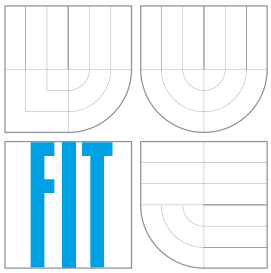
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ČTEČKA BRAILLOVA PÍSMA

BRAILLE READER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN MEZÍRKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ MARŠÍK
SUPERVISOR

BRNO 2012

Abstrakt

Tato bakalářská práce se zabývá problematikou převodu naskenovaného Braillova dokumentu na text. Nejprve se zaměřuje na samotné Braillovo písmo a princip skenerů. Dále představuje návrh možného způsobu rozpoznávání Braillových dokumentů. Nakonec je umístěn popis implementace programu s přehledem použitých knihoven OpenCV a Qt frameworku. Závěr poskytuje přehled spolehlivosti použitého řešení a diskutuje další možná rozšíření programu.

Abstract

This bachelor's thesis concern with problem transforms Braille's documents into text form. At first focus on Braille alphabet and principle of the scanners. Next is represent design of potential method of recognition Braille's documents. At the end is situated description of program's implementation with used library OpenCV and Qt framework. Conclusion discusses reliability and some program's extensions.

Klíčová slova

Braillovo písmo, rozpoznávání obrazu, zpracování obrazu, skenery, OpenCV, Qt framework.

Keywords

Braille system, image recognition, image processing, scanners, OpenCV, Qt framework.

Citace

Martin Mezírka: Čtečka Braillova písma, bakalářská práce, Brno, FIT VUT v Brně, 2012

Čtečka Braillova písma

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Lukáše Maršíka. Testovací dokumenty mi poskytlo středisko Teiresiás Masarykovy univerzity v Brně a také Bc. Jiří Fabík. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Mezírka
15. května 2012

Poděkování

Rád bych poděkoval především Ing. Lukáši Maršíkovi za příkladné vedení této bakalářské práce.

© Martin Mezírka, 2012.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **2**

2 Braillovo písmo **3**
2.1 Braillovo písmo 3
2.2 Normy Braillova písma 4

3 Skenery **6**
3.1 Obrazové pojmy 7

4 Návrh rozpoznávání **8**
4.1 Redukce barevného prostoru 8
4.2 Natočení obrazu 10
4.3 Lokalizace a detekce bodu 11
4.4 Detekce vzdáleností mezi body 12
4.5 Detekce znaků 13
4.6 Oboustranně tištěné dokumenty 15
4.7 Další minimální nutné změny 16

5 Implementace **18**
5.1 OpenCV 18
5.2 Qt framework 19
5.3 Mapování znaků 21
5.4 Programová implementace 23

6 Dosažené výsledky **26**

7 Závěr **27**

A Obsah CD **29**

B Česká Braillova abeceda **30**

Kapitola 1

Úvod

Tato bakalářská práce se zabývá problematikou převodu naskenovaného Braillova dokumentu na text. Nechtěl jsem implementovat a popisovat známé postupy, ale vyzkoušet něco jiného. Především jsem se snažil rozvinout původní jednoduchý nápad na detekci teček a převodu jednostranně tištěných Braillových dokumentů. Implementoval jsem tento postup a chtěl zjistit, zda-li je vůbec možné dosáhnout nějakých pozitivních výsledků. Dále jsem se pokoušel tento postup mírně modifikovat a přizpůsobit i oboustranně tisknutým (vytláčeným) dokumentů.

Na začátku práce uvedu několik zajímavých skutečností Braillova písma, které musíme zohlednit při převodu Braillova dokumentu na text.

Ve třetí kapitole popisuju skenery a jejich vlastnosti při skenování Braillových dokumentů. Dále zde uvádím některé základní pojmy, se kterými se můžete setkat ve zpracování obrazu, včetně softwaru určeného k rozpoznávání Braillových či textových znaků ze skenovaných předloh.

Celá čtvrtá kapitola pojednává o mém návrhu tvorby čtečky Braillových dokumentů. V první části se zaměřuji na rozpoznání a převod jednostranně tištěných dokumentů. Ve druhé části poskytnu náhled do problematiky a návrh překladu oboustranně vytlačených dokumentů.

Popis implementace programu je obsažen v páté kapitole současně s přehledem použitých knihoven pro práci s obrazem OpenCV a grafickým uživatelským rozhraním QT framework.

Poslední kapitola závěr shrnuje celou práci a poskytuje přehled dosažených výsledků.

Kapitola 2

Braillovo písmo

Snad každý tuší co znamená Braillovo písmo a k čemu se používá. Ale spousta z vás jistě nezná veškerá jeho pravidla použití a jiné detaily. Na začátku práce tedy shrnu různé poznatky o Braillově písmu.

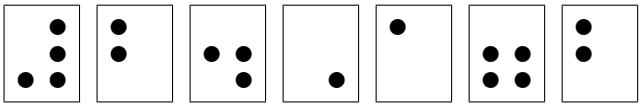
2.1 Braillovo písmo

Jedná se o prostředek, který nevidomým (zrakově postiženým) umožňuje rozpoznávat text prostřednictvím hmatu. Písmo vytvořil přibližně v roce 1824 francouz Louis Braille ve svých patnácti letech úpravou vojenského systému, který umožňoval čtení za tmy. Původní vojenský systém byl založen na dvanácti bodech. Louis Braille tento systém zjednodušil a vylepšil s použitím pouze šesti bodů. Písmo se stalo celosvětově rozšířeným až dávno po jeho smrti ve třicátých letech 19. století [8]. Text zapsaný v tomto systému je sestaven z posloupnosti jednotlivých znaků Braillovy abecedy.

Braillův znak

Každý znak je tvořen šesticí bodů (2 x 3), přičemž některé z těchto bodů jsou vystouplé. Různé kombinace výstupků určují právě jeden význam znaku. Dokumenty zapsané tímto způsobem mohou býti oboustranné. Celkově je k dispozici pouze $2^6 = 64$ různých možností popisů jednoho znaku. Mezera je reprezentována prázdným polem. Vzhledem k malému počtu kombinací Braillova abeceda nepopisuje zvlášť velká písmena. Pokud chceme napsat velké písmeno, nebo řetězec velkých písmen, musí před textem předcházet speciální značka (prefix), která mění význam následujících symbolů. Obdobný způsob je uplatněn také při zadávání řeckých písmen a čísel. Při použití číselného prefixu písmena z počátku abecedy zastupují jednotlivé číslice a některé další znaky mohou mít navíc speciální význam. Jedná se například o písmena: *m*, *p*, *r*, které představují v číselném významu znaky: %, ‰, § [7].

Existuje i osmibodové písmo (2 x 4). Při jeho používání se sice vyhneme prefixům, ale tato varianta se příliš nepoužívá, jelikož písmo je již obtížně čitelné. Přehled základních symbolů české abecedy, prefixů a diakritických znamének je uveden v příloze B.



Obrázek 2.1: Příklad zápisu: $2 + A = b$

Na obrázku: 2.1 vidíte ukázku, jak bychom mohli zapsat rovnici $2 + A = b$. Znak označující písmeno „b“ se v příkladu vyskytuje dvakrát. Jednou ve významu písmene a podruhé jako číslice 2, jíž musí předcházet prefix měnící význam znaku „b“ na číslo. Platnost číselného prefixu se vztahuje na následující posloupnosti znaků „A“ až „J“, na desetinnou čárku a desetinnou tečku, která odděluje číslice po tisících. Platnost číselného prefixu končí prvním jiným výskytem znaku, než které byly vyjmenovány (do významu číselného prefixu mohou patřit i další znaky například dříve uvedené m, p, r). Další použitý prefix v příkladu mění pouze jedno následující malé písmeno na velké, v našem případě „a“ na „A“.

Platnost prefixu zápisu řetězce velkých písmen je ukončena jednou z možností:

- mezerou
- jiným prefixem
- interpunkčním znaménkem

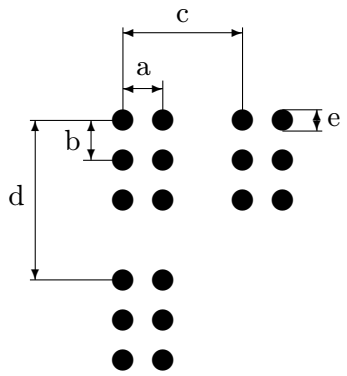
Problémy Braillova písma

Přestože se jedná o mocný dorozumívací nástroj nevidomých, existuje pro něj spousta omezení a problémů.

- Znaky Braillova písma jsou obecně mnohem vetší než tištěné písmo. Knihy zapsány v Braillově písmu bývají tedy objemnější.
- Vzhledem k tomu, že na jeden řádek se vleze omezený počet znaků, nastává problém jak sázet tabulky a další formátování. Všechny informace musí být totiž sázeny sekvenčně, což opět zvětšuje samotnou délku textu. jednoduché matematické a fyzikální zápisy mohou zabírat i několik řádků.
- Různé země používají různé normy Braillova písma. Ať se jedná o rozměry a vzdálenosti jednotlivých bodů, nebo o rozdíly v samotné abecedě znaků daného národa. Mezi českou a slovenskou normou je mezi základními písmeny Braillovy abecedy sedm rozdílů [11].

2.2 Normy Braillova písma

Rozměry buněk, stejně tak jako velikosti jednotlivých bodů a jejich vzájemné horizontální/vertikální rozestupy, jsou přesně dány zemí původu Braillova dokumentu. Následující obrázek 2.2 a tabulka 2.1 udává základní přehled těchto rozměrů v různých částech světa [9].



Obrázek 2.2: Rozměry a vzdálenosti bodů

	a $[mm]$	b $[mm]$	c $[mm]$	d $[mm]$	e $[mm]$
Evropa	$2,5$	$2,5$	$6,0$	$10,0$	$1,3$
Spojené státy americké	$2,3-2,5$	$2,3-2,5$	$6,1-7,6$	$10,0-10,1$	$1,5-1,6$
Austrálie	$2,29-2,50$	$2,29-2,54$	$6,0-6,1$	$10,16-10,41$	$1,4-1,5$
Japonsko	$2,13$	$2,37$	$5,4$	$13,91$	$1,43$
Korea	$2,0$	$2,0$	$5,0$	$6,0$	$1,5$

Tabulka 2.1: Rozměry a vzdálenosti bodů

Kapitola 3

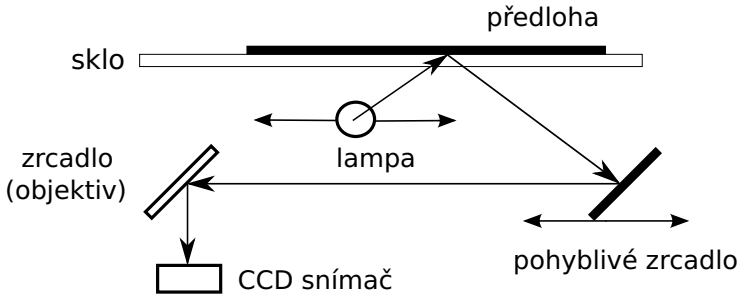
Skenery

Skener je vstupní zařízení, které umožňuje převod z dvojrozměrné, případně trojrozměrné předlohy do digitální podoby [6]. Existuje několik skupin skenerů dle typu konstrukce:

- Stolní skenery (ploché) – v této práci budeme uvažovat použití právě stolních skenerů.
- Ruční skenery – obraz se získává ručním posunem skeneru po snímané předloze. Rozlišení a kvalita obrazu není příliš vysoká.
- Bubnové skenery – předloha je umístěna na rotující válec, kde je dále snímána paprskem. Bubnové skenery se vyznačují vysokou kvalitou snímaných a rozsáhlých ploch.
- Filmové skenery – velmi specifické použití pro snímání políček ve filmovém pásu.
- 3D skenery – slouží ke snímání trojrozměrných objektů.

Princip stolního skeneru

Předloha se umiťuje na sklo, pod kterým se pohybuje snímací rameno. Rameno provádí nasvícení a snímání dokumentu. Dle vlastností povrchu se odráží pouze některé vlnové délky. Tmavá plocha světlo pohlcuje, světlé plochy naopak odrážejí více světla [2]. Nejčastěji se odražené světlo soustavou zrcadel dopraví k CCD snímači, který intenzitu světla převede na elektrický proud, například jak je to uvedeno na následujícím obrázku 3.1 [10].



Obrázek 3.1: Princip činnosti skeneru

Díky nasvícení předlohy ramenem z jednoho směru vystouplé body způsobí světlou oblast na protilehlé polorovině tečky než protlačené body. Tento jev můžeme využít k rozpoznání strany dokumentu, na které se tečka nachází.

3.1 Obrazové pojmy

Následuje výčet základních pojmů, se kterými se můžete setkat ve zpracování obrazu.

- Rozlišení – nejčastěji v souvislosti se skenováním uvádí v jednotkách DPI (dots per inch). Číslo udává počet bodů (pixelů) obrazu na vzdálenost jednoho palce ($2,53\,cm$). Pokud známe rozlišení obrazu, vzdálenost určená v pixelech lze přepočítat na vzdálenost v mm pomocí vzorce 4.3.
- Barevná hloubka – počet barevných odstínů obrazu. Typicky 24bitová barevná hloubka (tři barevné kanály RGB po osmi bitech) odpovídá $2^{24} = 16\,777\,216$ odstínů barev.
- Stupně šedi – (grayscale) obrázek určený pouze různými odstíny šedé barvy. Typicky 8bitové barevné hloubky, k dispozici je v tomto případě celkem 256 odstínů šedi.
- Černobílý obraz – informace o barvě pixelu je přítomna pouze ve dvou hodnotách černá a bíla.
- Komprese – způsob uložení obrazových dat se zmenšeným datovým tokem, pokud možno za cenu s co nejmenší ztrátou informace. Příklady různých obrazových formátů, které provádějí kompresi dat: *JPEG*, *TIFF*, *PNG*, formát bez komprese dat: *BMP*.

OCR software

OCR (Optical Character Recognition) slouží k rozpoznávání a digitalizaci textu z tištěných dokumentů. Vstupní předloha se získává prostým skenováním dokumentů. Úspěšné převedení textu záleží především na kvalitě původního tisku a částečne také na kvalitě skenování. Rozpoznávání může selhávat v případech, kdy jednotlivá písmena mají vůči sobě mírný překryv, dále při slabě viditelném tisku, nebo pokud byl text vytisknut méně kvalitními jehličkovými tiskárnami.

OBR sotware

OBR (Optical Braille Recognition) je komerční software firmy Neovision umožňující převod naskenovaných Braillových dokumentů (jednostranných i oboustranných) typicky ve formátu A3/A4 na text. OBR vyniká především velmi nízkou chybovostí překladu nepoškozených dokumentů. Ale i v případě nekvalitních a poškozených dokumentů je úspěšnost stále na vysoké úrovni. Udávaná chybovost softwaru činí v průměru $99,98\%$ (1 až 2 tečky na stránku). Za zmínku stojí podpora šesti i osmi bodového Braillova písma. OBR je dle specifikace dostupný pouze pro operační systém Windows a to do verze XP [5].

Kapitola 4

Návrh rozpoznávání

Tato kapitola popisuje postupy, které jsou uplatněny při tvorbě programu čtečka Braillova písma.

4.1 Redukce barevného prostoru

Dokument necháme načíst rovnou v odstínech šedi. Vztah barevného obrazu (barevný model RGB) a intenzity šedotónového obrazu popisuje empirický vztah 4.1. Odlišné váhy barevných složek jsou způsobeny různou citlivostí oka na tyto barvy [2].

$$I = 0,299R + 0,587G + 0,114B \tag{4.1}$$

Naskenovaný obraz může obsahovat na krajích rušivé elementy, vzniklé špatným skenováním okrajů. Tyto jevy by mohly nepříjemně ovlivnit samotný proces rozpoznávání. Proto bude vhodné na úplném začátku procesu rozpoznávání okraje dokumentu mírně ořezat.

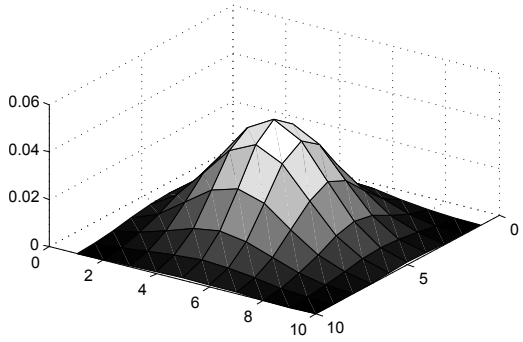
Gaussovo vyhlazení obrazu

Následně bychom mohli provést a také provedeme Gaussovo vyhlazení obrazu. Při výsledném rozpoznávání na kvalitních dokumentech nebude mít ono vyhlazení velký vliv. Dokumenty budou rozpoznány správně i bez vyhlazení. Gaussovo vyhlazení ale celkově redukuje šum v obraze a zároveň způsobí, že Braillovy tečky po prahování více vyniknou – nebudou tolik „roztřepené“ (obrázky 4.2c a 4.2d). Pokud bychom měli dokumenty v horším stavu, potom by užitek Gaussova vyhlazení byl znatelný. Proto využijeme této možnosti.

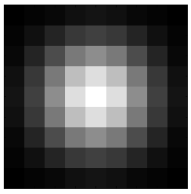
Gaussovo vyhlazení odstraňuje šum v datech za cenu zozmazání obrazu. Výsledný obraz lze získat konvolucí s maskou (maticí). Prvky matice odpovídají hodnotám (ploch) dvourozměrné Gaussovy funkce. Aby výsledný obraz měl stejný jas jako původní, musí se suma všech elementů matice rovnat jedné. Na obrázku 4.1b vidíte znázorněnou konvoluční masku a funkci (obrázek 4.1a) z níž se vypočítaly hodnoty masky (matice). Světlejší barva v obrázku odpovídá větším hodnotám prvků matice. Parametry filtru byly nastaveny na okolí 9×9 hodnot, při velikosti směrodatné odchylky $1,85$.

Prahování obrazu

Po Gaussovém vyhlazení přichází na řadu redukce šedotónového obrazu na černobílý. Černobílý obraz obsahuje barevnou informaci o každém pixelu pouze ve dvou hodnotách (černá a bílá). Klasický převod (thresholding) provádí porovnání intenzity každého pixelu s jednou



(a) Dvourozměrná Gaussova funkce



(b) Konvoluční matice (maska) 9x9

Obrázek 4.1: Gaussovo vyhlazení

globální prahovou hodnotou. Na základě tohoto porovnání rozhodne o výsledné barvě bodu. Jediným, ale za to podstatným problémem prahování je vhodné zvolení porovnávací hodnoty, tak aby došlo ke správnému separování informace. V OpenCV je k dispozici adaptivní prahování, zde se nerozhoduje podle globální prahové hodnoty, ale práh se určuje dynamicky na základě blízkého okolí bodu. Adaptivní prahování poskytuje uspokojivé výsledky.

Na obrázcích 4.2 vidíte dva Braillovy znaky, s použitím různých filtrů a následného adaptivního prahování. Obrázek 4.2a nám ukazuje jak může vypadat naskenovaná část Braillova dokumentu v odstínech šedi. V podstatě s těmito daty pracujeme. Obrázek 4.2b ukazuje pouze aplikaci adaptivního prahování. Zbylé dva obrázky 4.2c a 4.2d představují ukázku Gaussova vyhlazení šedotónového obrazu přes oblast 3×3 centrovanou na každý pixel, respektive okolí 9×9 s následným adaptivním prahováním.

(a) Odstín šedi

(b) Adaptivní prahování

(c) Gaussovo vyhlazení 3x3

(d) Gaussovo vyhlazení 9x9

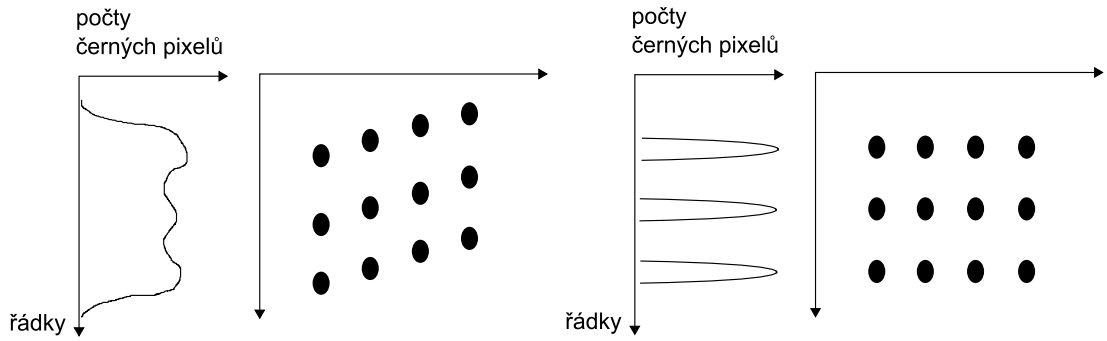
Obrázek 4.2: Braillovy znaky s adaptivním prahováním a Gaussovým vyhlazením

Pokud budeme chtít navíc rozpoznávat oboustranně vytlačené dokumenty, je nutné si uchovat informaci o barvě bodů ve třech hodnotách (sekce 4.6). Pomocí tohoto rozšíření budeme následně schopni určit orientaci vystouplých teček.

4.2 Natočení obrazu

Pravděpodobně naskenovaný dokument nebude přesně zarovnán – řádky jednotlivých bodů Braillova písma nebudou rovnoběžné s okraji obrazu. Tento fakt je nutné řešit, aby samotné rozpoznání znaků bylo co nejjednodušší. Diskrétní Fourierova transformace by mohla být využita k výpočtu úhlu rotace. Aby tato metoda pracovala zcela správně, bez odchylek způsobených špičkami v diskrétní Fourierově transformaci, které vznikají když se výsledný obraz blíží k původnímu, potřebovala by další zdokonalení [3]. Proto využiji jiný přístup.

Na obrázku 4.3 vidíte řádkový histogram udávající počty černých bodů na jednotlivých řádcích. Spočítáním průměru lokálních maxim hodnot černých pixelů na řádcích nám napoví o správném sklonu obrazu. Chceme maximalizovat tento průměr hodnot, v ten okamžik totiž budou jednotlivé tečky Braillova písma horizontálně i vertikálně zarovnány, jak si můžete všimnout v pravé části obrázku.



Obrázek 4.3: Řádkový histogram

Jednoduchým postupným natáčením obrazu a zkoumáním změn celkové průměrné sumy černých bodů, můžeme jít po zlepšujících se výsledcích. Následným zmenšováním kroků se ustálíme ve stavu, kdy náš dokument bude dokonale zarovnán. Tento ideální výsledek bude samozřejmě možný pouze za předpokladu, že původní náklon dokumentu nebyl příliš veliký. Algoritmus 1 nám ukazuje jeden krok natočení dokumentu. Dle počátečního nastavení *kroku* a počtu opakování úseku tohoto algoritmu docílíme výsledné přesnosti natočení. Po n iteracích bude jeden *krok*, o který se posuneme k lepšímu řešení roven $\frac{počáteční\ nastavení\ kroku}{2^{n-1}}$. Samotná rotace obrázku je časově náročná operace, alespoň částečného zrychlení na víceprocesorových systémech můžeme dosáhnout souběžným otáčením obrázku na levou i pravou stranu.

Algoritmus 1: Jeden krok otočení dokumentu

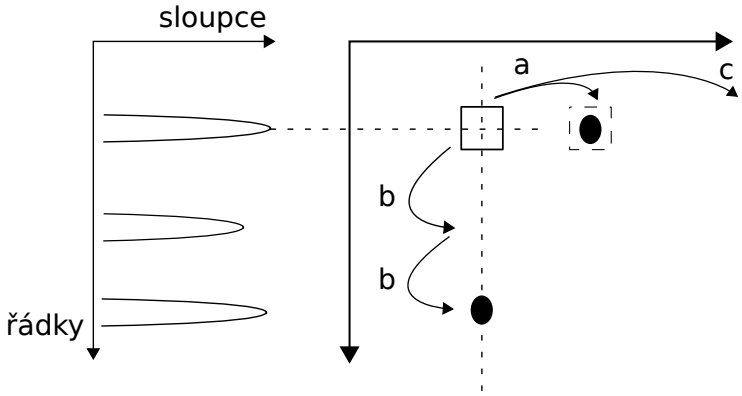
```
PrůměrL = Natočení obrázku doleva o krok
PrůměrP = Natočení obrázku doprava o krok
if PrůměrL > PrůměrStřed then
    aktuální úhel natočení += krok
    PrůměrStřed = PrůměrL
end
if PrůměrP > PrůměrStřed then
    aktuální úhel natočení -= krok
    PrůměrStřed = PrůměrP
end
krok /= 2
```

Proměnné označené jako *Průměr*X obsahují průměrnou hodnotu maxim v obraze (počet černých pixelů na jedno maximum). Čím větší je tato hodnota tím více leží jednotlivé body v přímce za sebou. Rozhodl jsem se, že v úvahu budu brát pouze maxima podél vertikální osy, protože řádků je více a jsou kratší. Hodnoty maxim budou přesnější – „ostřejší“. Při nepřesném tisku totiž na krátkém úseku řádku nedojde k tak výraznému odchýlení, jako na dlouhém sloupci. Navíc dokument může obsahovat velmi málo řádků, jedná-li se o konec kapitoly apod. Tato situace by představovala v rámci sloupců ťežce detekovatelná maxima. V řádkovém histogramu tyto problémy nehrozí, nebo alespoň nejsou tak výrazné.

Předešlé výpočty maxim s drobnou úpravou využijeme navíc i v další části práce. Nebude nás zajímat počet (průměr) pixelů, ale místo toho se budeme snažit zjistit pozice maxim (vertikálních i horizontálních).

4.3 Lokalizace a detekce bodu

Po správném natočení obrazu, kdy jednotlivé tečky jsou horizontálně i vertikálně zarovnány, můžeme přistoupit k další fázi. Jelikož nevíme nic o možných pozicích znaků a jejich teček, použiji přístup, který mi napoví očekávanou pozici první tečky v Braillově dokumentu (první tečka vlevo nahoře). Problém v samotném rozpoznání Braillova dokumentu je v tom, že nevíme zda na prvním místě - řádku, sloupci se má nacházet vystouplý bod či prázdné místo. Navíc nutně potřebujeme k rozpoznání textu více informací než pouze lokální okolí jednoho bodu. Pokud bychom znali pozici například i dvou, nebo více bodů, stále nevíme nic o jejich vzájemných souvislostech. Patři-ly by všechny tyto body k jednomu znaku, nebo část bodů by představovala již následující znak?



Obrázek 4.4: Lokalizace prvního a následujících bodů

Pomoc v určení pozice první tečky vlevo nahoře nám poskytnou dříve uvedené histogramy a jejich maxima. Nápad na lokalizaci prvních bodů Braillova písma vychází z pozic prvních maxim (špiček) řádkového a sloupcového histogramu. Vezmeme-li v úvahu tyto pozice, na jejich průsečíku se nachází potenciální bod-tečka obrázek 4.4. Zkontrolováním okolí této pozice zjistíme, zda se jedná o vystouplý bod v dokumentu či o prázdné místo. Toto rozhodnutí provedeme jednoduše součtem všech pixelů v okolí středu bodu a následným porovnáním s hraniční hodnotou. Po provedení rozhodnutí se přechází na další potenciální pozici tečky.

Aby rozhodovací okolí pokrylo celou předpokládanou plochu tečky, mělo by být co největší, ale zároveň nemůže zasahovat do oblastí cizích teček, musí se tedy zvolit optimální

rozměry. Velikost rozhodovacího okolí je obdélník o stranách rovných nejmenším vzdálenostem mezi tečkami ve vodorovných a svislých směrech. Střed obdélníku je umístěn na předpokládané místo tečky.

4.4 Detekce vzdáleností mezi body

Vzdálenosti mezi body jsou sami od sebe proměnlivé, lišící se o jednotky pixelů. Navíc dosahují různých hodnot mezi body v jednom znaku a mezi body náležícím různým znakům, jak vidíte na obrázku 2.2 a v tabulce 2.1.

Potřebujeme určit rozměry označené v obrázku 2.2 jako a, b, $c-a$, $d-2b$. Tyto hodnoty budeme odhadovat z pozic maxim, které jsou určeny celočíselnou hodnotou. Bohužel tyto hodnoty nejsou příliš přesné a navíc některé z nich mohou chybět. Například pokud se v jedné řadě (sloupci) vyskytlo velmi málo bodů, potom se nemuselo vůbec podařit detekovat dané lokální maximum. Ale pokud se zaměříme na četnosti velikostí rozdílů dvou pozic sousedních maxim tabulka 4.2, zjistíme, že ve dvou nejčetnějších hodnotách (a jejich okolí) získáme velikost rozměrů a a $c-a$.

Může stát, že některé maxima jsou detekovány zcela chybně – získáním četností rozdílů tyto chyby eliminujeme, protože se zaměříme pouze na vyšší hodnoty četností.

index	1	2	3	4	5	6	7	8	9	10
Rozdíl dvou maxim $[px]$	19	20	21	37	38	39	40	41	56	61
četnost	21	26	1	1	1	4	15	1	1	1

Tabulka 4.1: Rozdíly sousedních vertikálních maxim a jejich četnosti

index	1	2	3	4	5	6	7	8	9	10	11
Rozdíl dvou maxim $[px]$	16	17	18	19	26	27	28	29	30	45	46
četnost	1	7	8	11	1	5	9	6	6	1	1

Tabulka 4.2: Rozdíly sousedních horizontálních maxim a jejich četnosti

Vzdálenosti rozdílů maxim jsou různé po celém dokumentu, jak můžete vidět v tabulkách 4.1, 4.2. To je dáno nepřesností při tisku a také detekce maxim z histogramů dává vždy mírné odchylky. Abychom získali průměrnou hodnotu vzdáleností v dokumentu použiji vážený průměr. Z okolí od hodnot s největší četností beru v úvahu hodnoty v maximálních vzdálenostech $2px$ od středu. Provedu vážený průměr až pěti těchto vzdáleností. Pro každý výpočet (pro každou tabulku) se použije vzorec pro vážený průměr 4.2 dvakrát. Větší ze dvou vzdáleností musí odpovídat v horizontálním směru hodnotě $a-c$, menší hodnotě a. Stejně tak ve svislém směru tabulka 4.1 větší hodnota odpovídá vzdálenosti $d-2b$ a menší je b.

$$\overline{vzdálenost} = \frac{\sum_{i=0}^{n} četnost_i * rozdíl_i}{\sum_{i=0}^{n} četnost_i} \tag{4.2}$$

V předchozím příkladu z tabulky 4.1 by se například vzdálenost b spočítala z prvních třech sloupců, vzdálenost $d-2b$ zase ze sloupců 5, 6, 7 a 8. Převedením jednotek z *pixelů*

na mm pomocí vzorce 4.3 můžeme porovnat hodnoty výstupu programu s předpokládanou roztečí hodnot při tisku Braillova dokumentu (tabulka 2.1).

$$délka\,[mm] = \frac{počet\ pixelů}{DPI} * 25,4\,[mm] \tag{4.3}$$

	$a\,[mm]$	$b\,[mm]$	$c-a\,[mm]$	$d-2b\,[mm]$
Evropa	2,5	2,5	3,5	5,0
dokument 1	2,3055	2,4871	3,6077	5,0498
dokument 2	2,3300	2,4967	3,5658	5,0165
dokument 3	2,3397	2,5074	3,5609	5,0123
průměr	2,3251	2,4971	3,5781	5,0262

Tabulka 4.3: Srovnání vzdáleností mezi tečkami

Tyto testovací dokumenty byly skenovány v rozlišení pouze $200 \times 200\,DPI$. Přesto vidíme, že vypočtené hodnoty skutečně odpovídají předpokládaným. Přičemž vzdálenosti ve vertikálním směru vykazují menší odchylky než v horizontálním. Toto může býti dáno menším počtem dat (méně sloupců než řádků). Vypočítané hodnoty potvrzují předpoklad o přesnějším určování maxim v řádcích zmíněné v sekci o zarovnání obrazu 4.2.

Velikosti teček nejsou zahrnuty v porovnání, neboť jejich velikost je výrazně ovlivňována v závislosti na nastaveném prahování.

4.5 Detekce znaků

Pokud bychom zkoumali pozice předpokládaných bodů pouze na základě vertikálních a řádkových histogramů – zkoumání předpokládaných bodů pouze v průsečících maxim. Setkali bychom se s již zmíněnými problémy při chybějících, či špatně určených řádkových (sloupcových) maxim. Tento problém by zejména nastával, při ukončování odstavců, kdy se na jednom řádku může vyskytovat pouze několik Braillových znaků. Tyto znaky obsahují velmi málo informací – teček, né-li žádné, tudíž by z nich nešlo vyvodit maxima v histogramech. Problém by byl dále znatelný například při uvedení čísla strany dokumentu na spodním okraji, kde se typicky vyskytuje pouze prefix pro číslo a znaky představující číslo samo.

Detekce dalších bodů

Abychom se vyhnuli nepříjemnostem s chybějícími maximy využijeme přístup, který bude detekovat další tečky relativně vzhledem k aktuální poloze. Použijeme dříve vypočítané vzdálenosti mezi jednotlivými body Braillova písma (sekce 4.4). Na jejich základě se budeme pohybovat po dokumentu (obrázek 4.4). Díky relativně přesnému výpočtu vzdáleností, které jsou určeny desetinným číslem se můžeme pohybovat po předpokládaných pozicích bodech s dostatečnou přesností. Směr pohybu si mužeme zvolit. Například nejprve prozkoumáme první tři svislé tečky a poté druhý sloupec opět od horní ke spodní tečce – stejnou posloupností se obvykle textově zapisují kombinace Braillových znaků (obrázek 5.2). Následuje posun o vzdálenost $c-a$ na další znak apod.

Jednoduchý návrh implementace nalezení odpovídajícího znaku, při použití tohoto postupu by mohl vypadat následovně. Všechny znaky v Braillově abecedě jednoznačně očíslujeme 0 až 63. Čísla budou odpovídat indexům do pole patřičných znaků. Binární obraz

tohoto čísla odpovídá topologii teček ve znaku. Hodnota 1 značí přítomnost bodu, 0 nikoliv. Nejvýznamnější bit nechť je první zkoumaná pozice znaku. Algoritmus 2 ukazuje jak by mohlo vypadat získání indexu do pole znaků.

Algoritmus 2: Získání indexu znaku

```
index = 0 ;
while Posun na další zkoumanou pozici tečky do
    index <<= 1 ;                                  // posun o jeden bit doleva
    if Jedná se o tečku then
        index |= 1 ;                    // nastavení nejméně významného bitu
    end
end
```

Dodatečné centrování

Při rozpoznání tečky se navíc vždy může použít dodatečné centrování aktuálního středu předpokládané pozice tečky. Jedná se o obyčejný aritmetický průměr 4.5. Proměnná $sumX$ obsahuje sumu x-ových souřadnic pixelů představující bod a *počet* vyjadřuje počet těchto pixelů. Centrování by ale mělo být použito v případě, kdy jsme si jistí, že se skutečně jedná o Braillův bod a ne o šum v obraze. Jinak by hrozilo vychýlení ze správné pozice. V našem případě by nás mohla ujistit zvýšená hodnota prahu, na základě které se rozhoduje o existenci tečky v rozhodovacím obdélníku. Použití centrování nás obvykle vždy udrží ve správných mezích předpokládaných pozic bodů (i při méně přesném určení vzdáleností mezi tečkami).

$$\begin{aligned} středX &= \frac{sumX}{počet} \\ středY &= \frac{sumY}{počet} \end{aligned} \tag{4.4}$$

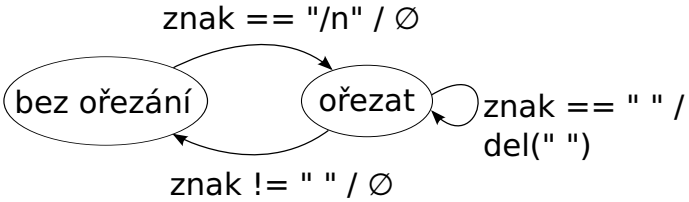
Centrování ale nemůže být využito vždy. Například při číslování stránek, kde se Braillovy znaky objevují osamoceně uprostřed stránky. Proto se i při používání centrování snažím určit co nejpřesnější průměrné vzdálenosti mezi body v dokumentu, abych se po několika krocích příliš neodchýlil z předpokládaných pozic teček.

Existuje jedno úzké místo. A to určení pozice úplně prvního znaku dokumentu, jeho první tečky vlevo nahoře. Jsme odkázání pouze na správnou detekci prvních horizontálních a vertikálních maxim. Pokud se tato pozice určí špatně, je pravděpodobné, že celý dokument bude špatně rozpoznán.

Odstranění přebytečných znaků

Při posunování po dokumentu a zaznamenávání rozpoznaných znaků, musíme logicky projít celý dokument až úplně ke krajům (pravý a spodní okraj). Tím pádem budou vznikat na koncích řádků prázdné znaky – mezery. V závislosti na pozici ukončení řádku jich bude různé množství. Dokonce ani výsledný text dohromady s nadbytečnými mezerami nebude zarovnaný. Převedený text nevypadá pěkně. Po provedení rozpoznání dokumentu necháme ořezat přebytečné mezery. Jak může vypadat řešení vidíte na obrázku 4.5, za předpokladu, že průchod textem je v obráceném směru od posledního znaku k prvnímu.

Stejně tak při malém počtu vytlačených řádků dokumentu, na konci vznikají přebytečné prázdné řádky, které ale ponecháme na svých místech. Nadbytečné řádky na konci souboru výrazně nenarušují strukturu textu, působí spíše jako očekávaný oddělovač jednotlivých dokumentů. Pokud má dokument navíc uvedeno číslo strany ve spodní části, jakož tomu často bývá, odstranění řádků není možné uplatnit ve velké míře.



Obrázek 4.5: Ořezání přebytečných mezer

4.6 Oboustranně tištěné dokumenty

Postup převodu naskenovaných Braillových dokumentů na text popsaný v předešlé části, pracuje spolehlivě na jednostranně tištěných dokumentech. V dnešní době, se ale s ohledem na šetření místem a prostředky, převážně vyrábí dokumenty s oboustranným tiskem. Braillovy body se protláčejí na protilehlých stranách s mírným posunem. Přičemž orientace výstupků Braillových teček určuje příslušnou stranu dokumentu.

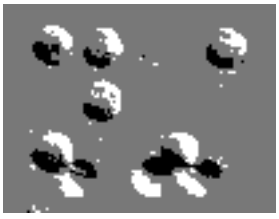
Předešlý postup nemůže rozpoznávat znaky správně. V první řadě musíme upravit prahování dokumentu (sekce 4.1), abychom mohli získat informaci o orientaci vystouplých bodů.

Redukce barevného prostoru

Prahováním pouze do binárního (černobíleho) obrazu ztrácíme důležitou informaci o náležitosti teček ke stranám. Na obrázku 4.6a vidíte jak může vypadat sken oboustranného dokumentu. Obrázek 4.6b byl vytvořen stejně jako v případě jednostranného dokumentu Gaussovým vyhlazením a adaptivním prahováním do dvou hodnot. Z tohoto černobílého obrazu nejde rozpoznat, které tečky leží na kterých stranách. Klíčem k rozpoznání správné strany je zaměřit se kromě tmavého středu bodu také na světleji osvícené místa. Obrázek 4.6a byl převeden do tříhodnotové barevné reprezentace (obrázky 4.6c a 4.6d). Nyní již dokážeme i rychlým pohledem poznat, které tečky leží na přední a které na zadní straně.

(a) Odstín šedi

(b) Gaussovo vyhlazení 9x9 s binárním adaptivním prahováním



(c) Tříhodnotové prahování

(d) Gaussovo vyhlazení 9x9 s tříhodnotovým prahováním

Obrázek 4.6: Výřez oboustranného Braillova dokumentu

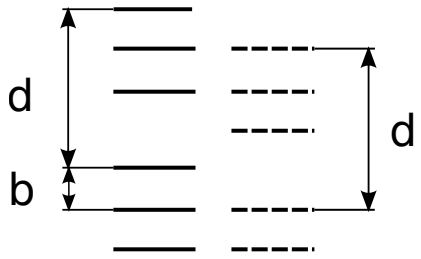
4.7 Další minimální nutné změny

Mnou navržený postup není nejvhodnější k rozpoznání textu v oboustranných Braillových dokumentech. Pokud ale provedeme několik dalších úprav můžeme získat alespoň částečně funkční čtečku oboustranných dokumentů.

Body z různých stran jsou vytlačeny s posunem v horizontálním směru, ve vertikálním směru posun není výrazně znatelný. Body na řádcích jsou relativně v zákrytu. Ke správnému zarovnání se může využít stejný algoritmus jako pro jednostrané dokumenty.

Narážíme ještě na další problém a tím je určení pozice prvního řádku první stránky textu. Jednotlivé tečky v horizontálním směru ze dvou protějších stránek jsou sice relativně za sebou, ale první linie teček z přední strany se nemusí shodovat s první linií teček ze zadní strany (obrázek 4.7). Musí se navíc provést prozkoumání několika prvních řádků/sloupců a zjistit první výskyt obou typů teček.

Obrázek 4.7 ukazuje možný posuv řádků na různých stranách dokumentů. Plná čára značí linie bodů na přední straně dokumentu, čárkovaná čára představuje linie bodů na protější straně dokumentu. Z vertikálních histohramů nejsme schopni korektně určit vzdálenost značenou v obrázcích 2.2 a 4.7 jako d. Velikost b může býti také nepříjemně ovlivněna.



Obrázek 4.7: Posun řádků oboustranných dokumentů

Detekce horizontálních maxim (sloupce) bude pracovat nadále celkem dobře. Ovšem tečky z různých stran někdy splývají ve větší a širší shluky černých bodů. Pozice maxim z nich detekovaných mohou býti méně přesné, respektive budou nejčastěji procházet středy těchto shluků.

Lokalizace a posun mezi body

Původní myšlenka zůstala stejná. Od počátečního místa se pohybujeme stejným způsobem. Nyní ale pozice horizontálních maxim nám udávají pozice středů mezi dvěma tečkami. Bude stačit pouze do inicializační fáze započítat vodorovný posun v případě přední strany dokumentu jedním směrem, v případě zadní strany směrem opačným.

Centrování ve vertikálním směru můžeme použít bez problémů. Problém nastává v horizontálním směru. Pokud bychom se snažili vycentrovat v oblasti podélného shluku, byli bychom místo centrování vychylováni z předpokládané pozice tečky. Spolu s méně přesným určováním horizontálních rozestupů je toto poněkud náchylná část na chyby v rozpoznávání.

Orientace tečky

Po nalezení předpokládané pozice bodu, opět prozkoumáme okolí, zda se zde nachází dostatek černých pixelů. Pokud ano provedeme ujištění o příslušnosti tečky ke správné straně dokumentu. Od středu černé plochy můžeme provést prozkoumání přítomnosti bílých pixelů na jedné či druhé polovině. Při dostatečné přesnosti bychom ani nemuseli zkoumat pozice světlých ploch, ale mohli bychom se rozhodovat pouze na základě pozic černých shluků (pixelů).

Dvojí průchod dokumentem

Přeložení přední strany dokumentu provádíme způsobem popsaným v podsekci 4.7 (Lokalizace a posun mezi body). Jelikož tečky patřící zadní straně se čtou z již obráceného listu. Zápis veškerých detekovaných znaků naležící zadní straně se jeví jako zrcadlově obrácený. Proto pro přeložení zadní strany musíme použít obrácený průchod. Po řádcích postupujeme zprava doleva, a také zmíněný posun v inicializaci musí být opačným směrem.

Alternativní způsob rozpoznávání

Kromě již zmíněného OBR v sekci 3.1 existují i další možnosti přístupu k problému převodu naskenovaných Braillových dokumentů na text, které pracují spolehlivěji na obosutranných dokumentech. Například provádění průchodu obrazu po řádcích pixelů a zaměření se na detekování Braillových bodů. Podobné způsoby jsou ale popsány v ostatních pracích.

Kapitola 5

Implementace

Tato bakalářská práce byla výhradně napsána v programovacím jazyce C++ s využitím knihovny pro práci s obrazem *OpenCV*. Pro vytvoření grafického uživatelského rozhraní byl použit oblíbený *Qt framework*. S ohledem na Qt framework jsem jako vývojové prostředí zvolil Qt Creator od společnosti Nokia.

5.1 OpenCV

OpenCv (Open Source Computer Vision) je multiplatformní knihovna pro práci s obrazem, napsána v jazycích C a C++. Tedy knihovnu lze využít v prostředí těchto jazyků. Dále také existuje možnost použití v Pythonu, Matlabu a případně i v jiných jazycích. Její implementace jsou dostupné na operačních systémech Windows, Linux i Mac OS X. OpenCV klade důraz mimo jiné na efektivitu výpočtu a aplikace pracující v reálném čase. Celá knihovna obsahuje na 500 funkcí se zaměřením na mnoho oborů – počítačové vidění, medicína, bezpečnost, robotika, strojové učení... [1].

Použití OpenCV v programu

V této práci je však využita pouze základní funkčnost knihovny. Jednu z nejvýraznějších pomocí nám OpenCV poskytne při práci s různými formáty obrazu. Konkrétně se jedná o toolkit HighGUI, který obsahuje potřebné funkce a je zahrnut v balíčku OpenCV. Příklad podporovaných formátů: BMP, DIB, JPEG, PNG, PBM, TIFF, ... Ať se jedná o kterýkoli podporovaný formát, nakládáme s obrazem jednotným způsobem přes ukazatel na strukturu *IplImage*. Je třeba si uvědomit, že v OpenCV neexistuje žádný objektový přístup jako zapouzdření apod. Datové struktury nejsou implementovány pomocí tříd, ale struktur. Všechny položky mají modifikátor přístupu *public*. K informacím o obrazu (počet barevných kanálů, hloubka, rozměry,...), stejně tak k obrazovým datům můžeme přistupovat přímo přes ukazatel na strukturu. Ke zpřístupnění pixelu šedotónového obrazu používám funkci *cvGetReal2D*.

Funkce *cvLoadImage* nám provede načtení obrázku. Při načítání *cvLoadImage* nebere v úvahu příponu názvu souboru. Místo toho analyzuje posloupnost prvních bytů a rozhodne se k použití vhodného kodeku. Prostřednictvím nastavení druhého parametru na `CV_LOAD_IMAGE_GRAYSCALE` zajistí načtení obrázku přímo v odstínech šedi (načtený obrázek *IplImage* bude mít pouze jeden osmi bitový kanál). Další funkce, které nám výrazně pomohou, pracují nad daty této struktury *IplImage*. V sekci 4.1 použijeme funkce OpenCV

k ořízntutí obrazu. Na Gaussovo vyhlazení existuje funkce *cvSmooth* s nastaveným parametrem `CV_GAUSSIAN`. Prahování jsem provedl pomocí *cvAdaptiveTreshold*, obojí popisuji v 4.1. K otočení obrazu (sekce 4.2) musíme vypočítat rotační matici funkcí *cv2DRotationMatrix*. Matice se následně aplikuje na obrazová data přes funkci *cvWarpAffine*. Pro dealokaci nepotřebných obrázků slouží funkce *cvReleaseImage*. To byl základní přehled OpenCV metod používaných v této práci.

5.2 Qt framework

Qt je podobně jako OpenCV multiplatformní knihovna, dostupná na desktopových i mobilních systémech. Převážně se používá pro vytváření programů s grafickým uživatelským rozhraním. Qt framework není pouze knihovna pro uživatelské rozhraní, obsahuje také velmi rozsáhlou dokumentaci. Dokáže nahradit i STL (Standard Template Library – Standardní knihovna šablon jazyka C++) [4]. Jelikož celý program je napsán v Qt frameworku, nebudeme využívat STL, ale použijeme moduly, které nám Qt nabízí. Najdeme zde totiž téměř vše potřebné včetně tříd pro práci s XML dokumenty, dynamických kolekcí, podpory pro paralelní řízení výpočtu apod.

Použití Qt frameworku v programu

Správná zásada psaní programů nám říká, že aplikace by neměla přestat odpovídat na vstup od uživatele. Pokud potřebujeme delší čas na provedení úlohy, měli bychom spustit výpočet na pozadí ve druhém vlákně (případně procesu) a informovat uživatele o průběhu výpočtu například pomocí progress baru. Dalším dobrým zvykem je umožnit uživateli kdykoli přerušit výpočet. Qt poskytuje k tomuto účelu třídy *QFuture* a *QFutureWatcher*. *QFuture* reprezentuje výsledek asynchroního počítání. *QFutureWatcher* zase dohlíží a monitoruje *QFuture*, umožňuje nastavit příznak *Cancel* pro přerušení výpočtu.

Nejdéle trvá operace zarovnání dokumentu, čas potřebný ke správnému otočení obrazu je dán především počtem iterací natáčení a také velikostí rozlišení obrazu. Aby se urychlila časově náročná operace zarovnávání dokumentu, použiji zde také rozvětvení výpočtu do dvou vláken. Každé z nich provede otočení obrazu na jednu ze dvou stran a spočítá ohodnocení na základě velikosti vertikálních maxim. Tento postup mohu provést díky tomu, že vlákna provedou kopii původního dokumentu a dále pracují pouze s vlastní lokální kopií dat a navzájem se nijak neovlivňují.

Veškeré seznamy o předem neznámé délce jsou implementovány třídní šablonou *QList*. Pro výpočet četností rozdílů maxim (4.4) používám kontejner *QMap*, kde hodnota klíče odpovídá rozdílu dvou hodnot a příslušná položka ke klíči vyjadřuje četnost hodnot. Pro parsování XML dokumentů (5.3) existuje v Qt frameworku spousta tříd, díky nimž je zpracování XML opravdu jednoduché a rychlé (*QDomDocument*, *QDomNode*, *QDomElement*, ...).

Grafické rozhraní aplikace

Hlavní část aplikace je tvořena grafickým prvkem umožňující výběr dokumentů (*QListWidgetItem*) a prvkem zobrazující přeložený text (*QTextEdit*). Pro jednoduchou manipulaci s nejčastějšími úkony je přítoment toolbar. Do aplikace jsem zahrnul dva dokovací widgety (*QDockWidget*). První poskytuje přehled statistik přeložených dokumentů jako počet znaků, počet jednotlivých teček v aktuálním dokumentu i ve všech doposud přeložených

dokumentech. Druhý ukazuje aktuálně vybraný dokument, ve kterém si můžete nechat zakroužkovat Braillovy znaky na základě vybraného přeloženého textu.

Obraz nastavuji jako pixmapu prvku *QLabel*. Vyskytuje se zde menší problém, popíši nástin jeho řešení. Převod obrazu ze struktury *IplImage* (OpenCV) na objekt typu *QPixmap*. Nenašel jsem žádnou funkci, která by prováděla převod přímo. Nejprve se musí vytvořit objekt *QImage* pomocí přetíženého konstruktoru, který příjmá jako jeden z parametrů ukazatel na buffer již existujících dat typu *unsigned char*. Protože v celé aplikaci od načtení obrazu funkcí cvLoadImage náš obrázek má pouze jeden barevný kanál (grayscale) a konstruktor objektu QImage příjmá pouze omezený počet formátů (mezi nimiž chybí náš požadovaný formát – z dostupných se mi nejvhodnější jevil `QImage::Format_RGB888`), musíme dočasně vytvořit pomocný obraz. Tento nový obraz bude obsahovat místo jediného barevného kanálu tři. Převodní funkce se jmenuje *cvCvtColor* a jak název napovídá nabízí nám ji OpenCV. Námi požadovaný převod zajistí parametr `CV_GRAY2RGB`. Jelikož objekt *QImage* v destrukci nemaže data v bufferu, nesmíme zapomenout uvolnit paměť ručně, jinak by při požadavku na každé překreslení obrazu vznikaly nemalé paměťové úniky (memory leak). Získání objektu *QPixmap* je už jednoduché přes metodu *QPixmap::fromImage*. Je třeba podotknout, že paměťové nároky na zobrazení dokumentu jsou překvapivě celkem vysoké. Proto vždy při skrytí náhledu na dokument je vhodné dealokovat prostředky. Také

Obrázek 5.1: Screenshot aplikace se dvěma dokovacími widgety pro zobrazení dokumentu a statistik

nastavování změny velikosti náhledu obrázku metodou *resize* objektu *QLabel* není příliš rychlé.

Na obrázku 5.1 je zachyceno hlavní okno aplikace. Většina z použitých ikon v aplikaci byla získána z balíčku Visual Studio Image Library.

5.3 Mapování znaků

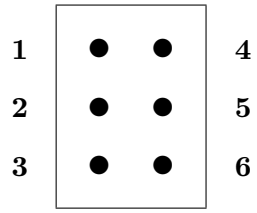
Abychom mohli interpretovat výsledky, musíme znát význam všech Braillových znaků. Každý národ má odlišný jazyk a také rozdílnou abecedu. Jak bylo zmíněno dříve i mezi českou a slovenskou normou je až sedm rozdílů. Mapování textových znaků na znaky Braillova písma by nemělo být zapsáno přímo do kódu programu. Jedna z možností je vytvoření XML dokumentů. Každý jeden dokument definuje mapování a částečně i sémantický význam znaků.

Popis struktury XML dokumentu

Root element je označen jako *alphabet*, jeho jediný atribut *language* obsahuje textový řetězec s názvem jazyka dokumentu. Určil jsem celkem sedm typů elementů:

1. *B* – pro následující znak bude použita jeho velká reprezentace – prefix velkého písmene.
2. *BS* – celá následující sekvence znaků bude vytištěna velkým písmem – prefix sekvence velkých písmen.
3. *S* – prefix malého písmene.
4. *N* – číselný prefix.
5. *GS* – prefix pro malé řecké písmeno.
6. *GB* – prefix pro velké řecké písmeno.
7. *symbol* – textový znak.

Potřebujeme popsat (identifikovat) pozice teček uvnitř Braillova znaku. Jedna z možností je použití šesticiferného binárního čísla. Přítomnost číslic 0 a 1 by informovala o vytlačeném bodu. Jiný často používaný přístup k identifikaci je použití posloupnosti čísel 1 až 6 (obrázek 5.2), kde přítomnost čísla v posloupnosti značí přítomnost bodu [9]. Stejný popis je využit i v XML dokumentu, kdy všechny znaky musí obsahovat v atributu *id* řetězec číslic určující strukturu znaku. V úvahu se bere pouze přítomnost číslic, né jejich vzájemná pozice. Speciální hodnota 0 odpovídá znaku bez jediného bodu (typicky mezeře).

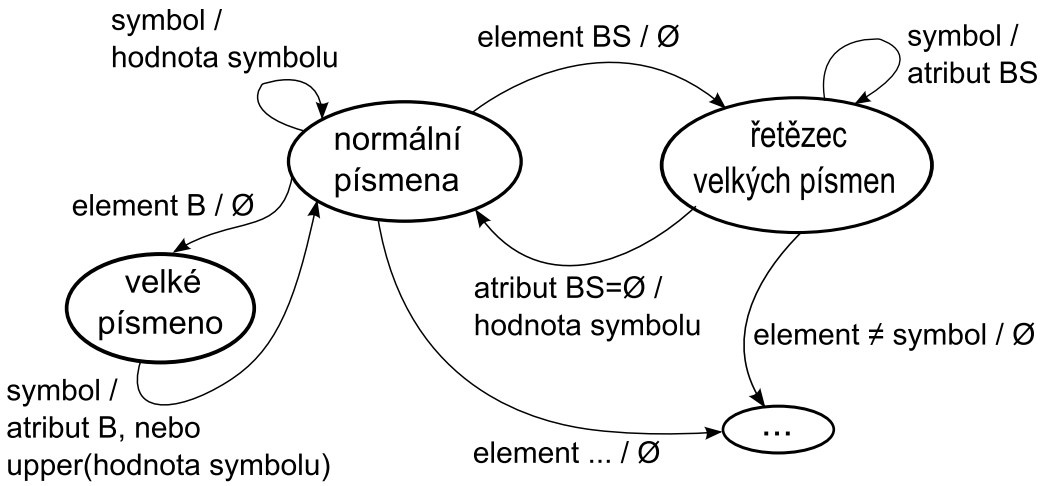


Obrázek 5.2: Číslování teček ve znaku

Prefixy Braillovy abecedy mění pouze sémantiku znaků následujících. Obsah jejich elementů, stejně tak i přítomnost jakýchkoli dalších atributů (kromě atributu *id*), ztrácí svůj význam.

Určení výsledného písmene

Vzhledem k přítomnosti prefixů v Braillově abecedě, nemůže existovat přímé mapování Braillových znaků na odpovídající písmena. Před vytištěním znaku použiji jistou formu konečného automatu, část jeho sémantiky je přítomna v konfiguračním souboru. Každý element může obsahovat atributy stejného jména, jako zkratky prefixů uvedené výše. Pokud se konečný automat nachází v některém ze stavu zpracování prefixu, potom pro každý další symbol využije hodnotu zapsanou v odpovídajícím atributu. Nepřítomnost vyžadovaného atributu způsobí ukončení platnosti prefixu a přechod konečného automatu do výchozího stavu. Tímto chováním je dána možnost uživateli si částečně přizpůsobit chování programu vůči prefixům – hlavně jednoznačné délky jejich platnosti. Platnost jednoho prefixu končí mimo jiné i při výskytu dalšího prefixu.



Obrázek 5.3: Část konečného automatu

Na obrázku 5.3 je vidět část automatu, který popisuje chování v rámci příchodu prefixu velkých písmen (*BS*) a prefixu velkého písmene (*B*). Značka *symbol* vyjadřuje obyčejný znak Braillovy abecedy. Zadání atributu *B* není povinné protože platnost prefixu velkého písmene se vztahuje pouze na jeden následující znak a ten je možné získat jednoznačným převodem znaku na jeho odpovídající velkou reprezentaci.

Následuje ukázka části XML dokumentu.

```
<alphabet language='Czech'>
    <B  id = '6' />
    <BS id = '56' />
    <S  id = '5' />
    <N  id = '3456' />
    <GS id = '45' />
    <GB id = '46' />

    <symbol id = '0' > </symbol>
    <symbol id = '1' B='A' BS='A' N='1' GS='α' GB='A' >a</symbol>
```

```
    <symbol id = '12' B='B' BS='B' N='2' GS='β' GB='B' >b</symbol>
    ...
    <symbol id = '456' >|</symbol>
    <symbol id = '4' >'</symbol>
</alphabet>
```

5.4 Programová implementace

V této části popíši nejdůležitější programové aspekty mého řešení. Jazyk C++ svým zaměřením náleží především k objektově orientovanému paradigmatu. Tedy celý program je složen z objektů patřící třídám. Následuje stručný popis programových tříd.

Image

Třída zapouzdřuje strukturu *IplImage* a některé další OpenCV metody pro práci s touto strukturou. Metody třídy se zaměřují na splnění výchozích požadavků zmíněných v sekcích 4.1, 4.2 (redukce barevného prostoru a zarovnání obrazu), prostřednictvím funkcí OpenCV popsaných v podsekci 5.1. Konkrétní názvy metod třídy *Image* volající OpenCV funkce jsou *CutOff*, *Treshold*, *Smooth*, *Align*, *GetPeaks*.

Image implementuje také zmíněný převod obrazu (podsekce 5.2) z typu *IplImage* do objetku *QImage* pomocí metody *CreateQImage*.

Token

Objekt třídy *Token* určuje právě jeden význam znaku Braillovy abecedy. Jeho vnitřní stav obsahuje příznak o typu (prefixu, či obyčejném znaku) a hodnoty datového typu *QString*, kterých nabývá znak v různých prefixových módech. Tyto hodnoty budou tisknuty na výstup.

Tokens

Jedná se víceméně o pole 64 tokenů (2^6 možných kombinací Braillových znaků).

XmlRead

Slouží k načtení a zpracování XML konfiguračního souboru. Jelikož XML soubor si může každý uživatel vytvořit a přizpůsobit sám, bude pravděpodobně obsahovat ze začátku několik chyb: chybná struktura XML dokumentu, špatně zadaná id čísla znaků, případně opakující se čísla atributu id, apod. Vždy při pokusu o neúspěšné nastavení nového konfiguračního souboru metoda objektu XmlRead způsobí vyjímku typu XMLException, která obsahuje textový řetězec, který popisuje příčinu chyby.

Některé znaky mohly být opomenuty ve specifikaci XML dokumentu. Při překládání bychom tedy tyto znaky neviděli. Abych upozornil na tuto skutečnost, XmlRead vynutí existenci všech 64 tokenů. Těm tokenům, které nebyly specifikovány přiřadí výchozí hodnotu "?". Navíc uloží všechny hlášky o chybějícím mapování. Upozorňující zprávy jsou dostupné přostřednictvím metody *GetWarnings*. Vzhledem k možnosti mapování řeckých písmen jsem zvolil kódování textových řetězců v UTF-8.

Objekt třídy *XmlRead* vytvoří objekt *Tokens*. *XmlRead* při vytváření nového mapování znaků vezme elementy XML dokumentu a pro každý z nich udělá následující: Vytvoří

objekt token patřičného typu dle typu elementu. Převede id znaku zadané ve formátu specifikovaném uživatelem v XML dokumentu (obrázek 5.2) na index používaný ve vnitřní reprezentaci. Tento index nabývá hodnot 0 až 63 a jeho přostřednictvím budeme indexovat daný token ve zmíněném poli (pomocí algoritmu 2 v sekci 4.5).

GeneralException

Všechny vyjímky v programu dědí od této jednoduché třídy obsahující pouze textovou zprávu o vzniklé chybě.

Symbol

Jakmile rozpoznám znak a zjistím jaký symbol (cokoli co bylo nadefinováno v XML souboru) má být vytisknut na výstup, vytvořím objekt třídy *Symbol*. Objekt je naplněn hodnotou typu *QString*, která se bude zobrazovat uživateli na výstupu. Navíc také obsahuje pole šesti ukazatelů na typ *QPoint*. Šestice bodů určuje pozice teček detekovaného Braillova znaku. Má-li bod hodnotu *NULL*, pak se příslušná tečka ve znaku nenachází.

DetectedObject

Podobně jako v případě tříd *Token* a *Tokens*, tak *DetectedObject* je kolecke ukazatelů na objekty *Symbol*. Protože nevíme kolik symbolů dokument tvoří, kolekce musí být dynamická (konkrétně `QList<Symbol *> *`). Mimo jiné objekt si uchovává a spravuje celý obrázek (třída *Image*). V programu se ve skutečnosti pracuje s polem objektů *DetectedObjekt*, které je aktuální po celou délku běhu aplikace.

Metoda *Trimmed* implementuje ořezávací mechanismus odstranění přebytečných znaků představený v podsekci 4.5.

Fsm

Konečný stavový automat – jeho vnitřní stav a kombinace vstupního tokenu určuje hodnotu z objektu *Token*, která se použije na výstup (obyčejný znak, číslo, vélké písmeno, ...). Parametrem metody *GetPrintableQString* je token, návratovou hodnotou je tisknutelný znak. V případě, že by token byl roven prefixu, bude pouze změněn vnitřní stav objektu *Fsm* a navrácen prázdný řetězec. V podstatě metoda *GetPrintableQString* implementuje konečný stavový automat popsaný v podsekci 5.3 a na obrázku 5.3.

Analyser

Analyser po předložení obrázku (objektu *Image*) provede zjištění vertikálních i horizontálních maxim pomocí metod objektu *Image*. Dále analyzuje vzdálenosti rozdílů maxim a zjistí hodnoty a, b, $c-a$, $d-2b$ (návrh nastíněn v sekci 4.4). Výpočet těchto vzdáleností se děje v metodě *doCalculations*. Z pozic prvních maxim (popsáno v sekci 4.3) začne prohledávat okolní body a překládat postupně celý dokument (sekce 4.5).

Objekt *Analyser* provádí rozpoznání dokumentu metodou *Detection*. Z dokumentu získává indexy do pole Tokenů. Pomocí *Fsm* vytváří tisknutelné řetězce. Naplní výstupní objekty třídy *Symbol*. Výstupem metody *Detection* je tedy seznam symbolů.

Uživatelské rozhraní

Zbylé třídy slouží k interakci s uživatelem. Klasické použití Qt frameworku (sekce 5.2): třídy dědí od předdefinovaných tříd uživatelského rozhraní (*QMainWindow*, *QDockWidget*, *QDialog*, *QTextEdit*, *QListWidget*) a přidávají k nim požadovanou funkčnost.

Zařadil jsem zde i třídu pojmenovanou *Worker*, která zajišťuje výsledný proces rozpoznávání dokumentů prostřednictvím metody *Detect*. Tato metoda provede všechny akce nad polem objektů *DetectedObject* nutné k převodu Braillova dokumentu na text. Kód této metody je vykonáván v dalším vlákně z důvodů popsaných v 5.2. Proto metoda *Detect* obsahuje navíc kód způsobující aktualizaci průběhu výpočtu a také podmínku pro předčasné ukončení překládání dokumentů.

Objekt třídy *Worker* vykonává další doplňující funkce a to uložení přeloženého textu do souboru/ů, vytvoření obrazu dokumentu se zakroužkovanými body.

Kapitola 6

Dosažené výsledky

V následující tabulce 6.1 vidíte přehled charakteristik dokumentů a dosaženou úspěšnost. Na testovací sadě bylo dosaženo kvalitních výsledků. Testovací dokumenty byly získány ze tří různých skenerů (v tabulce odděleny dvojitou čarou).

Dokument	Rozlišení $[DPI]$	Počet znaků[1]	Počet teček	Počet chybně rozpoznaných teček	Úspěšnost [%]
1	200	673	1638	1	99, 85
2	200	608	1499	1	99, 84
3	200	526	1304	0	100
4	???	784	1967	0	100
5	???	761	1832	0	100
6	100	761	1832	2	99, 74
7	200	761	1832	0	100
8	300	761	1832	0	100

[1] Včetně bílých znaků (mezer a nových řádků).

Tabulka 6.1: Úspěšnost rozpoznání jednostranně tištěných Braillových dokumentů

Dokumenty naskenované v menším rozlišení vykazovaly zvýšené množství chyb. V největší míře se jednalo hlavně o špatně detekované číslo stránky, spolu se zkratkou „str.“. Tyto znaky se nacházejí uprostřed strany. Způsobené chyby vznikly nepřesným odhadem parametrů dokumentu (kvůli menšímu rozlišení obrázku). Po provedení několika vodorovných krocích bylo odchýlení od předpokládaných pozic Braillových bodů příliš veliké. Jednotlivé tečky byly sice detekovány, ale byly přiřazeny ke špatným sloupcům (znakům). Díky použití dodatečného centrování pozic byla zbylá vetšina dokumentu rozpoznána správně.

Ostatní chyby byly způsobeny detekcí nesprávných bodů (poškození dokumentu), případně nebyly detekovány kvůli většímu jasu dokumentu. Chybám by se dalo předejít upravením parametrů programu (prahování, vyhlazení obrazu, práh černých pixelů určující tečku), nebo zvolením jasově vyváženějších dokumentů.

Rychlost převodu silně závisí na rozlišení obrazu a to hlavně kvůli prováděnému zarovnání dokumentu.

Kapitola 7

Závěr

Cílem práce bylo vytvoření čtečky Braillova písma z naskenovaných dokumentů. Na základě nastudování pravidel Braillova písma a úvodu do dané problematiky se mi podařilo navrhnout a implementovat vlastní způsob rozpoznávání Braillových dokumentů.

Podařilo se mi splnit všechny body body zadání. Na začátku práce jsem se seznámil s Braillovým písmem a jeho pravidly. Návrh aplikace umožňující rozpoznání Braillova písma v naskenovaném dokumentu na text je učiněn ve čtvrté kapitole. První bod zadání seznámení se s knihovnou OpenCV popisuji až v implementační části v páté kapitole, spolu s vlastní implementací aplikace a použití Qt frameworku. Celá šestá kapitola poskytuje přehled o dosažených výsledcích. Zbylý bod zadání o rozšíření programu diskutuji v této závěrečné kapitole.

Popsaný způsob rozpoznávání jednostranně tištěných dokumentů pracuje dobře. Ovšem pro oboustranné dokumenty by bylo zapotřebí uskutečnit ještě několik dalších úprav. Program při testování dokázal částečně rozpoznávat oboustranně tištěné dokumenty (ovšem vykazoval zvýšené chybovosti), ale často jsem musel provést ruční úpravu parametrů. Vzhledem k citlivosti a nepřesnostem určování vzdáleností mezi body jsem nezahrnul do výsledné aplikace podporu pro oboustranně tištěné dokumenty. Budoucí rozšíření programu by tedy spočívalo především ve zlepšení (návrhu) rozpoznávání oboustranně tištěnných dokumentů. Hlavně bych se zaměřil na získání přesnějších parametrů dokumentu a snížení citlivosti chyb na tyto parametry.

Literatura

[1] Bradski, G.; Kaehler, A.: *Learning OpenCV*. O'Reilly Media, první vydání, 2008, iSBN 978-0-596-51613-0.

[2] Kršek, P.: Základy počítačové grafiky: Studijní opora. [cit. 2012-01-20].

[3] Mennens, J., Tichelen, L. v.: Optical Recognition of Braille Writing Using Standard Equipment. *Rehabilitation Engineering, IEEE Transactions on*, ročník 2, č. 4, 1994, ISSN 1063-6528.

[4] Molkentin, D.: *The Book of Qt 4: The Art of Building Qt Applications*. No Starch Press, 2007, 440 s., iSBN 978-1-59327-147-3.

[5] Neovision: Optical Braille Recognition [online]. [cit. 2012-04-28].
URL http://www.neovision.cz/cz/prods/obr/

[6] Pecinovský, J.: *Skenery a jak skenovat*. Computer Press, a.s., první vydání, 2009, iSBN 978-80-251-2492-5.

[7] SONS: Česká slepecká Braillova abeceda - pravidla [online]. 2012 [cit. 2012-01-20].
URL http://www.sons.cz/braillska_abeceda_pravidla.php

[8] Wikipedia, the free encyclopedia: Louis Braille [online]. 2012-05-02 [cit. 2012-06-05].
URL http://en.wikipedia.org/wiki/Louis_Braille

[9] WWW stránky: Braille Cell Dimensions [online]. 2009-11-20 [cit. 2012-01-20].
URL http://www.tiresias.org/research/reports/braille_cell.htm

[10] WWW stránky: Working of Scanner [online]. 2010-03-16 [cit. 2012-05-05].
URL http://www.circuitstoday.com/working-of-scanner

[11] WWW stránky: Historie Braillova písma [online]. [cit. 2012-01-20].
URL http://i-bryle.cz/index.php?adr=8&docid=72

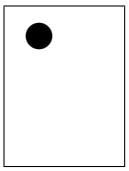
Příloha A

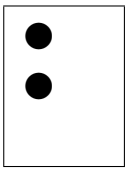
Obsah CD

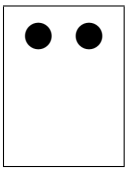
- **bin** – adresář se spustitelnou aplikací
- **latex** – adresář se zdrojovým textem bakalářské práce
- **src** – adresář se zdrojovými soubory aplikace
- **technická zpráva.pdf** – tato technická zpráva
- **test** – adresář s testovacími dokumenty

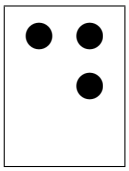
Příloha B

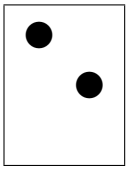
Česká Braillova abeceda

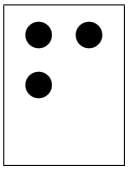
A, 1 

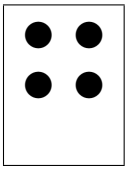
B, 2 

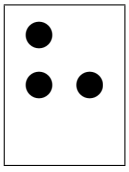
C, 3 

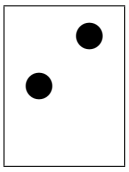
D, 4 

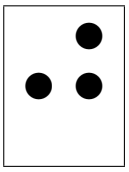
E, 5 

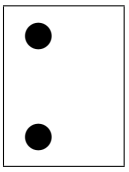
F, 6 

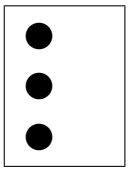
G, 7 

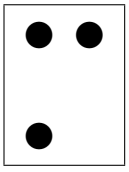
H, 8 

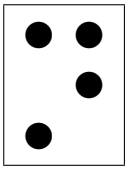
I, 9 

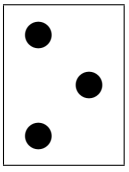
J, 0 

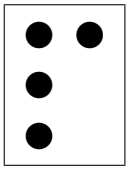
K 

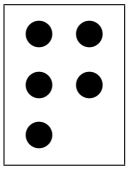
L 

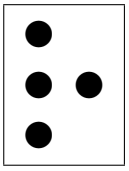
M 

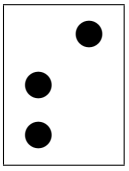
N 

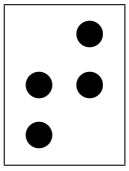
O 

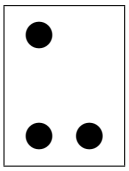
P 

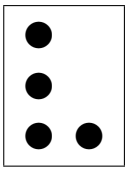
Q 

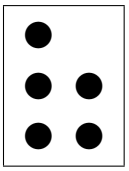
R 

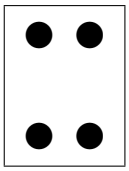
S 

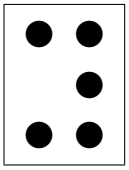
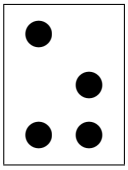
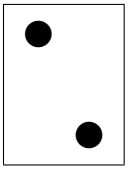
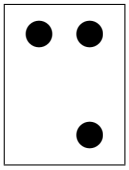
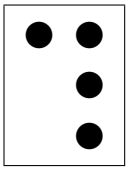
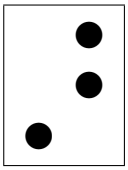
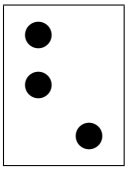
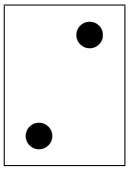
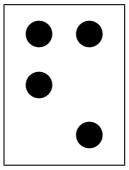
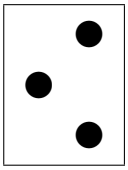
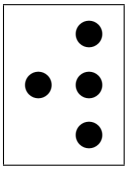
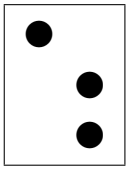
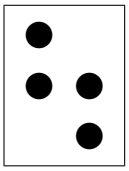
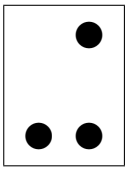
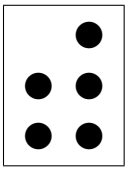
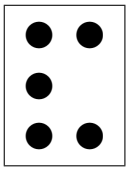
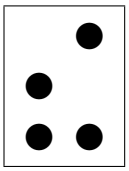
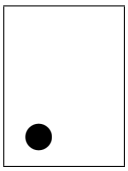
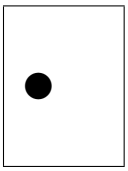
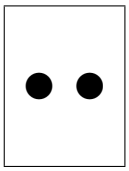
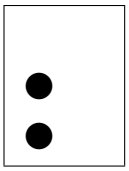
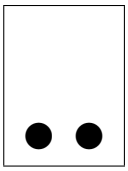
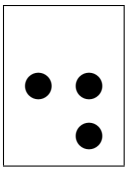
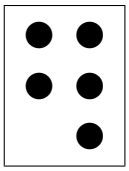
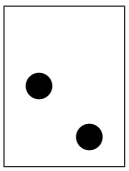
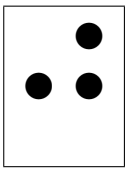
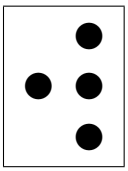
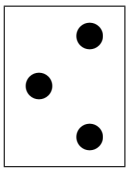
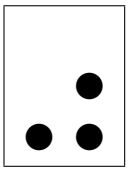
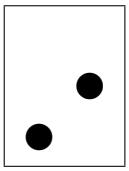
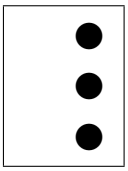
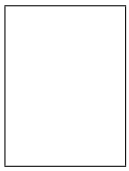
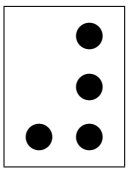
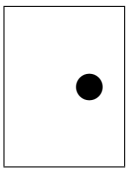
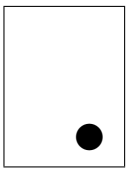
T 

U 

V 

W 

X 

Y		**Z**		**Á**		**Č**	
Ď		**É**		**Ě**		**Í**	
Ň		**Ó**		**Ř**		**Š**	
Ť		**Ú**		**Ů**		**Ý**	
Ž		.		,		:	
;		-		+		/	
?		!		“		(	
)		*		|		**mezera**	
číslo		**malé písmeno**		**velké písmeno**		**řetězec velkých písmen**