TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 Informační technologie
Studijní obor: 1802R007 Informační technologie

Grafické rozhraní pro OCR Tesseract

Graphical interface for Tesseract OCR

Bakalářská práce

Autor: **Jiří Beneš**
Vedoucí práce: doc. Ing. Josef Chaloupka, Ph.D.
Konzultant: Ing. Karel Paleček

V Liberci 23.4.2013

Vložte originální zadání...

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. O právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím práce.

Datum

Podpis

Poděkování

Rád bych poděkoval vedoucímu práce panu doc. Ing. Josefu Chaloupkovi, Ph.D. za cenné rady k dané problematice a veškerou pomoc v průběhu řešení práce.

Dále bych chtěl poděkovat rodině za podporu během studia.

Abstrakt

Tato práce je zaměřena na tvorbu aplikace, která poslouží jako grafické rozhraní pro konzolovou aplikaci Tesseract OCR. Vytvořené prostředí uživateli usnadní rozpoznávání textu v obrazových datech a též vytváření nových dat, sloužících k rozpoznávání.

V dnešní době je problematika OCR softwarů velmi aktuální a to už ať z pohledu vytváření knih pro, dnes velmi populární, elektronické čtečky knih nebo z pohledu využívání elektroniky v knihovnách popřípadě archivace. Tesseract OCR, vyvíjený firmou Google má velmi dobré výsledky, ovšem jeho ovládání pomocí příkazové řádky není vždy úplně snadné a následná úprava dat je často dosti zdlouhavá.

Aplikace je rozdělena do tří částí, z nichž každá využívá řadu modulů projektu Tesseract OCR k vytváření tréninkových dat nebo k rozpoznávání textu. Dovoluje též editaci vytvořených dat.

V rámci této práce byla testována funkčnost aplikace na vzorku obrazových dat a následně došlo k vyhodnocení. Pro této účel byla vytvořena aplikace pro hromadnou a automatickou úpravu výstupních dat tréninkové části aplikace.

Práce má pět části, z nichž první se zabývá projektem Tesseract OCR. Druhá, třetí a čtvrtá, vytvořeným prostředím a poslední potom testováním aplikace, kde je též řešena problematika kvality a rozměrů zdrojových dat.

Klíčová slova:

OCR, rozpoznávání textu, prostředí, obrazová data

Abstract

This thesis is focused on the creation of an application, which will be used as a graphical interface for the console application Tesseract OCR. This application will simplify the recognition of text in visual data and also creation of new data, serving for recognition.

In current times the issue of OCR softwares is very actual from two perspektives: creation of today very popular electronic book readers, and use of electronics in libraries or for the purpose of archiving. Tesseract OCR, which is being developed by Google, has very good effects, but his control with the command line is not always easy and following adjustment of data takes a lot of time. The application is devided into three parts, each of them uses a number of moduls of project Tesseract OCR for creation of training data or for text recognition. It also enables editing of created data.

Within this thesis the functionality of the application was examined on the sample of visual data and afterwards the evaluation was made. An application for mass and automatic adjustment of output data of the application training part was created for the purpose of this thesis.

The thesis has five parts. The first one deals with project Tesseract OCR, the second, third and fourth one with the created application and the last one with testing of the application. In this part there are also discussed the issues of quantity and the dimension of source data.

Key words:

OCR, text recognition, environment, image data

Obsah

Seznam obrázků

Seznam zkratek

OCR	Optical Character Recognition
OSD	Orientation and script detection
TIFF	Tag Image File Format
UTF-8	UCS Transformation Format
DAWG	Directed Acyclic Word Graph

Úvod

Problematika OCR aplikací je již několik let velmi oblíbeným tématem. Tyto aplikace jsou využívány k mnoha účelům, jedním z nich je například digitalizace knih nejen pro dnes již velmi rozšířené elektronické čtečky knih, ale i pro archivaci nebo vytvoření elektronické knihovny. Dalším místem, kde je OCR využíváno, jsou aplikace pro automatické čtení testovacích obrazových textů *captcha*, sloužící například k ověřování při odesílání formuláře. Ověřuje se tak, zdali je formulář vyplněn živou osobou, či internetovým robotem. Další možností využití OCR je při rozeznávání spamu v elektronické poště.

Tesseract OCR je jednou z aplikací, která je šířena pod licencí Apache, není tedy komerční. Drží se na vrchních pozicích žebříčku používaných OCR. Jedinou nevýhodou pro užívání v širším měřítku je jeho konzolová podstata. Z tohoto důvodu jsou komerční aplikace jako je ABBYY FineReader OCR nebo Readiris více v povědomí uživatelů, i přes nutnost jejich financování.

Z tohoto důvodu vznikla výše jmenovaná aplikace, která by měla být snadno ovladatelná, rychlá a měla by uživateli umožnit užívat Tesseract OCR bez nutných znalostí ovládacích příkazů.

1. Tesseract OCR

1.1. Historie

Vývoj Tesseract OCR započal v roce 1984 jako projekt doktorandského výzkumu v laboratořích HP Labs Bristol[1]. Společnost HP posoudila potenciál projektu jako velmi vysoký. Rozhodla se proto tedy projekt propojit se svou divizí vývoje skenerů v Coloradu. Díky tomu získal Tesseract významný náskok v přesnosti oproti konkurenčním komerčním softwarům, které v té době byly též v začátcích. Zároveň tím udala směr vývoje technologii skenování. Další vývoj Tesseractu byl prováděn v HP Labs Bristol a zabýval zlepšením rozpoznávacích schopností, aby bylo sníženo procento špatně přeložených znaků a dále kompresí dat.

V roce 1995 se zúčastnil Tesseract OCR testu přesnosti, Annual Test of OCR Accuracy[2] prováděný univerzitou University of Nevada Las Vegas ve spolupráci s Ministerstvem energetiky Spojených států amerických, kde překvapil svými výsledky. Poté se znovu vývoj ponořil do vývojových laboratoří HP Labs Bristol. V roce 2005 společnost Hewlett Packard uvedla Tesseract jako *open-source*.

Poté začala Tesseract OCR vyvíjet i společnost Google. Ve své verzi 2.00 již nabízela šest defaultních jazyků a též prostředky pro vytváření nových sad. Verze 3.00 přinesla vylepšené trénování dat a nové defaultní sady včetně jazyku, které běžně nepoužívají latinku jako korejština či čínština. Díky novému přístupu k samotným jazykům, je od této verze možno jazykové sady vkládat jako samotné soubory, které se užívají bez úpravy aplikace. V prosinci roku 2012 bylo k dispozici několik stovek různých znakových sad použitelných v aplikaci Tesseract OCR. Na tom měla zásluhu hlavně licence, pod kterou byla aplikace vyvíjena, a která umožňuje vytvářet znakové sady takřka komukoliv. Proto nejsou výjimkou ani znakové sady písem, které se již stovky let nepožívají např. hlaholice či cyrilice, jež byla využívána v době Velkomoravské říše v devátém století našeho letopočtu.

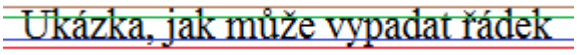
Poslední verze 3.02, vydaná v listopadu roku 2012, přidává schopnost více jazyků v jednom obraze a zlepšuje schopnost rozeznávání dat.

1.2. Funkčnost Tesseract OCR

Tesseract pracuje na principu rozeznávání shluků černé barvy na bílém podkladě. Algoritmus řádek po řádku hledá seskupení těchto shluků a tvoří z nich bloky odpovídající velikosti písma. Prvním krokem algoritmu je připojení popisu obrysu ke každému znaku. Podle těchto popsaných obrysů se pak dále vytvářejí kritéria pro jednotlivé znaky.

1.2.1. Hledání řádků a slov

O hledání řádků se stará tzv. přímkový algoritmus. Tento algoritmus hledá řádek nezávisle na tom, zdali je rovnoběžný s okrajem obrazu nebo je v jakémkoli jiném úhlu. Tím je uživatel ušetřen otáčení textu a nedochází tak ke snížení kvality obrazových dat. Algoritmus vyhledává linii diakritických znamének a jiných znaků, které by mohly být považovány za nepřesnosti vytvořené například skenováním. Každý řádek má čtyři důležité linky, viz Obr. 1.1. Zde je vidět, že diakritika je vždy mezi zelenou a hnědou linkou, ostatní shluky mimo tuto část jsou pouze šum. To samé platí pro části přesahující modrou linii směrem dolů.



Obr. 1.1. Znázornění čtyř linek, vedoucích znaky

Pro všechny tyto linky platí, že jejich vzájemná vzdálenost je stejná na celé jejich délce. To platí i pro texty psané například do kruhu.

1.2.2. Rozteč znaků

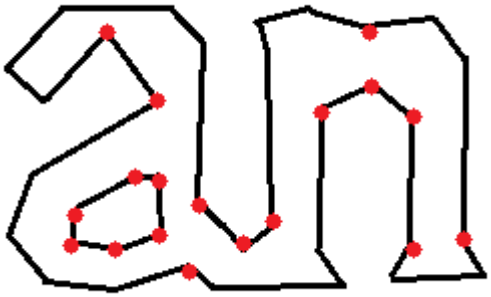
Algoritmus zjišťuje, zdali znaky v rozpoznávaném textu mají pevnou rozteč, pokud zjistí, že tomu tak je, automaticky vytvoří mřížku a nezabývá se rozpoznáváním hranic jednotlivých znaků. Tomuto se dá též předejít nastavením písma při rozpoznávání. Pokud je vybráno písmo, které je definováno jako písmo s konstantní šířkou znaků, není nutno tento algoritmus spouštět.

Pokud není u písma rozpoznána konstantní šířka, nastává nepříliš triviální úkol, nalezení hranic znaků. Velmi často se stává, že se znaky téměř dotýkají. V takovém

případě je přistoupeno k rozmazávání, tím se mezera mezi znaky zvětší a algoritmus má možnost rozeznat hranice.

1.2.3. Polygony ohraničení

Tesseract si pro každý nejasný shluk znaků vytvoří zjednodušený obrazec úseček, který jej ohraničuje. Následně hledá body (viz Obr. 1.2, červené body), které se do shluku znaků tzv. „zakrajují“, ty poté porovnává s charakteristikou ASCII znaků a podle toho vybírá nejpravděpodobnější rozdělení znaků.



Obr. 1.2. Zjednodušený obrys znaků „an“

1.3. Moduly aplikace Tesseract OCR

Program je rozdělen do několika částí, některé z nich slouží pro vytváření nových tréninkových dat, jiné pro samotnou funkci OCR. Celý software je ve formě konzolové aplikace, která je tvořená několika různými moduly. Všechny tyto moduly jsou vytvořeny v jazyce C/C++. Po nainstalování Tesseract OCR dojde k implementaci softwaru do operačního systému. Od této chvíle lze spouštět moduly bez znalosti jejich fyzického umístění v souborovém systému.

1.3.1. Modul *tesseract*

Rozklad textu OCR

Tento modul slouží právě jako samotné OCR. V základním tvaru volání má pouze dva argumenty jak lze vidět níže.

tesseract vstupni_data.tif vystupni_data [-l jazyk] [-psm N] [configfile]

První určuje zdrojová obrazová data, která musí být v obrazovém formátu TIFF a druhý udává umístění výstupních dat. Výstupní data jsou zapisována do textového souboru typu **.txt* s kódováním UTF-8.

Třetím volitelným parametrem modulu je jazyková sada, která se používá při rozpoznávání textu. Zvolená znaková sada musí být uložena v kořenovém adresáři aplikace Tesseract OCR a musí mít název ve formátu *jazyk.traineddata*. Je pravidlem označit jazyk třemi písmeny, např. *ces* nebo *deu*. V případě, že není jazyk zvolen, je defaultně zvolena integrovaná sada anglických znaků.

Dalším volitelným parametrem je režim, ve kterém má být rozpoznávání spuštěno. Tedy N je nahrazeno některým z čísel 0–10. Dané číslo odpovídá požadované akci. Zde je možno nastavit směr čtení znaků, popřípadě formu textu a podobně.

Posledním parametrem je konfigurační soubor, kterým je možné povolit nebo zakázat různé další funkce aplikace. Jednou z nejčastěji používaných funkcí je například potlačení defaultního nastavení. Níže je seznam možných funkcí.

0 - pouze OSD
1 - automatická segmentace stránky pomocí OSD
2 - automatická segmentace stránky bez použití OSD nebo OCR
3 - plně automatická segmentace stránky bez použití OCR (defaultní)
4 - data jsou sloupec textu s proměnnou velikostí písma
5 - data jsou jeden jednotný blok svisle zarovnaného textu
6 - data jsou jeden jednotný blok textu
7 - data jsou pouze jeden řádek textu
8 - data jsou pouze jedno jediné slovo
9 - data jsou pouze jediné slovo zapsané v kruhu
10 - data jsou pouze jeden znak

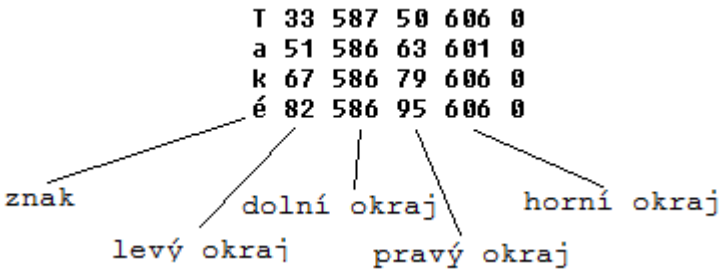
Vytváření informací o poloze znaků

Pomocí konfiguračního souboru lze v modulu *tesseract* spustit proces, kdy je modul použit pro vytváření souborů s daty o poloze znaku v daném souboru TIFF.

Ten zapisuje do souboru každý znak na jeden řádek a přidává k němu jeho pozici na obrázku. Pozice je měřena od levého dolního rohu. Tvar příkazu pro spuštění je zobrazen níže. Zde je *makebox* konfigurační soubor a *batch.nochop* parametr, označující tvorbu souborů **.box*.

tesseract *vstupni_data.tif* *vystupni_data* batch.nochop makebox

Data jsou následně užívána celou řadou dalších modulů aplikace Tesseract OCR. Příklad, jak může vypadat několik řádků souboru s informacemi o pozici, představuje Obr. 1.3. První a třetí číslo v řádku značí začátek a konec znaku od levého okraje obrázku. Druhé a třetí potom ukazuje začátek a konec od dolního okraje. Tím je přesně definována pozice znaku pro další využití.



Obr. 1.3. Příklad struktury řádků souboru **.box*

Vytváření informací o znaku

Další případ, kdy je modul *tesseract* spuštěn se vstupním konfiguračním souborem, konkrétně *box.train* nebo *box.train.stderr*. Ve vytvořeném souboru se nacházejí data určující přímky, které ohraničují znak. Tato data jsou zapsána pro každý znak zvlášť. Soubor obsahuje informace o všech znacích na zdrojovém souboru TIFF. Příklad struktury souboru, viz Příloha A.

tesseract *vstupni_data.tif* *vystupni_data* nobatch box.train

1.3.2. Modul *unicharset_extractor*

Pro vytváření tréninkových dat je třeba znát množinu znaků, které jsou obsaženy ve zdroji. Tuto informaci vytváří modul *unicharset_extractor*. Jediným parametrem modulu je množina všech souborů s informacemi o pozicích.

unicharset_extractor *vstupni_data00.box vstupni_data01.box ...*

Modul postupně projde všechny vstupní soubory a od každého znaku si uloží vždy jeden exemplář. Po dokončení vytvoří soubor *unicharset*. Příklad struktury dat, viz Příloha B.

1.3.3. Modul *shapeclustering*

Tento modul byl přidán v poslední verzi programu Tesseract OCR 3.02. Slouží k vytvoření prototypu tvarů jednotlivých znaků. Data čerpá ze souborů **.tr*, výstupem je poté soubor *shapetable*, který obsahuje prototyp každého znaku znakové sady. Modul má tři parametry, prvním je vstupní soubor *font_properties*, ten nese informaci o fontu písma použitého ve zdrojových souborech TIFF. Druhý soubor *unicharset* je též vstupní. Je to soubor generovaný modulem *unicharset_extractor*. Posledním parametrem je množina všech souborů typu **.tr*.

shapeclustering -F font_properties -U unicharset *vstupni_data00.tr vstupni_data01.tr ..*

Soubor *font_properties* obsahuje název fontu a informace o stylech v pořadí: *italic, bold, fixed, serif* a *fraktur*.

timesitalic 1 0 0 1 0

Obr. 1.4. Příklad struktury souboru font_properties

V souboru je pro jednotlivé vlastnosti zapsána buď 0 nebo 1, podle toho zdali danou vlastnost písmo má nebo ne, viz Obr. 1.4.

V případě, že typ písma není zřejmý, nebo nelze rozeznat vlastnosti, je možné nastavit písmo, jako neznámé, viz Obr. 1.5. Ačkoli je soubor *font_properties*, není nutno v něm typ uvádět.

unknownfont 0 0 0 0 0

Obr. 1.5. Příklad neznámého fontu

1.3.4. Modul *mftraining*

Tento modul zpracovává soubory **.tr*, generuje z nich troje výstupní data. Prvním souborem je *inttemp*. Ten obsahuje tvarové prototypy znaků, což jsou shluky bodů, které jsou pro daný znak specifické, a podle kterých lze tento znak odlišit od ostatních. Dále *pffmtable*, ve kterém je uložen pro každý znak údaj o počtu očekávaných funkcí, určujících hrany. Posledním souborem je vylepšená znaková sada *xxx.unicharset*, která je užívána pro konečné kompletování dat. Trojice písmen *x* naznačuje umístění označení jazyka jeho zkratkou např. *cze* nebo *eng*.

mftraining -F font_properties -U unicharset -O *xxx*.unicharset *vstupni_data .tr*

1.3.5. Modul *cntraining*

Další část aplikace, která je využívána pro tvorbu nové znakové sady. Vstupem je množina souborů **.tr*, jak lze vidět níže. Z nich je poté vytvořen soubor *normproto*, obsahující pro každý znak prototyp s normalizovanou citlivostí pro rozeznávání.

cntraining *vstupni_data00.tr vstupni_data01.tr ...*

1.3.6. Modul *combine_tessdata*

Kombinování dat

Combine_tessdata je nejdůležitější modul pro tvorbu nové znakové sady. Kombinuje všechna data předchozích modulů a vytváří z nich soubor *xxx.traneddata*, který může být využit aplikací Tesseract OCR k rozeznávání textu v obrazových datech.

Aby modul správně pracoval, musí adresář, pro který je spuštěn, obsahovat všechna tato data:

- *xxx.shapetable* – vytvořeno modulem shapeclustering
- *xxx.normproto* – vytvořeno modulem cntraining
- *xxx.inttemp* – vytvořeno modulem mftraining
- *xxx.pffmtable* – vytvořeno modulem mftraining
- *xxx.unicharset* – vytvořeno modulem unicharset_extractor

Jak je naznačeno, soubory musí být přejmenovány tak, aby jména všech souborů obsahovala tři písmena označující jazyk, např. *ces* nebo *deu*. Pokud je toto splněno, lze modul spustit. Jeho jediným parametrem je zkratka jazyka opatřená tečkou na konci, příklad níže. Místo slova *dir*, musí být uvedena celá cesta k adresáři s daty.

combine_tessdata */dir/xxx.*

Extrahování dat z *traineddata* souboru

Modul umožňuje extrahovat dvě komponenty, *unicharset* a konfiguraci pro daný jazyk. To se dá využít třeba pro vytváření více znakových sad pro jeden jazyk, ale pro více fontů. V tomto případě má modul tři parametry, prvním je vstupní soubor *xxx.traineddata*, a další dva jsou potom výstupní soubory *xxx.config* a *xxx.unicharset*.

combine_tessdata -e *xxx.traineddata xxx.config xxx.unicharset*

1.3.7. Modul *ambiguous_words*

Tento modul generuje sadu slov, které považuje za nejednoznačné. Tedy slova, u kterých je jistá možnost záměny písmen. Vygenerovaný seznam se dá zakomponovat do tréninkových dat, není však povinný. Spouštění níže.

ambiguous_words -l *tessdata_dir wordlist ambiguosfile*

1.3.8. Modul *wordlist2dawg*

Slouží ke konverzi seznamu slov do tzv. DAWG, to je orientovaný acyklický graf slov. Seznam je komprimován a efektivně, tak šetří prostor i čas. Tento modul není jedním z nutných modulů pro tvorbu dat, jeho použití je nadstavbové. Příklad, jak lze modul spustil.

wordlist2dawg *wordlist dawg xxx.unicharset*

2. Grafické rozhraní

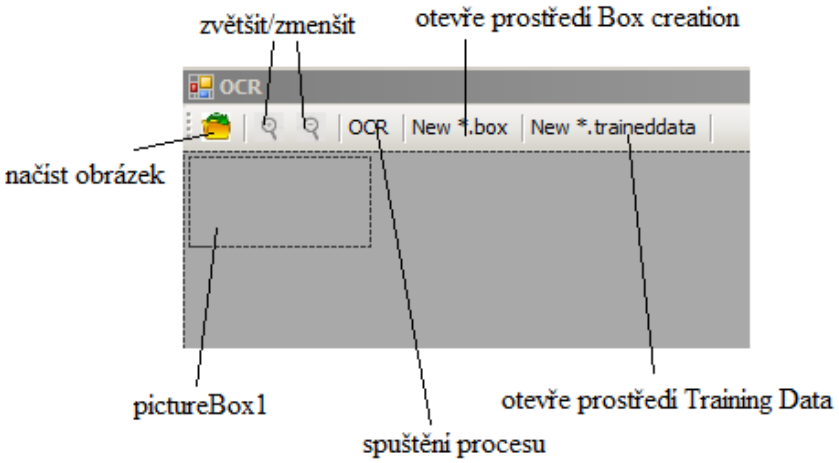
2.1. Prostředí *OCR*

Aplikace se skládá z několika částí. Prvním krokem bylo vytvoření prostředí pro rozpoznávání textu v obrazových datech. Požadavkem bylo vytvořit ho tak, aby bylo co nejpříjemnější uživateli. Proto bylo rozděleno do dvou částí pomocí prvku *spiltContainer1*. Levá strana zobrazuje obrázek a pravá rozpoznaný text. O změnu velikosti všech grafických objektů, při změně velikosti okna, se stará událost *SplitterMoved*, která je vyvolána vždy, když *Splitter* jakkoli změní polohu. V události je spuštěn algoritmus na přepočítávání polohy a rozměrů všech prvků prostředí tak, aby nebyla při zvětšení okna některá příliš velká a jiná naopak malá. Objektem *Splitter* lze též svévolně hýbat a měnit tak pouze poměr plochy obrázku vůči přeloženému textu.

Poté bylo nutno vyřešit otázku zobrazení obrazových dat. Prvním pokusem bylo vykreslit je přímo na pracovní plochu aplikace, ovšem toto řešení se ukázalo jako zcela nevhodné. Vykreslování bylo velmi pomalé a zatěžovalo celou aplikaci natolik, že na kratší dobu přestávala pracovat, což je nežádoucí. Proto bylo přistoupeno ke grafickému objektu *pictureBox1*. Ten ulehčil práci tím, že již sám dokáže využívat vlastností obrázku a nezatěžuje tak aplikaci obtížným vykreslováním. *PictureBox1* byl nastaven tak, aby přizpůsoboval své rozměry pracovní ploše prostředí, tím byl dán prostor i pro případné zvětšení obrázku. Toto nastavení lze provést pomocí vlastnosti *SizeMode* a hodnoty *Zoom*. K zvětšování obrázku je použito pouhé zvětšení rozměru *pictureBox1*. Pro posouvání po zvětšeném obrázku byla nastavena objektu *splitContainer1.Panel1* vlastnost *AutoScroll* na hodnotu *true*. Ta zobrazí posuvníky vždy, když rozměr obsahu přesáhne rozměr samotného *splitContainer1.Panel1*.

Celé prostředí nemá defaultně nastavené rozměry. Ty se přizpůsobují až ve chvíli, kdy je aplikace spuštěna, popřípadě kdy je změněn její rozměr. Díky tomu je možné s aplikací pracovat na jakkoliv velké pracovní ploše.

Menu aplikace bylo vytvořeno pomocí prvku *toolStrip1*. Menu obsahuje několik ovládacích tlačítek. Jsou zde také tlačítka pro spuštění dalších dvou částí projektu, viz Obr. 2.1.



Obr. 2.1. Popis menu prostředí OCR

Pro zvolení otevíraného obrázku byl vybrán klasický *openFileDialog1*. Na začátku práce na této aplikaci byla zvolena pro formátování obrázku do formátu TIFF přímo metodami jazyka C++, později se však ukázalo, že takto formátované soubory nevyhovují parametrům, které Tesseract OCR požaduje. Z toho důvodu mohou do aplikace vstupovat pouze soubory typu TIFF předem formátované. Byl nastaven filtr pro otevírání souborů, aby bylo možné otevírat pouze podporovaný formát TIFF, což znemožní uživateli vkládat do aplikace jiné formáty a zároveň zabezpečí správnou funkčnost aplikace. K načítání je používána funkce obsažená ve třídě *Image*, *Image::FromFile()*. Do ní je vkládána cesta k souboru jako parametr. Po načtení se obrázek zobrazí v prvku *pictureBox1* s velikostí přizpůsobenou prostředí. Aplikace si obrázek ukládá do globální proměnné datového typu *Image*. Děje se tak pro případné pozdější využití, například pro možnost zjištění velikosti, při počítání přiblížení.

Jazyková sada, kterou je nutné zvolit pro rozpoznávání textu, se vybírá v pravé části aplikace. Seznam jazykových sad je načítán ze souboru *c:\Program Files\Tesseract-OCR\tessdata\all.lang*. K tomuto řešení bylo dospěno po nezdařeném pokusu o automatické načítání jazykových sad. Myšlenka byla založena na procházení názvů všech souborů kořenového adresáře aplikace Tesseract OCR. Toto řešení však vykazovalo mnoho chyb u operačních systémů Microsoft Windows XP

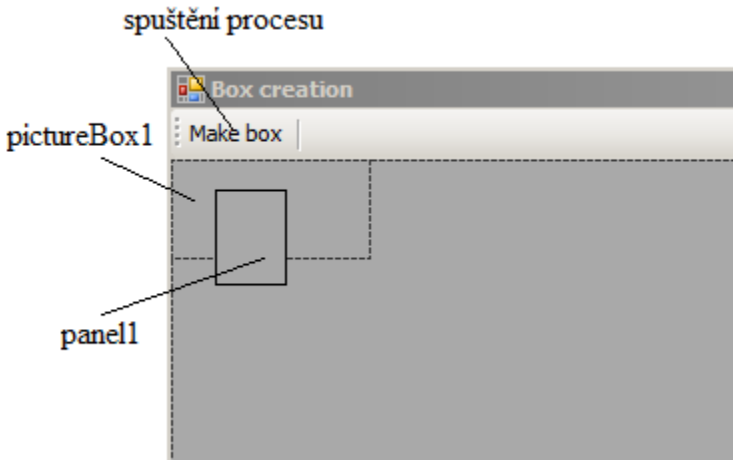
a Microsoft Windows 8, proto od něho bylo ustoupeno a bylo nahrazeno již zmíněnou metodou. Soubor se zkratkami jazykových sad musí uživatel editovat ručně.

Pro zobrazení přeloženého textového souboru byl vybrán *richTextBox1*. Do něho jsou data načítána a je zde možnost text případně upravovat a nakonec uložit. Z tohoto základního prostředí je možno přestupovat k dalším oknům aplikace, *Box creation* a *Training data*.

2.2. Prostředí *Box creation*

2.2.1. Prostředí

Prostředí pro vytváření souborů s informacemi o pozicích znaků, bylo vytvořeno jako podokno aplikace OCR. Toto okno slouží k vytváření a editaci souborů pro vytváření nové znakové sady pro Tesseract OCR. Prostředí je znovu rozdělené do dvou částí pomocí prvku *splitContainer1*, v levé straně je zobrazen obrázek, na kterém je vždy zvýrazněn zvolený znak. Na horní levé straně je grafický objekt *dataGridView1*, ve kterém jsou vypisovány všechny znaky obsažené v obrázku. Dolní levá strana obsahuje prvky k editaci, mazání či přidávání znaků. Ke spuštění procesu vytváření souboru s daty je určeno tlačítko *Make box*, to se jako jediné nachází v menu aplikace, viz Obr. 2.2.



Obr. 2.2. Popis prostředí *Box creation*

Prvním krokem bylo třeba zvážit, jak docílit nejvyššího pohodlí uživatele. Za tímto účelem byly testovány různé způsoby, jak zobrazit obrázek a přitom dát uživateli jasnou představu o zpracovávaných datech.

Prvním pokusem bylo zobrazit obrázek na panelu a znaky ohraničit pouhým vykreslením obdélníku okolo upravovaného znaku. Tato metoda se projevila jako nenáročná na vykreslování a bylo možné pozici měnit pomocí změny parametrů znaku. Nedávala však možnost fyzického posouvání přímo obdélníku pomocí polohovacího zařízení, protože obdélník byl tvořen pouze grafikou čar a neměl žádnou objektovou podstatu. Posun polohovacím zařízením byl jednou z priorit na počátku tvorby. Proto bylo přikročeno k nahrazení obdélníku dalším grafickým objektem *panel1*. Bylo však nutné vyřešit problém jeho průhlednosti, protože za předpokladu, že uživatel neuvidí přes co je *panel2* překryt, nemůže ho efektivně posouvat na daný znak. Za tímto účelem byly provedeny další pokusy s různými nastaveními objektu *panel2*, ve kterém byl zpracován obraz. Po vyzkoušení několika různých nastavení byl *pane12* pro zobrazení obrázku nahrazen *picturBox1*. Poté byly nastaveny parametry a z objektu *panel1* se stal rámec, jehož průhlednost byla svázána právě s *picturBox1*.

Jediná nevýhoda tohoto provedení je, že *panel1* zobrazuje obsah *pictureBox1* posunutý vždy o tloušťku svého orámování. Ovšem s tloušťkou *1px* to není vážné zkreslení.

Posouvání objektu *panel1* je zajištěno pomocí akce *MouseMove()*. V této akci bylo definováno, že pokud je stlačeno pravé tlačítko polohovacího zařízení, potom *panel1* kopíruje cestu kurzoru. Snímání pozice kurzoru je obstaráváno pomocí funkce *PointToClient()*, která přejímá jako parametr právě absolutní pozici kurzoru vůči celé obrazovce. Výstupem funkce je bod, který ukazuje na pozici kurzoru právě pro daný objekt, pro který je funkce volána.

Než bylo nalezeno toto řešení, docházelo k chybě zobrazování ohraničení znaku. Pokud byl obraz posunut do jiné části pomocí posuvníků, docházelo k chybnému počítání polohy, protože byla počítána pouze z absolutní polohy kurzoru vůči celé obrazovce. Tedy pokud byl zobrazen jako první řádek, ten který je na obraze jako třináctý a byl na něm vybrán znak, došlo zobrazení objektu *panel1* na prvním řádku na obraze. Tato chyba byla odhalena až ve chvíli testování aplikace na vzorku dat.

Pro lepší manipulaci je šipka kurzoru vždy mimo *panel1*, aby nepřekážela výhledu, a bylo tak možné přesně přemístit panel na požadovanou pozici. Je posunuta vždy k pravému dolnímu rohu obdélníku ohraničujícímu znak.

Pro spuštění procesu je nejprve nutné vytvořit dávkový soubor. Ten je generován podle jména vstupního souboru a jeho umístění.

Dávkový soubor je poté spuštěn tak, aby aplikace počkala na provedení a teprve poté načetla vytvořená data. Toto řešení bylo zvoleno proto, že proces vytváření souboru s informacemi o pozici písmen není okamžitý a data by nemohla být načtena. Třída *Process*, je jedním z několika možných řešení. Byla vybrána, jelikož všechny její metody přesně simulují chování příkazové řádky.

Načítání dat o vlastnostech znaků je ošetřeno pomocí odchytávání chyb. Pokud nelze soubor s daty otevřít, třída *StreamReader* vytvoří výjimku, ta je odchycena a spustí se vytvářecí proces přes Tesseract OCR. Následně je soubor načten. Bylo tak učiněno, aby nebyla přepsána již rozpracovaná data. Pro vytvoření nových dat stačí pouze odstranit soubor s daty, tedy soubor typu **.box* a též dávkovací soubor.

Po vytvoření dat, jsou tato data načtena a vepsána do objektu *dataGridView1*, zde jsou přehledně v řádcích předána uživateli k editaci. Samotná editace přes *dataGridView1*, byla zakázána, aby nedocházelo k rozdílným datům mezi zadávacím formulářem a tabulkou.

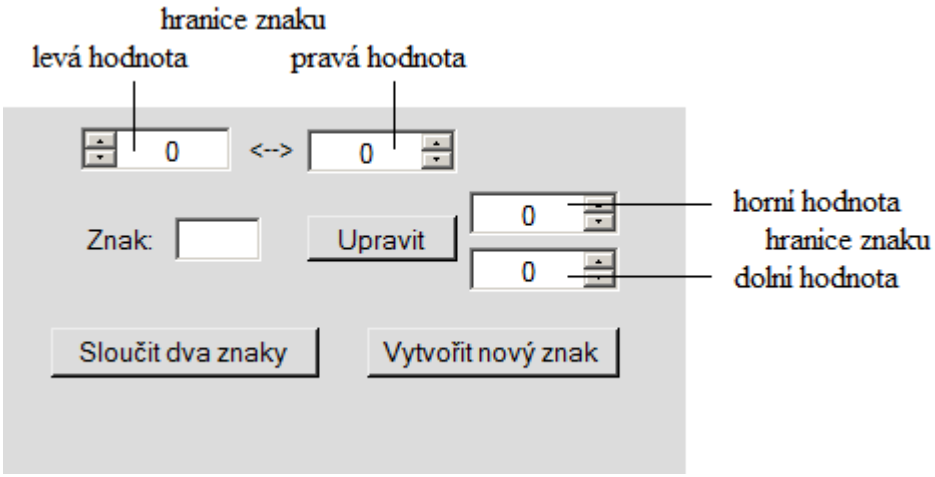
Pro editaci je nutné zvolit nejprve znak, a to lze klepnutím na daný řádek v tabulce nebo přímo na znak v obrázku. Při kliknutí do obrázku je podle pozice kurzoru a dat v tabulce nalezen odpovídající znak a je vybrán pro editaci. O nalezení se stará algoritmus vyobrazen na Obr. 2.3.

```
if((y1i<=x.Y)&&(x.Y<=y2i)){
    if((x1i<=x.X)&&(x.X<=x2i)){
        dataGridView1->ClearSelection();
        dataGridView1->Rows[index]->Selected = true;
        break;
    }
}
```

Obr. 2.3. Algoritmus vyhledávání znaku v tabulce

Zde je zjišťováno, zdali pozice kurzoru odpovídá hraničním čárám znaku. Algoritmus je volán postupně pro všechny znaky v tabulce, dokud není nalezen ten správný. Poté je tento znak zvolen k editaci.

Když je vybrán jeden ze znaků, jsou jeho data přenesena do formuláře, viz Obr. 2.4, kde je lze měnit. Zvětšování či zmenšování rozměrů znaku lze pomocí prvků *numericUpDown*. Byla vytvořena i funkce vytvoření nového znaku, ten vznikne vždy s parametry, které má v danou chvíli *panel1*, tedy s jeho umístěním a velikostí. Další funkcí je možnost spojení dvou znaků.



Obr. 2.4. Formulář pro úpravu znaku

Z obou zdrojových znaků jsou vzaty krajové rozměry, tedy ty co spolu tvoří obrys o nejvyšších rozměrech. Jsou-li zvoleny více než dva znaky, je poslední ignorován. Proto je slučování více než dvou znaků nutné dělat ve více krocích.

Kromě zmiňovaného objektu *panel1*, který ohraničuje vybraný znak, byl implementován též algoritmus pro ohraničování všech znaku zároveň. Pro vykreslování byla využita událost *Paint* objektu *pictureBox1*. Tato událost se stará o veškeré překreslování tohoto prvku. Algoritmy v ní obsažené, se volají při jakékoli změně velikosti, pozice nebo překrytí jiným prvkem. Pro samotné vykreslení ohraničení všech znaků byla využita funkce *DrawRectangles(pen, rects)*, která je součástí třídy *Graphics*. Je to funkce, která vykreslí větší množství obdélníků najednou. Jejím vstupním parametrem je pole prvků datového typu *Rectangle*, a dále nástroj pro vykreslení, který určuje, jaký typ ohraničení obdélník bude mít. Tato proměnná musí být datového typu *System::Drawing::Pen*, který obsahuje informaci o barvě a tloušťce ohraničení.

Proto bylo nutné vytvořit pole objektů typu *Rectangle*. Dle nápovědy ve funkci *DrawRectangles(pen, rects)* bylo zvoleno *array<Rectangle>^ rects*. Tento datový typ by se dal považovat za jakési dynamické pole, protože lze v průběhu užívání měnit jeho velikost. Pole bylo vytvořeno jako globální proměnná a to proto, že je nutné k němu přistupovat z více než jedné metody. Při načítání dat do prvku *dataGridView1* je spočítán počet znaků, poté je pomocí metody *Resize(rects, pocet)* nastavena velikost. Po nastavení velikosti je pole naplněno. Vždy když jsou data v *dataGridView1* upravena, je aktualizováno i pole s obdélníky. Událost *Paint* poté automaticky překreslí ohraničení znaků.

Pro vykreslování byla zvolena barva *Color::Lime*, která je vidět i na obrazových datech horší kvality.

2.3. Prostředí *Training data*

Toto poslední prostředí slouží k zpracování dat vytvořených prostředím *Box creation*. Pro zpracování dat je nutné načíst jména souborů, k tomu slouží *openFileDialog1*. Dialogu byla nastavena vlastnost *Multiselect* na hodnotu *true*, to umožní uživateli vybírat více souborů naráz. Poté, co jsou soubory vybrány, jsou jejich celé názvy zapsány do prvku *listBox1*. Při tomto zapisování jsou odstraněny čtyři poslední znaky, tedy znaky určující typ souboru. To se využívá pro načítání soborů nejen typu **.box* ale i **.tif* a **.tr*.

2.3.1. Vytváření **.tr* souborů

Pro každý ze souborů zapsaných v prvku *listBox1* je vytvořen dávkový soubor pro vytvoření dat **.tr*, který je poté spuštěn. Na Obr. 2.5 je vidět algoritmus, který má za úkol vygenerovat obsah dávkového souboru. Při generování je načten jeden řádek z prvku *listBox1*, dále je převedena cesta z datového typu *String* na *std::string*[6], to zjednoduší práci s řetězci. Dále je již jen složen příkaz.

```
string temp1="";
String^ temp3 = listBox1->Items[i]->ToString();
for (int j=0;j<temp3->Length;j++){
    temp1+=temp3[j];
}
davka += "tesseract "+ temp1 +".tif "+ temp1 +" nobatch box.train\n";
temp1="";
```

Obr. 2.5. Algoritmus vytvoření volacího příkazu pro tvorbu souborů **.tr*

Příkaz je následně zapsán do souboru pomocí třídy *StreamWriter*, ta byla ještě opatřena odchytáváním výjimek pomocí *try-catch*, viz Obr. 2.6.

```
try{
    System::IO::StreamWriter^ sw1 = gcnew System::IO::StreamWriter(jmeno+"XF.bat");
    sw1->Write(davkaS);
    sw1->Close();
}catch(System::Exception ^e){
    MessageBox::Show("Nepodařilo se vytvořit.", "Chyba zápisu", MessageBoxButtons::OK,
                                                                MessageBoxIcon::Error);
}
```

Obr. 2.6. Ukládání souboru opatřené odchytáváním chyb *try-catch*

Soubor je poté spuštěn, viz Obr. 2.7, tím je vytvořen soubor **.tr* pro daná data, po vytvoření je zapsána informace o vytvoření do prvku *listBox2*. Celý proces byl znovu opatřen odchytáváním výjimek.

```
try{
    Process::Start( jmeno+"XF.bat" )->WaitForExit();
    listBox2->Items->Add("Soubory *.tr vytvořeny.");
}catch(System::Exception ^e){
    MessageBox::Show("Nepodařilo se spustit dávkový soubor.",
        "Chybné spuštění", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

Obr. 2.7. Spuštění dávkového souboru pomocí třídy *Process*

2.3.2. Vytváření zbývajících dat

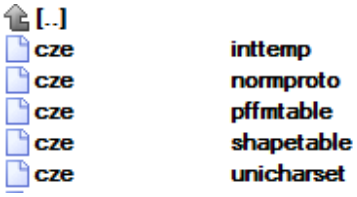
Stejným způsobem, jako v případě souborů typu **.tr*, jsou generovány příkazy pro tvoření dalších důležitých souborů. Rozdíl mezi těmito příkazy a příkazy předchozími je v nutnosti spojit všechna vstupní data do jednoho dávkového příkazu, tedy ne jako v předchozím případě, kde se pro každý vstupní soubor generovala zvlášť. Tyto dávky jsou generovány pomocí cyklu, který prochází *listBox1* načítá z něho adresy vstupních souborů a poté je zapisuje do řetězce, viz Obr. 2.8. Následný zápis do souboru a spuštění je stejné jako na Obr. 2.6 a Obr. 2.7.

```
string uni="unicharset_extractor ",cn="ncntraining ";
string sha="nshapeclustering -F "+jmenoSKS+"font_properties -U unicharset ";
string mf="nmftraining -F "+ jmenoSKS +"font_properties -U unicharset  -O "
                                          +jmenoSKS+jazykSKS+".unicharset ";
for (int i=0; i<listBox1->Items->Count;i++){
    for (int j=0;j<listBox1->Items[i]->ToString()->Length; j++){
        uni += listBox1->Items[i]->ToString()[j];
        sha += listBox1->Items[i]->ToString()[j];
        mf += listBox1->Items[i]->ToString()[j];
        cn += listBox1->Items[i]->ToString()[j];
    }
    uni += ".box "; sha += ".tr "; mf += ".tr "; cn += ".tr ";
}
uni = uni+"\n"+sha+"\n"+mf+"\n"+cn;
String^ cs = gcnew String(uni.c_str());
```

Obr. 2.8. Generování spouštěcích příkazů pro zbývající moduly

2.3.3. Kompilace dat do tréninkového souboru

Poslední část tohoto prostředí sloučí všechna data do jednoho souboru. Prvním krokem bylo přejmenovat veškeré soubory tak, aby bylo možné je využít. Všechny soubory, které mají být použity, musí mít specifický tvar. Každý ze souborů, který je třeba pro vytvoření tréninkových dat, musí mít tvar *xxx.**, tedy např. *cze.**, viz Obr. 2.9.



Obr. 2.9. Soubory potřebné pro spuštění tvoření tréninkových dat

Toto přejmenovávání bylo vytvořeno pomocí funkce, obsažené v hlavičkovém souboru *stdio.h, rename(char* puvodni, char* nove)*. Pro převod mezi datovými typy *std::string* a *char** byla posloužila metoda *c_str()*, jenž je součástí metod datového typu *std:string*. Protože se jedná o metodu pracující se soubory, tak ji bylo třeba z možných problémů se zápisem ošetřit odchytáváním výjimek pomocí *try-catch*. V takovém případě bylo nutné dát tuto skutečnost uživateli najevo. K tomu se využilo prvku *MessageBox*, ten vyvolá tabulka, u které je možnost volby obsahu.

V našem případě bylo navoleno tlačítko *OK*, ikona chybového hlášení a samotný text, který předá uživateli zprávu, viz Obr. 2.10.

```
try{
    temp6 = soubor + ".shapetable";
    rename("shapetable", temp6.c_str());
}catch(System::Exception ^e){
    MessageBox::Show("Nepodařilo se přejmenovat soubor shapetable.",
                "Chybna", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

Obr. 2.10. Ukázka použití metody *rename()* pro změnu jména souboru

Tímto se vytvořilo přejmenovávání pro čtyři soubory, soubory *inttemp*, *normproto*, *pffmtable* a *shapetable*. Po té, co jsou přejmenovány soubory nutné k vytvoření tréninkového souboru, je spuštěn proces vytváření. To se stane pomocí příkazu *combine_tessdata xxx.* tedy v našem případě *combine_tessdata cze.* Znaky *cze.* určují soubory, které budou vybrány pro tvoření dat.

3. Testování prostředí

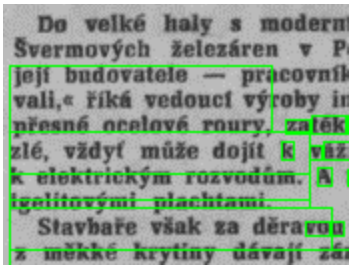
3.1. TIFF a řešení v aplikaci

TIFF je souborový formát pro ukládání rastrové počítačové grafiky. Slouží k ukládání skenovaných dat, nebo dat, jenž jsou určena pro tisk. Formát též umožňuje vícestránkové ukládání dat, proto se často tento formát užívá pro ukládání faxů. Formát má velmi volně specifikovanou strukturu, proto velmi často dochází k nekompatibilitě mezi jednotlivými soubory. Specifikace TIFF dává volnou ruku v tvorbě nových *tagů* a voleb v obrazovém souboru. Tím vzniká problém s podporou, není v možnostech programu, znát veškeré vytvořené verze datového typu.

Stejný problém nastal i v případě obrazových dat užívaných v Tesseract OCR. V aplikaci bylo vytvořeno převádění obrazových datových typů do souboru TIFF pomocí metody *save(jméno,typ_souboru)*, která je součástí třídy *Bitmap*. Parametr *typ_souboru* byl nahrazen formátem *System::Drawing::Imaging::ImageFormat::Tiff*. Takto vytvořené soubory TIFF byly nekompatibilní s aplikací Tesseract OCR, proto došlo k odstranění tohoto algoritmu a bylo ponecháno formátování dat na uživateli a externím grafickém programu.

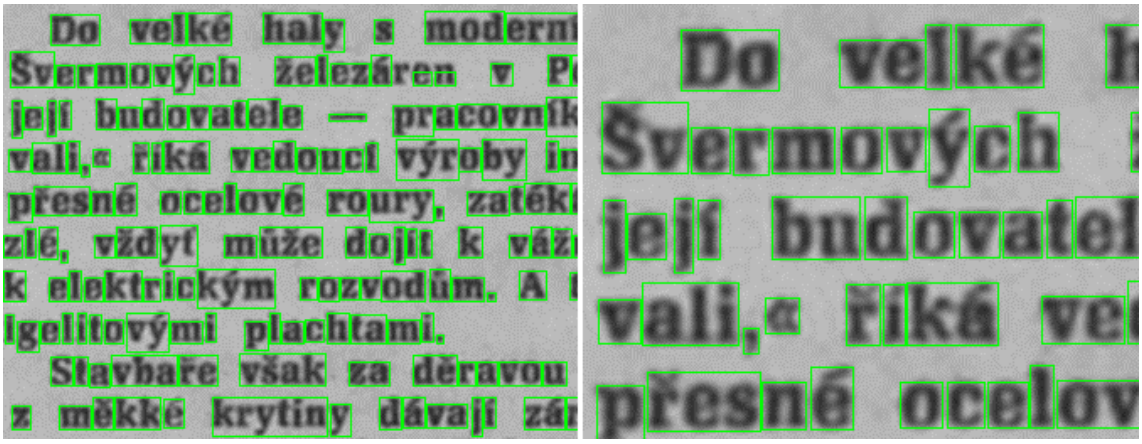
3.2. Kvalita obrazových dat

Pro testování aplikace byla použita skenovaná data deníku Rudé Právo. Každý skenovaný dokument má rozlišení 1800x2401px. Pro vytváření testovacích dat bylo nutné stránky rozstříhat na odstavce a ty poté zvětšit. Bez této úpravy docházelo ke špatnému rozpoznávání znaků, viz Obr. 3.1, protože znaky byly příliš malé (cca 6:7px). Ve chvíli, kdy byli znaky zvětšeny na dvojnásobnou velikost, se prvotní rozpoznávání znaků zlepšilo a již byly jednotlivé znaky rozpoznány.



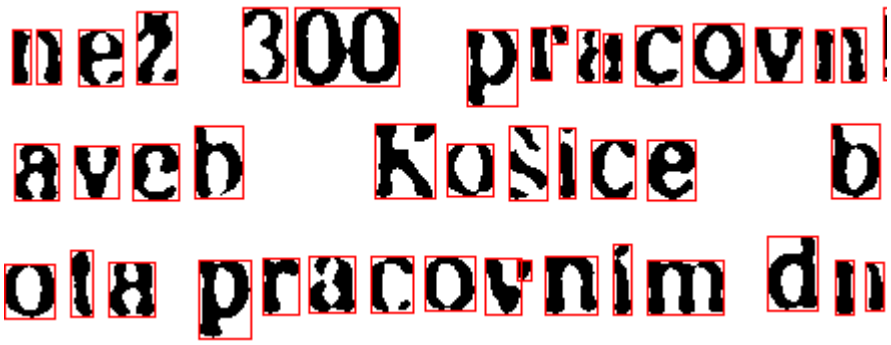
Obr. 3.1. Chybné rozeznávání příliš malých znaků

Předepsaná minimální velikost znaku je 15px na výšku, při nižším rozlišení pak dochází k chybnému rozpoznávání hranic znaku, jako tomu bylo v našem případě. Při pokusu, zdali nedojde k dalšímu zlepšení při čtyřnásobném zvětšení, se již provedené změny nikterak na zlepšení rozpoznávání neprojevily, viz Obr. 3.2, pouze došlo ke zvětšení dat a zpomalení procesu. S ohledem na toto zjištění bylo usouzeno, že výška znaku 15px je dostatečná a není nutno jí měnit.



Obr. 3.2. Porovnání zvětšených dat, vlevo dvojnásobné, vpravo čtyřnásobné

Při vytváření dat s informacemi o pozicích znaku docházelo k záměnám znaků *t* a *i*, dále *i* a *l* popř. *1* a dalších několika dvojic. Tyto chyby byly přisouzeny typu písma a nekvalitnímu rozlišení obrazů.

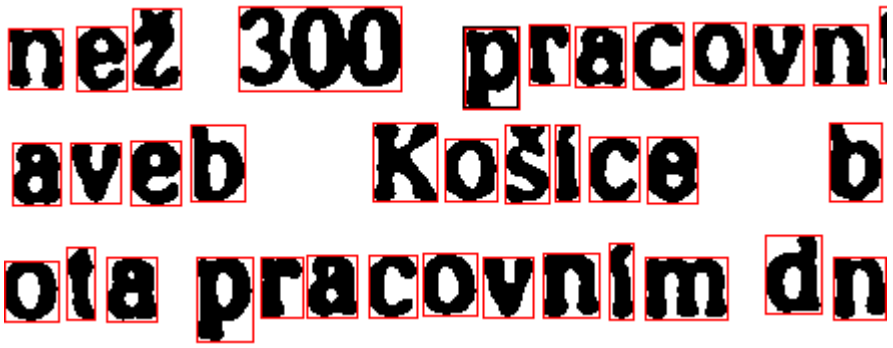


Obr. 3.3. Rozpoznaný obraz po binární konverzi

Dalším pokusem byla manipulace s obrazem před použitím Tessearct OCR. Obraz byl převeden pomocí binární konverze, ta vytvořila černo-bílý obrazec. Při testování byl

však výsledek mnohem horší než s originálními daty. Při konverzi totiž došlo k zaokrouhlení barev a vznikly v některých částech znaků mezery. Ty Tesseract OCR posoudil jako mezery mezi znaky, to je například vidět v Obr. 3.3.

S ohledem na toto zjištění pokračovaly pokusy o zlepšení pomocí úpravy. Další pokus začínal znova na původním obraze. Nejprve byla provedena konverze na osm odstínů šedí, a teprve poté byla provedena konverze binární. Výsledný obrazec projevil velmi dobré vlastnosti pro rozpoznávání. Byly odstraněny nečistoty, které neměly nic společného s textem, znaky byly lépe rozeznatelné než v originálním obraze, viz Obr. 3.4.

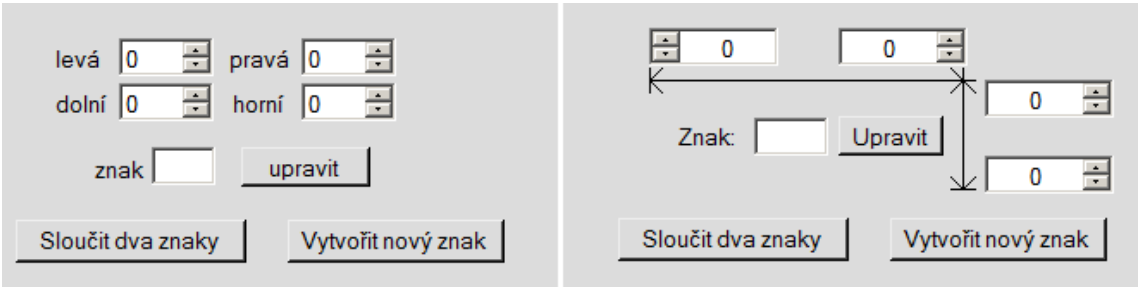


Obr. 3.4. Rozpoznaný text po konverzi do osmi stupňů šedi a binární konverzi

3.3. Úpravy prostředí pro lepší ovládání

3.3.1. Úprava formuláře znaku

Při testování prostředí bylo přistoupeno k několika změnám. První, a asi největší změnou byla změna rozestavení prvků formuláře pro úpravu vlastnosti znaku v prostředí *Box creation*.



Obr. 3.5. Změna formuláře pro úpravu zanku

Ten byl upraven tak, aby lépe naznačoval opravdovou pozici upravované linie znaku. V prvotním návrhu byly prvky *numericUpDown* pouze pojmenovány podle hodnot, jenž představovaly. V nově navrženém rozmístění je místo slovního pojmenování grafické znázornění upravované vlastnosti. Rozdíly je možné vidět na Obr. 3.5.

3.3.2. Výběr znaku

Režim prvku *dataGridView1* byl upraven tak, aby byl vždy vybrán celý řádek. Toho bylo docíleno nastavením vlastnosti *SelectionMode* na hodnotu *FullRowSelect*.

Další přidanou funkcí, již zmiňovanou na straně **Chyba! Záložka není definována.**, byl výběr znaku kliknutím do obrazu. Tato funkce byla přidána jako jedna z posledních. O samotné hledání znaku v tabulce se stará algoritmus, který je znázorněn na Obr. 2.3. Po vytvoření této funkce, bylo nutné docílit toho, aby byl vybraný znak v tabulce vidět a uživatel nemusel ručně listovat tabulkou a hledat vybraný řádek. K docílení došlo pomocí přidání příkazu *dataGridView1->FirstDisplayedScrollingRowIndex = index;* do algoritmu hledání. Tento příkaz má za úkol, nastavit posuvník tabulky tak, aby vybraný index nastavit jako první v tabulce.

3.4. Testování tvorby tréninkových dat

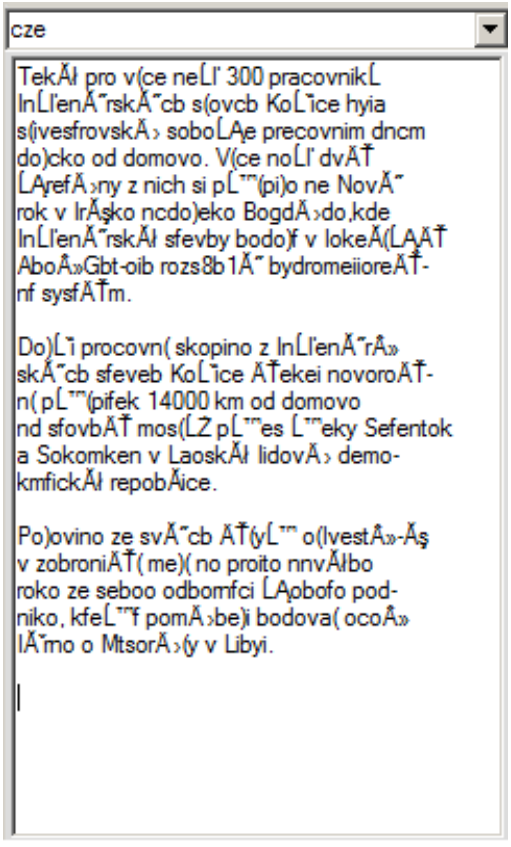
Bylo sestříháno 20 odstavců textu. Celkem 13.231 znaků zapsaných ve 2499 slovech. Při vytváření dat došlo k chybám, což znamená, že znaky byly odstraněny ze zdrojových dat. Takovýchto chyb se vyskytlo 12. Při následném vytváření seznamu znaků, vznikl z tohoto vzorku dat *unicharset* soubor, který obsahoval pouhých 86 znaků. Při porovnání se znakovou sadou již vytvořených tréninkových dat českého jazyka, které je možno stáhnout na stránkách věnovaných projektu[4], který obsahuje 172 znaků, je jasné, že znaková sada, která byla vytvořená, není dostatečná. To bylo s nejvyšší pravděpodobností způsobeno nedostatečným množstvím vstupních dat. Dalším důvodem pak může být skutečnost, že určitá část znakové sady není běžně využívána v textech. To se týká některých písmen např. ‘*q*’ nebo ‘*w*’ nebo ‘*x*’ a také větší části znaků, nepatřících mezi ty alfanumerické např. ‘<’ nebo ‘*}*’.

Dalším problémem byla četnost jednotlivých znaků, konkrétně znaků, které jsou pro jazyk specifické. U těchto znaků je dosti problematické docílit požadovaného počtu kvalitních vzorků. Tvůrci aplikace Tesseract OCR uvádí na stránkách projektu[5], ideální počet vzorku je cca 20. Pro získání takovéhoto počtu vzorků některých znaků, např. čísel, je třeba velmi velké množství vstupních dat.

Tréninkový soubor, který byl vytvořen ze zmíněných dat, měl velikost přibližně 380kB. Při porovnání s jinými, již hotovými daty, byl asi 10 krát menší. Již zmiňovaný tréninkový soubor českého jazyka je velký cca 2.500kB. Pro další příklad soubor, který Tesseract OCR používá jako primární jazykovou sadu má velikost cca 21.300kB.

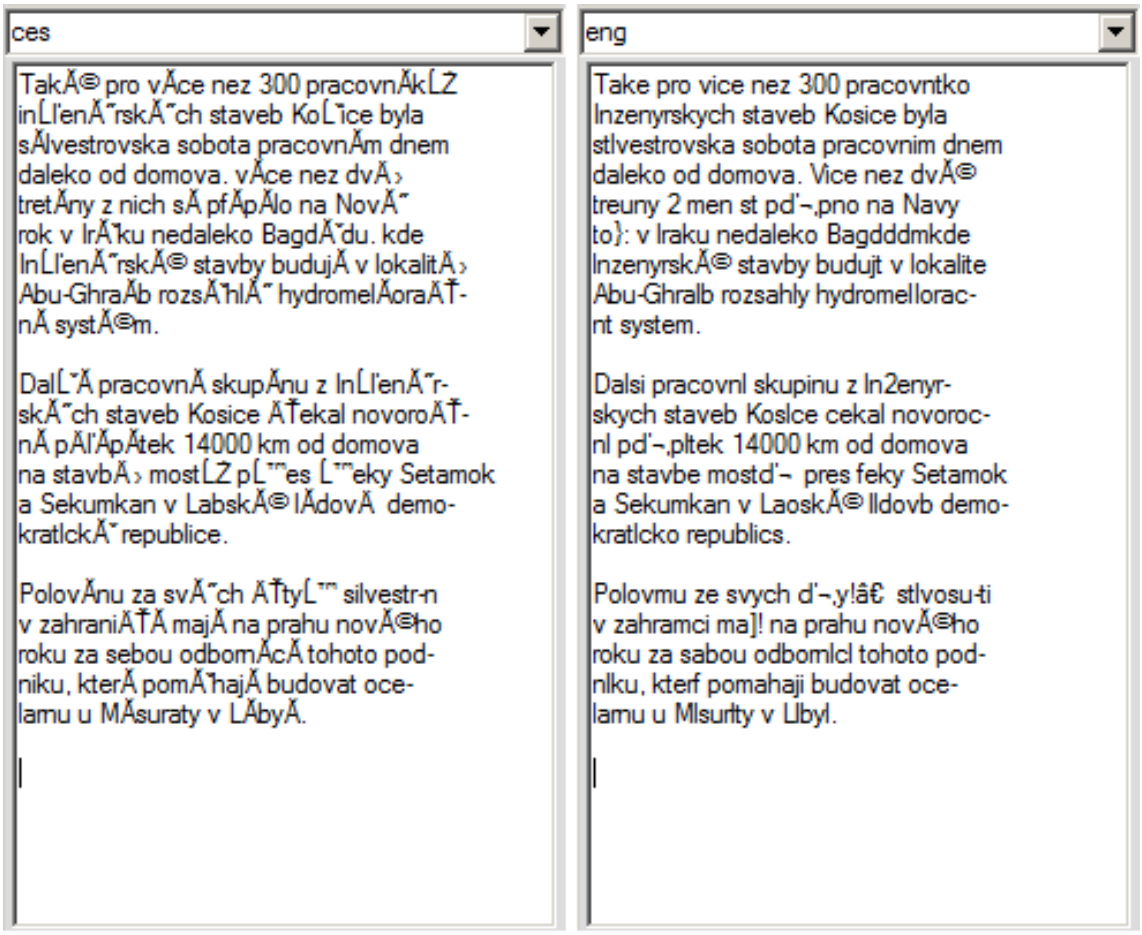
Z těchto dat bylo usouzeno, že pro vytvoření kvalitních tréninkových dat, jsou potřebná mnohem obsáhlejší zdrojová data. Vzhledem ke skutečnosti, že aplikace Tesseract OCR při zpracovávání zdrojových dat dovoluje zpracovat pouze 32 obrazových souborů, bylo by nutno některé texty sloučit tak, aby se snížil počet.

Na Obr. 3.6 je výsledný rozpoznaný text, který vznikl za použití vytvořeného tréninkového souboru. Je patrné, že znaky jako ‚ž‘ nebo ‚č‘ nejsou rozpoznány.



Obr. 3.6. Text rozpoznaný vytvořeným souborem

Na Obr. 3.7 je znázorněn výsledek rozpoznávání za použití stažených znakových sad. Jak je patrné, tak ani znaková sada, která je veřejně přístupná na internetových stránkách, nebyla schopná rozpoznat obrazová data. Z toho lze usoudit, že data nejsou příliš kvalitní. Nejlepších výsledků dosáhl tréninkový soubor anglického jazyka, který je defaultním pro práci aplikace Tesseract OCR, a který má nejobsáhlejší tréninkový soubor ze všech zmiňovaných.



Obr. 3.7. Výsledek rozpoznání při použití stažené znakové sady ces (vlevo) a defaultní sady eng (vpravo)

Závěr

V rámci této bakalářské práce bylo vytvořeno grafické prostředí, které uživateli dovoluje ovládat moduly aplikace Tesseract OCR. Prostředí bylo rozděleno do tří částí.

Prostředí OCR bylo vytvořeno pro rozpoznávání textu v obrazových datech. K tomu leze využívat volitelné znakové sady, které jsou načítány jako vstupní soubor. Rozpoznaný text lze editovat a uložit.

Další vytvořenou části je okno, sloužící jako editor souborů s informacemi o pozicích znaků v obraze. Pomocí formuláře lze data upravovat. Úpravy jsou v reálném čase zobrazovány na obrázku. To uživateli umožní efektivně opravovat daná data. Byla přidána funkce sloučení znaků a také odebrání těch nepotřebných. Další důležitou přidanou funkcí je možnost vytvoření nového znaku. To vše dá uživateli možnost upravovat automaticky vygenerovaný soubor.

Tato data jsou pak používána v poslední části prostředí. Ta slučuje upravené soubory s informacemi o pozicích znaků a vytváří z nich soubor, použitelný jako znakovou sadu pro rozpoznávání textu v obrazových datech.

Aplikace byla nakonec testována a upravována tak, aby lépe plnila svou funkci. Bylo opraveno několik funkčních chyb a dále některé designové prvky.

Při použití kvalitních dat má Tesseract OCR velmi dobré výsledky. Testování aplikace na vzorku dat ukázalo, že pro tvorbu tréninkových dat je třeba velmi velkého vorku dat, ve kterém jsou obsaženy všechny znaky užívané v daném jazyce. Bylo by tedy třeba vytvářet tréninkový soubor spíše z technické zprávy, kde je větší pravděpodobnost výskytu všech znaků v hojném počtu.

Literatura

[1] R. W. Smith, *The Extraction and Recognition of Text from Multimedia Document Images,* PhD Thesis, University of Bristol, November 1987.

[2] S.V. Rice, F.R. Jenkins, T.A. Nartker, *The Fourth Annual Test of OCR Accuracy, Technical Report 95-03,* Information Science Research Institute, University of Nevada, Las Vegas, July 1995

[3] R. Smith, *An Overview of the Tesseract OCR Engine,* [online], [cit. 2013-04-12], URL:< http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4376991>

[4] tesseract-ocr: *An OCR Engine that was developed at HP Labs between 1985 and 1995... and now at Google.* [online], [cit. 2013-4-21], URL:<https://code.google.com/p/tesseract-ocr/downloads/detail?name=tesseract-ocr-3.02.ces.tar.gz&can=2&q=>

[5] tesseract-ocr: *An OCR Engine that was developed at HP Labs between 1985 and 1995... and now at Google.* [online], [cit. 2013-4-22], URL:<https://code.google.com/p/tesseract-ocr/wiki/TrainingTesseract3>

[6] cplusplus.com: *string - C++Reference,* [onlie], [2013-4-26], URL:<http://cplusplus.com/reference/string/string/>

Příloha A

```
UnknownFont L 38 771 52 791 0
 4
mf 9
 0.046875 0.226786 0.210867 0.665954 0 0
 -0.084375 0.332143 0.369855 0.987693 0 0
 -0.24375 0.0428571 0.609198 0.263077 0 0
 -0.075 -0.248214 0.288585 0.513805 0 0
 0.021875 -0.141071 0.0827665 0.868928 0 0
 -0.00625 0.0125 0.246429 0.75 0 0
 0.39375 -0.126786 0.161506 0.800095 0 0
 0.209375 -0.110714 0.341096 0.057929 0 0
 0.24375 -0.219643 0.351473 0.514575 0 0
cn 1
 0.269531 0.225 0.21875 0.125
if 53
 61 209 75
 64 197 75
 68 186 75
 71 174 75
 73 162 64
 73 148 64
 73 134 64
 73 120 64
 73 106 64
 73 92 64
 73 80 64
 73 67 64
 79 61 132
 103 63 132
 115 65 132
 127 66 132
 139 67 132
 136 87 222
 129 95 222
 126 105 192
 126 117 192
 126 130 192
 200 73 131
 213 74 131
 225 75 131
 230 82 204
 226 95 204
 222 108 204
 214 112 15
 202 108 15
 191 103 15
 180 99 15
 168 95 15
 157 90 15
 145 86 15
tb 1
 60 230 110
```

Příloha B

```
86
NULL 0 NULL 0
L  5 0,255,0,255,0,32767,0,32767,0,32767      NULL   8 0 0   # L [4c ]A
o  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 72 0 0   # o [6f ]a
ň  3 0,255,0,255,0,32767,0,32767,0,32767      NULL -1 0 0   # ň [148 ]a
s  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 52 0 0   # s [73 ]a
k  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 57 0 0   # k [6b ]a
ý  3 0,255,0,255,0,32767,0,32767,0,32767      NULL -1 0 0   # ý [fd ]a
p  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 63 0 0   # p [70 ]a
l  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 1 0 0    # l [6c ]a
á  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 81 0 0   # á [e1 ]a
n  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 59 0 0   # n [6e ]a
i  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 37 0 0   # i [69 ]a
ř  3 0,255,0,255,0,32767,0,32767,0,32767      NULL -1 0 0   # ř [159 ]a
č  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 51 0 0    # č [10d ]a
a  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 33 0 0    # a [61 ]a
ě  3 0,255,0,255,0,32767,0,32767,0,32767      NULL -1 0 0   # ě [11b ]a
v  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 58 0 0   # v [76 ]a
y  3 0,255,0,255,0,32767,0,32767,0,32767      NULL -1 0 0   # y [79 ]a
t  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 54 0 0   # t [74 ]a
m  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 64 0 0   # m [6d ]a
ž  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 67 0 0   # ž [17e ]a
í  3 0,255,0,255,0,32767,0,32767,0,32767      NULL -1 0 0   # í [ed ]a
d  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 62 0 0   # d [64 ]a
c  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 79 0 0   # c [63 ]a
h  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 47 0 0   # h [68 ]a
r  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 53 0 0   # r [72 ]a
f  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 83 0 0    # f [66 ]a
é  3 0,255,0,255,0,32767,0,32767,0,32767      NULL -1 0 0   # é [e9 ]a
e  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 71 0 0    # e [65 ]a
-  10 0,255,0,255,0,32767,0,32767,0,32767     NULL 29 0 0    # - [2d ]p
ů  3 0,255,0,255,0,32767,0,32767,0,32767      NULL -1 0 0   # ů [16f ]a
.  10 0,255,0,255,0,32767,0,32767,0,32767     NULL 31 0 0    # . [2e ]p
»  10 0,255,0,255,0,32767,0,32767,0,32767     NULL 32 0 0   # » [bb ]p
A  5 0,255,0,255,0,32767,0,32767,0,32767      NULL 14 0 0   # A [41 ]A
ú  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 82 0 0   # ú [fa ]a
«  10 0,255,0,255,0,32767,0,32767,0,32767     NULL 35 0 0    # « [ab ]p
j  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 41 0 0   # j [6a ]a
I  5 0,255,0,255,0,32767,0,32767,0,32767      NULL 11 0 0   # I [49 ]A
u  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 84 0 0   # u [75 ]a
b  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 60 0 0   # b [62 ]a
g  3 0,255,0,255,0,32767,0,32767,0,32767      NULL 61 0 0   # g [67 ]a
J  5 0,255,0,255,0,32767,0,32767,0,32767      NULL 36 0 0   # J [4a ]A
Z  5 0,255,0,255,0,32767,0,32767,0,32767      NULL 48 0 0   # Z [5a ]A
,  10 0,255,0,255,0,32767,0,32767,0,32767     NULL 43 0 0    # , [2c ]p
1  8 0,255,0,255,0,32767,0,32767,0,32767      NULL 44 0 0   # 1 [31 ]0
9  8 0,255,0,255,0,32767,0,32767,0,32767      NULL 45 0 0   # 9 [39 ]0
4  8 0,255,0,255,0,32767,0,32767,0,32767      NULL 46 0 0   # 4 [34 ]0
```

Obsah přiloženého CD

- Text bakalářské práce ve formátu PDF
- Aplikace Tesseract OCR 3.01
- Vytvořený projekt OCR
- Vzorek dat vytvořený při testování