesena, a tak se ostatní implementace příliš nehodí ke čtení fotografií.

Následuje seznam implementací dekodérů pro různé typy 1D kódů.

- UPC – `EAN13Decoder`
- EAN-13 – `EAN13Decoder`

- Code 25 – `Code25Decoder`
- Code 128 – `Code128Decoder`

4.2.3 Ověření kontrolního součtu

Třídy pro výpočty kontrolních součtů jsou definovány rozhraním `Checksum`. Kontrolní součty se počítají z již převedených dat a tím odpadají problémy se zpracováním nepřesných vstupů. Způsoby výpočtu kontrolních součtů jsou dobře definovány ve specifikacích. Jednotlivé implementace pak pouze vypočítají kontrolní součet a porovnají ho s kontrolním součtem načteným z kódu. Pokud se liší, je kód chybně načten a zpracování končí výjimkou `BCLChecksumError`. Výjimka `BCLChecksumError` byla zavedena speciálně pro hlášení chyb vzniklých až při kontrolách. Uživatel knihovny s takovou informací může dále pracovat lépe než s obecnou chybou čtení. Výjimka chyby kontrolního součtu je potomkem výjimky `BCLReadError`, takže pokud uživatel nechce chyby kontrolního součtu zpracovávat samostatně, nemusí. Výjimku lze zpracovávat jako obecnou chybu čtení.

4.3 Implementace čtení 2D kódů

Čtení 2D kódů se svými principy výrazně odlišuje od čtení 1D kódů. Jsou zde odlišné problémy a požadavky. Definice 2D kódů jsou mnohem složitější, stejně jako jejich možnosti. Bohužel se nepodařilo zajistit kvalitní specifikace kódů typu Data Matrix a PDF 417. Tyto typy tedy nakonec nejsou implementovány a v kategorii 2D kódů zbývá jediný zástupce, typ QR Code. Následující popis implementace se tedy týká pouze typu *QR Code*.

Výsledná implementace se opět drží rámcového návrhu v sekci 3.2. V knihovně je základní třída `ReaderQR` implementující čtení QR kódu. Protože je čtení 2D kódů obecně složitější jak u 1D, je zde také více pomocných tříd, které základní třída využívá.

4.3.1 Přípravné činnosti

Přípravné činnosti před čtením QR kódu zahrnují z hlediska návrhu všechny fáze od převodu na stupně šedi až po zjištění metainformací. Při těchto činnostech základní třída (`ReaderQR`) spolupracuje s dalšími pomocnými třídami.

První takovou pomocnou třídou je `Binarizer2D`. Spojuje v sobě převod na úrovně šedi, prahování a implementuje čtecí mřížku. Vhledem k určení implementace čtení QR kódu pouze pro syntetické obrázky je prahování provedeno pevně nastavenou hodnotou. Použití algoritmu k-means pro syntetické obrázky by ztrácelo smysl. Celková činnost třídy je následující. Usazení čtecí mřížky se provádí v konstruktoru objektu, ale je možné do usazení zasahovat i později, v průběhu práce s mřížkou. Po usazení čtecí mřížky na obrázek pak třída provádí přepočet mezi souřadnicemi modulu (čtecí řádek, čtecí sloupec) a souřadnicemi pixelu v obrázku. Pixel, který odpovídá danému řádku a sloupci, je přečten a prahován. Současná implementace čtecí mřížky umožňuje umístění čtecích řádků a sloupců pouze rovnoběžně s osami obrazu, nelze tedy číst natočené kódy.

Správné nastavení objektu `Binarizer2D`, a tím správné umístění čtecí mřížky, má na starosti opět základní třída `ReaderQR`. Protože tato práce se nevěnuje vyhledáním kódů v obraze, je vyhledání kódu a tím i umístění čtecí mřížky provedeno zjednodušeným způsobem. Umístění spočívá ve vyhledání černého pixelu který je nejblíže levému okraji vstupního obrázku. Tento pixel je pak považován za hranici kódu. Stejným způsobem se vyhledají hrany kódu pro zbývající okraje vstupního obrázku. QR kód tedy nemusí vyplňovat celou

plochu vstupního obrázku, může mít okraje, které ale musí být celé bílé. V opačném případě dojde k chybnému umístění čtecí mřížky a tím chybám při čtení. Jakmile jsou takto zjištěny hranice QR kódu, je spuštěn odhad velikosti modulu.

Velikost modulu se odhaduje z velikosti zaměrovacího čtverce umístěného v levém horním rohu kódu. Stačí spočítat počet černých pixelů v řadě od levého horního rohu kódu směrem vpravo. Podle specifikace QR kódu (sekce 2.2.3) je pak jasné, že tato vzdálenost má mít 7 modulů. Velikost modulu je třeba k odhadu velikostní verze QR kódu. U menších verzí (verze 1 až 6) musí tento odhad stačit k dalšímu čtení. U větších verzí (od verze 7) specifikace předepisuje speciální systémovou oblast v ploše kódu, kde je zakódováno správné číslo verze. Číslo verze je třeba správně dekódovat a podle něj vypočítat počet čtecích řádků a sloupců.

Jak bylo uvedeno v popisu kódu (sekce 2.2.3), je informace o verzi kódu tvořena 18-ti bity, z nichž je 6 bitů s informací o verzi a zbývajících 12 bitů je kontrolní součet. Kontrolní součty jsou počítané opět alogitmem Reed-Solomon. Jeho dekódování přesahuje rozsah této práce, ale v případě informace o verzi kódu se dá obejít jiným způsobem.

Velikostních verzí je definováno 40, číslo verze se však vkládá do kódu jen u 34 verzí (verze 7 až 40). V oblasti pro zápis verze kódu se tak může nacházet pouze 34 variant informace. Při tomto počtu je možné mít v programu předpočítanou tabulku všech povolených hodnot. Při dekódování informace o verzi pak stačí pouze porovnat přečtenou informaci s jednotlivými dovolenými možnostmi v tabulce. Při každém porovnání se vypočítá, o kolik bitů se hodnoty liší. Pokud se od některé dovolené možnosti načtená informace liší o méně než než 4 bity, je použita tato dovolená možnost. Tím je možné načíst a zkontrolovat načtení informace o verzi a je dokonce možné opravit malou chybu čtení.

Po zjištění skutečné velikosti se vypočítá počet čtecích sloupců a řádků, jimiž se upraví nastavení čtecí mřížky. Tím je mřížka správně umístěna a je plně připravena pro čtení. Informace o verzi kódu není poslední infromací, kterou je třeba získat. Ještě důležitější je formátovací informace, která říká, jaká maska byla na kód aplikována a jaká je použitá úroveň zabezpečení dat proti chybám. Informace o formátu má 15 bitů, z nichž je 5 datových a 10 s kontrolním součtem. 5 datových bitů se pak ještě dělí na 2 bity indikace ochrany dat a 3 bity s číslem použité masky. Zatímco informaci o verzi by bylo možné odhadnout, formátovací informace jsou pro další čtení naprosto zásadní. Pokud se nepodaří dekódovat informaci o formátu, není možné pokračovat ve čtení. Dekódování formátovacích informací probíhá na stejném principu jako u informace o verzi. Knihovna je tedy schopna zkontrolovat správnost a také opravit malou chybu čtení.

Uchování metainformací o načítaném kódu má v knihovně na starost třída `QRCodeInfo`. Třídě byly svěřeny i metody pro dekódování načtených informací. Základní třída `ReaderQR` tedy po umístění čtecí mřížky načte formátovací informaci a informaci o verzi. Ty stále ještě v binární podobě předá do třídy `QRCodeInfo`, která se postará o dekódování a uchování metainformací. Nyní je čtecí mřížka umístěna a máme všechny potřebné informace ke čtení datových oblastí.

4.3.2 Čtení a dekódování dat

Řešení celého čtení i dekódování dat v jediné třídě, jako je tomu v případě 1D kódů, by bylo zbytečně složité a nepřehledné. Implementace tedy dostala strukturu připomínající vrstvy. Každá vrstva zpracování dat je řešena jinou třídou v knihovně.

První vrstva je implementována třídou `QRByteStream`. Jak název napovídá, jejím výstupem je proud bajtů. Ke své činnosti potřebuje jak čtecí mřížku, tak metainformace o kódu.

Třída má za úkol postupně číst jednotlivé bajty z kódu. Při čtení zároveň ze čtených bitů odstraňuje použitou masku. Čtení jednotlivých bajtů znamená, že implementuje algoritmus umísťování bajtů v QR kódu uvedený v normě [1]. Bohužel většinou nestačí pouze přečíst tuto řadu bajtů. Bajty jsou ve většině verzí kódu zpřeházené.

Jak bylo uvedeno v popisu QR kódu (sekce 2.2.3), jsou data rozdělena na určitý počet bloků a bloky se do kódu umísťují střídavě. Třída `QRByteStream` si z metainformací o kódu vypočítá, kolik bloků a jakých velikostí bylo použito, a během čtení data uspořádá do správné posloupnosti. Velikosti a počty bloků závisí jak na velikostní verzi, tak na použité ochraně dat. Možných kombinací je poměrně hodně ($40 \cdot 4 = 160$), přesto je potřeba řešit získání počtu bloků částečně vyhledáváním hodnot v tabulkách. I v normě kódu je pro počty a velikosti bloků uvedena dlouhá tabulka. Logicky by se zde měla provádět kontrola a oprava dat. Implementace algoritmu Reed-Solomon nebyla provedena, data tedy nejsou kontrolována na správnost a kontrolní součty se vůbec nečtou. V případě poškození vstupního obrázku pak může dojít k chybné interpretaci binárních dat. Třída `QRByteStream` implementuje stejné rozhraní, jaké používá Java na čtení z binárních souborů. Od této chvíle se s daty z QR kódu pracuje stejně jako s otevřeným souborem. Kód je tím přehlednější a pochopitelnější.

Na výstupu ze třídy `QRByteStream` jsou již jednotlivé bajty ve správném pořadí, nicméně se stále jedná o binární data, nikoliv o znaky. Zde začíná další vrstva zpracování, ta má za úkol interpretovat binární data a převést je na znaky. Je implementována třídou `QRDataDecoder`. Ta čte binární data z předchozí vrstvy, třídy `QRByteStream`.

Vstupní binární data mají svůj jednoduchý protokol. Na začátku dat jsou 4 bity identifikující režim. Na těchto režimech je dobře patrné, že QR kód vznikal v Japonsku. Kromě obvyklých režimů jako jsou numerický a alfanumerický obsahuje i režim kódující japonské Kanji znaky. Norma tento režim sice označuje jako povinný, ale má knihovna jej z logických důvodů neimplementuje. Stejně tak režim 8-bit má kódovací tabulku, která obsahuje část Kanji znaků. Spodní polovina tabulky je shodná s ASCII tabulkou, a tak knihovna tento režim implementuje plně jako ASCII. Další režimy, numerický a alfanumerický, již neobsahují japonské znaky a jsou implementovány v plném rozsahu. Norma připouští i režim nazvaný ECI. Ten je však označen jako nepovinný a není implementován.

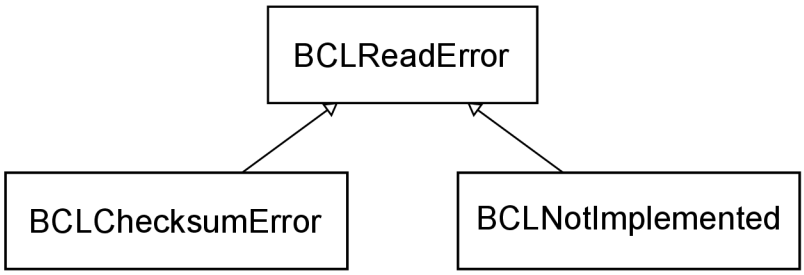
S vysvětlením funkce dekódování se tedy zaměříme na režimy numerický, alfanumerický a 8-bit. Za čtyřmi bity indentifikujícími konkrétní režim následuje informace o počtu znaků. Část s počtem znaků může mít různou délku, ta je určena jak zvoleným režimem, tak velikostní verzí kódu a pohybuje se v rozsahu od 8 bitů po 16 bitů. Po zjištění počtu znaků ve zvoleném režimu již lze přistopit k vlastnímu dekódování jednotlivých znaků podle kódovacích tabulek. Po dekódování daného počtu znaků následují opět 4 bity identifikující další režim a vše se opakuje. Jeden kód tedy může použít více režimů kódování. Podle normy má generátor kódu režimy střídat za účelem maximálního zmenšení výsledného kódu. Pokud je číslo režimu 0000, jedná se o konec dat a zpracování končí. Drobné komplikace přináší i malá poznámka v normě, která říká, že pokud do konce oblasti vyhrazené pro data zbývá méně jak 4 bity, a tudíž nelze bez zvětšení kódu ukončit data, nemusí se ukončení uvádět.

Na výstupu třídy `QRDataDecoder` je již proud dekódovaných znaků. Protože dekodér neimplementuje úplně všechny dekódovací režimy, je možné za pomoci třídy `ReaderQR` získat i binární proud dat. Uživatel knihovny tak může použít i vlastní dekodér nebo používat obsah kódu nestandardním způsobem. Nestandardní způsob může být třeba nějaký uzavřený systém, který bude QR kód používat k přenášení binárních dat.

4.4 Implementace společných částí knihovny

V této části si popíšeme nejpodstatnější část celé knihovny. Tou je třída `BarCode` a její metoda `read`. Tato metoda je vlastně to jediné, co musí uživatel knihovny použít. V parametrech očekává vstupní obrázek kódu a určení typu kódu. Vrací pak řetězec s dekódovanými znaky nebo končí výjimkou. Drtivá většina použití knihovny se pravděpodobně bude omezovat pouze na zmíněnou metodu. Nelze ji použít pouze v případě, že není zamýšleno použití výchozích implementací v knihovně nebo je třeba provést pouze část čtení kódu. Typ čteného kódu se určuje jako jedna z konstant výčtů `BarCodeType1D` a `BarCodeType2D`.

Za zmínku ještě stojí systém vyvolávaných výjimek. Jedná se v zásadě o 3 výjimky. Jejich struktura je na obrázku 4.1. Hlavní je výjimka `BCLReadError` a z té dědí výjimky `BCLChecksumError` a `BCLNotImplemented`. Díky chování jazyka Java stačí odchytávat výjimku `BCLReadError`. Ta vždy značí nečitelný kód. Pokud chce uživatel knihovny znát bližší podrobnosti o problému, může zachytávat i zbývající dvě výjimky. `BCLChecksumError` znamená sice načtený a dekódovaný kód, ale nesouhlastí kontrolní součet. Je tedy pravděpodobně chybně přečten. `BCLNotImplemented` může nastat pouze při čtení z QR kódu a značí, že QR kód používá k zakódování dat některý z režimů, který není podporován dekodérem.



Obrázek 4.1: Struktura výjimek v knihovně.

4.5 Shrnutí požadavků na vstupní obraz

Většina požadavků na vstupní obraz již byla zmíněná v kapitolách návrhu (3) a implementace (4) i s odůvodněním. Nyní budou všechny požadavky a omezení přehledně shrnuty.

4.5.1 1D kódy

- *Dostatečné rozlišení.* Syntetické obrázky musí mít rozlišení alespoň 1 pixel/modul kódu. V případě reálné fotografie (pouze EAN-13) více, podle kvality fotografie.
- *Vhodné okolí kódu.* Pokud kód nezaplňuje celý obrázek, nesmí být okolo nic, co by mohlo zmást algoritmus hledání začátku kódu. Zvláště pak na levé straně.
- *Správné natočení.* Knihovna není schopna otáčení obrázků s kódem. Kód musí být vždy čitelný zleva doprava. Obraz může být lehce natočen, pokud vodorovná čtecí linka protne všechny čáry kódu.
- *Podporovaná startovací sekvence.* Platí pouze u kódu typu Code 25, u ostatních se problém nevyskytuje. Různé dostupné generátory Code 25 používají dvě různé startovací sekvence. Podporována je pouze jedna, bližší podrobnosti jsou v sekci 2.1.1.

V případě kódu typu EAN-13 je implementováno více korekcí a tím je implementace vhodná i pro čtení reálných fotografií. Ostatní typy čárových kódů předpokládají pouze generované přesné obrázky, nikoliv fotografie. Vždy platí, že kód by měl zabírat většinu plochy vstupního obrázku.

4.5.2 2D kódy

Protože je implementovám pouze QR Code, týkají se omezení přirozeně pouze tohoto typu. Čtení předpokládá pouze vygenerované obrázky, algoritmy nejsou uzpůsobeny čtení fotografií.

- *Dostatečné rozlišení.* Na 1 modul v kódu musí připadat alespoň 4 pixely, tj. modul čtvercového tvaru se stranou čtverce alespoň 2 pixely.
- *Správné natočení.* Knihovna není schopna detekovat otočení kódu. Vstupní obrázek musí mít kód natočen správným způsobem. To znamená, že hlavní 3 zaměřovací čtverce jsou umístěny v levém dolním, levém horním a pravém horním rohu. Hrany kódu musí být rovnoběžné s hranami obrázku.
- *Vhodné okolí kódu.* Pokud kód nezaplňuje celou plochu obrazu, nesmí být ve zbytku obrazu nic jiného než bílá barva.

Implementace 2D kódů neprování kontrolu a opravy načítané informace. Pokud dojde k chybě při čtení dat, projeví se tato chyba i ve výstupních datech. Při čtení generovaných syntetických obrázků by chyby čtení neměly nastat.

4.6 Implementace ukázkové aplikace

Výsledná ukázková aplikace se drží návrhu (sekce 3.4) a snaží se dodržet navrženou jednoduchost. Aplikace používá jednoduché grafické prostředí. Při startu programu si z knihovny vyžádá seznam implementovaných typů kódů a připraví jejich seznam pro uživatele. Z tohoto seznamu pak uživatel vybírá načítaný typ kódu. Aplikace umožňuje otevření souboru s obrázkem, výběr typu kódu a zobrazení výsledku čtení. Aplikaci tvoří jediná třída. Při spuštění vytváří jeden textový soubor, do kterého jsou vypisovány ladící informace z aplikace i knihovny. Soubor se záznamem ladících informací se při každém spuštění přepisuje.

Aplikace je opravdu určená pouze k demonstraci funkčnosti knihovny, nikoliv k reálnému nasazení. Pro své spuštění samozřejmě vyžaduje přeloženou knihovnu. Jednoduchý manuál k použití aplikace se nachází v příloze B.

Kapitola 5

Dosažené výsledky

V této kapitole budou diskutovány výsledky dosahované knihovnou při čtení různých typů čárových kódů. Pozornost bude věnována také výsledkům prahování s použitím algoritmu k-means (viz sekce 4.2.1).

5.1 Výsledky prahování

Jak už bylo řečeno v návrhu čtení 1D kódů (sekce 3.1) i v popisu implementace (sekce 4.2.1), je pro odhad vhodné prahovací hodnoty použitý algoritmus k-means. V průběhu testování se ukázalo, že je třeba provést drobné úpravy na výstupu algoritmu (viz sekce 4.2.1). Výsledky jsou prezentovány již se všemi popsanými úpravami.

Na obrázku 5.1 je zobrazena část výsledku prahovací operace během čtení kódu z reálné fotografie. Horní polovina obrázku představuje vstup před prahováním v odstínech šedé, spodní polovina pak výstup prahování. Vstup zachycuje čtecí linku, která prochází přes část 1D čárového kódu. Tmavé a světlé oblasti jsou čáry kódu a mezery mezi nimi. Lze si snadno všimnout rozmazaných okrajů čar. Rozmazání je typické pro reálné fotografie, zvláště pokud nejsou focené za ideálních světelných podmínek. Vstupní obrázek byl poměrně tmavý, což je vidět na světlých částech vstupu, které jsou výrazně šedivé, nikoliv bílé. Úkolem prahování je rozhodnout, která část obrázku paří černé čáře v kódu a která část obrázku patří bílé mezeře. To je nejsložitější právě v rozmazaných okrajích oblastí. Algoritmus k-means se snaží najít takovou prahovou hodnotu, která se bude nacházet asi uprostřed rozmazaného přechodu. Na výstupu prahování ve spodní polovině obrázku již jsou vidět identifikované černé a bílé oblasti.

Obrázek 5.1: Výsledek prahování.

Kvalita prahování je důležitá hlavně pro reálné fotografie pořízené za špatných světelných podmínek. Pak se začnou více projevovat rozmazané okraje čar a čtení čárového kódu je tím méně spolehlivé. Na obrázku 5.1 je vyznačena oblast uprostřed. Jedná se o oblast, která v původním obrázku patří střední dělicí části kódu typu EAN-13. Tato oblast se skládá ze 2 čar širokých 1 modul obklopených stejně širokými mezerami. Jak je vidět

z výsledku prahování, jsou velikosti čar i mezer téměř stejné. Prahování tedy určilo správné hranice čar, přestože byly rozmazané.

5.2 Výsledky čtení 1D kódů

Pro testování spolehlivosti čtení byla vytvořena sada testovacích obrázků. S výjimkou kódu typu EAN-13 se jedná pouze o syntetické obrázky. Jednotlivé obrázky se snaží pokrýt většinu ze zakódovatelných znaků, aby odhalily případné překlepy v kódovacích tabulkách programu. Pro kód EAN-13 bylo vyfotografováno několik kódů na různých výrobcích. Každá fotografie má svůj účel a ověřuje nějakou vlastnost knihovny. Na přiloženém CD jsou jednotlivé obrázky rozděleny na čitelné a nečitelné. V následujícícm popisu se zaměříme hlavně na ty nečitelné, kde bude vždy okomentována problémová vlastnost, která brání načtení.

Mezi nečitelnými kódy jsou zařazeny čtyři obrázky. Dva obrázky zobrazují kód typu Code 25 a dochází při jejich čtení ke stejnému problému. Oba používají startovací sekvenci, kterou implementovaná knihovna nepodporuje. Při zvětšení obrázků a porovnání šířek čar ve startovací sekvenci a v těle kódu je jasně vidět, že startovací sekvence používá širší části ve velikosti 3 moduly. Knihovna předpokládá jinou startovací sekvenci, viz sekce 2.1.1. Právě kvůli nepodporované startovací sekvenci je implementovaná knihovna nedokáže načíst.

V případě kódu EAN-13 jsou mezi nečitelnými obrázky zařazeny dvě fotografie. První z nich, nazvaná `ean13-1N.jpg`, má neobvyklý problém s pozadím kódu. Pokud si obrázek zvětšíme, zjistíme že pozadí kódu tvoří jemné samostatné barevné tečky. S takovýmto pozadím si knihovna neporadí. Implementace prahování považuje jednotlivé tečky za černé oblasti a ve fázi dekódování čtení končí chybou. Pro zajímavost bylo pro jeden test vyzkoušeno dočasné vypnutí korekce deformací (viz sekce 4.2.2). S vypnutou korekcí deformací knihovna nebyla schopna načíst další dvě fotografie z testovací sady. Jsou to právě fotografie kódu na nerovném povrchu. Lze tedy říct, že implementovaná korekce deformací má také svou nezanedbatelnou zásluhu na schopnostech dekódování z reálných fotografií.

Druhá fotografie s názvem `ean13-2N.jpg`, vypadá na první pohled bez vad. Kód je sice lehce deformován nalepením na lahev, nicméně deformace není veliká. S tímto by si měly poradit korekce deformací při čtení kódu. Problém se však vyskytuje mnohem dříve. Hned ve startovací sekvenci jsou čáry oproti mezeře moc úzké. Šířka čáry i mezery mezi nimi by měly být shodné. Zde je po prahování velikost mezery výrazně větší a tím knihovna při čtení kódu chybně detekuje jeho začátek. Začátek je detekován mnohem dál, až v oblasti dat. Následné čtení pak nutně musí skončit chybou.

5.3 Výsledky čtení 2D kódů

Ve třídě 2D kódů knihovna podporuje pouze čtení typu QR Code. Testování se tedy omezilo pouze na tento jediný typ. Pro testování čtení byla vytvořena sada testovacích obrázků. Jednotlivé obrázky mají za úkol ověřit hlavně implementaci dekódování dat. Zastupují různé velikosti kódu a jsou kódovány za použití různých režimů. V některých případech se v jednom kódu použité režimy střídají. Všechny obrázky jsou na přiloženém CD v sekci čitelných. Vzhledem k faktu, že knihovna není schopna číst z reálných fotografií, jsou všechny obrázky přesné, synteticky vytvořené. U generovaných přesných obrázků ani nebyl očekáván nečitelný kód.

Kapitola 6

Závěr

Tato práce se věnuje dekódování čárového kódu v obraze. Představuje problematiku čárových kódů a jejich nejčastější oblasti využití. Popisuje dělení čárových kódů a hlavní vlastnosti vybraných typů kódů v kategorii 1D i 2D kódů.

Dále se práce věnuje návrhu a implementaci programové knihovny pro čtení vybraných typů 1D kódů. Z 2D kódů se bohužel podařilo implementovat pouze typ QR Code. Z implementovaných typů kódů je nejpokročilejší zpracování kódu typu EAN-13. Narozdíl od všech ostatních zvládá i čtení z vhodných reálných fotografií. V textu práce je navržena řada opatření, která je třeba učinit pro čtení reálných fotografií. Návrh i implementace čtení čárových kódů jsou v práci podrobně popsány.

Práce prezentuje netradiční přístup k prahování obrazu pomocí algoritmu k-means. Je prezentován návrh, podrobnosti implementace i výsledky takového přístupu k prahování.

Jako praktická část práce vznikla programová knihovna pro čtení čárových kódů ze statických obrázků. Knihovna umožňuje snadné použití a je podrobně dokumentována. Vedle knihovny vznikla i velmi jednoduchá demonstrační aplikace. Úkolem aplikace je umožnit testování a demonstrovat jednoduchost použití knihovny.

Možností pro další rozšíření práce zůstává mnoho. V oblasti 1D kódů by mohlo rozšíření spočívat v přidání dalších typů kódů nebo v rozšíření možností čtení reálných fotografií. Další možností je dodat detekci pozice a natočení kódu v obraze. V oblasti 2D kódů lze práci rozšířit také o lepší detekci polohy a otočení QR kódu v obraze. V implementaci také stále schází dekódování kontrolních součtů dat.

Literatura

[1] ISO/IEC 18004. Information technology – Automatic identification and data capture techniques – Bar code symbology – QR Code. Switzerland: ISO/IEC, 2000.

[2] Barcode coder: Standard 2 of 5 barcode [online]. `http://barcode-coder.com/en/standard-2-of-5-specification-103.html`, [cit. 2011-05-10].

[3] BarCodeIsland.com: Standard 2 of 5 symbology [online]. `http://www.barcodeisland.com/2of5.phtml`, 2006 [cit. 2011-05-10].

[4] BarCodeIsland.com: Code 128 symbology [online]. `http://www.barcodeisland.com/code128.phtml`, 2006 [cit. 2011-05-11].

[5] BarCodeIsland.com: EAN-13 [online]. `http://www.barcodeisland.com/ean13.phtml`, 2006 [cit. 2011-05-12].

[6] GAMMA, E.: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995, ISBN 0-201-63361-2.

[7] GS1 Czech Republic: Systém GS1 Základní informace [online]. `http://www.gs1cz.org/download/gs1-zakladni-informace.pdf`, 2006 [cit. 2011-05-13].

[8] Wikipedia: Data Matrix [online]. `http://en.wikipedia.org/wiki/Datamatrix`, 2010-05-10 [cit. 2011-05-14].

[9] Wikipedia: Reed-Solomon error correction [online]. `http://en.wikipedia.org/wiki/Reed-Solomon`, 2010-05-11 [cit. 2011-05-14].

[10] Wikipedia: PDF417 [online]. `http://en.wikipedia.org/wiki/PDF417`, 2010-12-18 [cit. 2011-05-13].

[11] Wikipedia: Code 128 [online]. `http://en.wikipedia.org/wiki/Code_128`, 2011-01-19 [cit. 2011-01-24].

[12] Wikipedia: k-means clustering [online]. `http://en.wikipedia.org/wiki/K-means_clustering`, 2011-04-25 [cit. 2011-05-10].

[13] Wikipedia: International Article Number (EAN) [online]. `http://en.wikipedia.org/wiki/International_Article_Number_(EAN)`, 2011-04-26 [cit. 2011-05-12].

Dodatek A

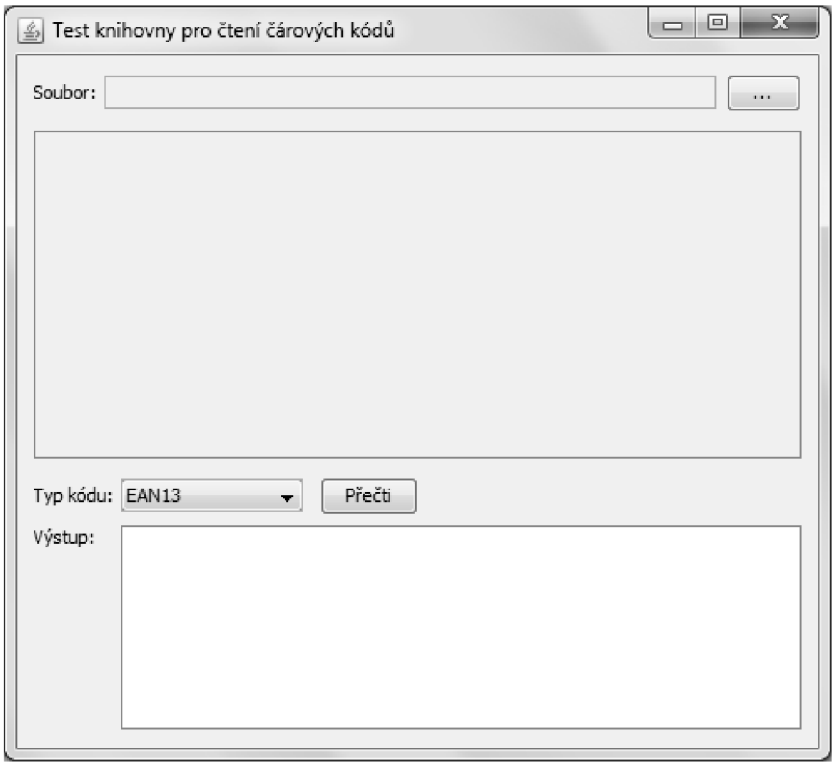
Obsah CD

- Zdrojové kódy knihovny (\BarCodeLib\src\).
- Přeložená forma knihovny (\BarCodeLib\bin\).
- Programová dokumentace knihovny (\BarCodeLib\doc\).
- Zdrojové kódy demonstrační aplikace (\BCLTest\src\).
- Demonstrační aplikace ve spustitelné formě (\BCLTest\bin\).
- Testovací sada čárových kódů (\Testy\).
- Zdrojové kódy technické zprávy (\Zprava\src\).
- Technická zpráva ve formátu PDF (\Zprava\Zprava.pdf).

Dodatek B

Uživatelský manuál

Okno aplikace je velmi jednoduché, viz obrázek B.1.



Obrázek B.1: Okno aplikace.

Nejprve je třeba stisknout tlačíto `...` v pravém horním rohu. Otevře se dialog výběru souborů. V dialogu zvolíme vstupní soubor, ten se po potvrzení zobrazí ve střední části. Nyní je třeba vybrat ze seznamu typ čárového kódu. Stiskem tlačítka `Přečti` je spuštěno čtení obsahu kódu. Výsledek čtení se zobrazí ve spodní části popsané `Výstup`.

Po přečtení kódu je možné vybrat další vstupní obrázek opět pomocí tlačítka vpravo nahoře.