VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

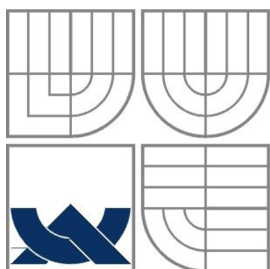
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FILTROVÁNÍ TEXTŮ EXTRAHOVANÝCH Z PDF, OCR NEBO WEBU

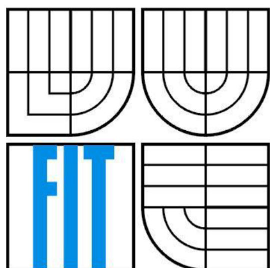
BAKALÁŘSKÁ PRÁCE
BACHELOR‘S THESIS

AUTOR PRÁCE FILIP LEHNERT
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMEDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FILTROVÁNÍ TEXTŮ EXTRAHOVANÝCH Z PDF, OCR NEBO WEBU

FILTERING OF TEXTS EXTRACTED FROM PDF, OCR OR WEB

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — FILIP LEHNERT
AUTHOR

VEDOUCÍ PRÁCE — Ing. IGOR SZÖKE, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Předmětem této práce je pomocí sadou skriptů zdokonalit převod různých typů dokumentů do čistě textové podoby. Převodem různých nástrojů dochází ke vzniku šumu a ne zcela korektním převodem znaků. Tyto skripty extrahovaný textový soubor vyčistí tak, aby výsledný text byl čitelný, dával smysl a neobsahoval zbytky různě vyskytujících se znaků z převodu grafů, tabulek, vzorců apod. Skript pracuje univerzálně a nevyžaduje vstup vzniklý pouze z nástrojů OCR nebo převodu z formátu PDF či webu.

Abstract

The objective of this thesis is to implement a set of scripts to improve the transfer of various types of documents into fully text. There appears noise and not entirely correct character conversion by converting various file formats. These scripts extracted text file cleans so that the resulting text is readable, make sense and does not contain any residues of various characters appearing by the transfer of graphs, tables, formulas, etc. The script works universally and does not require input solely by OCR tools or converting from PDF or web.

Klíčová slova

filtrovat, extrahovat, PDF, OCR, čistý text, slovník, gramatická pravidla, skripty, filtrování dat, převod dat

Keywords

filter, extract, PDF, OCR, plain text, dictionary, grammar rules, scripts, filtering data, data conversion

Citace

Filip Lehnert: Filtrování extrahovaných textů z PDF, OCR nebo webu, bakalářská práce, Brno, FIT VUT v Brně, 2013

Filtrování extrahovaných textů z PDF, OCR nebo webu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Igora Szöke, Ph.D. Další informace jsem čerpal z odborné literatury a internetových publikací.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Filip Lehnert
15.5.2013

Poděkování

Chtěl bych poděkovat Ing. Igoru Szöke, Ph.D., vedoucímu mé bakalářské práce za čas, úsilí a popř. i nervy, které mi a mé práci obětoval. Dále zkušenějším spolužákům za podporu a možnosti získat nové dovednosti a také lidem, kteří se teprve chystají tuto práci přečíst a případně ji poté i ocení. Nakonec také samozřejmě rodičům, kteří mě vedli tímto směrem a z větší části finančně podporovali.

© Filip Lehnert, 2013
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

Obsah ... 1
1 Úvod ... 2
  1.1 Extrakce textu ... 3
  1.2 Vybrané problémové příklady extrakce ... 3
2 Principy řešení ... 6
  2.1 Vyřazení šumu ... 6
  2.2 Zkratky ... 7
  2.3 Převod čísel ... 8
  2.4 Oblasti ... 8
  2.5 Požadavky pro spouštění ... 11
  2.6 Ostatní ... 11
3 OCR proces ... 12
  3.1 Implementace OCR filtru ... 14
  3.2 Vliv DPI na úspěch rozpoznání ... 17
  3.3 Nerozpoznání rozestupu mezi slovy ... 20
  3.4 FineReader ... 20
  3.5 OCR OpenSource ... 21
4 Návrh ... 22
  4.1 Vstup do filtru ... 22
  4.2 Základní filtr ... 23
  4.3 Skripty, tvorba pravidel a slovníků ... 23
  4.4 Konečný filtr ... 23
  4.5 Uživatelská konfigurace ... 24
  4.6 Schéma návrhu ... 25
5 Experimentování ... 26
  5.1 Srovnání s předchozím řešením ... 26
  5.2 Srovnání s referenčním řešením ... 26
  5.3 Testy ... 27
6 Závěr ... 29
Literatura ... 30
Seznam příloh ... 31
Plakát ... 32
Obsah CD ... 33

1 Úvod

V této práci je řešena implementace filtru extrahovaných textů. Sada skriptů předpokládá vstup znečištěného běžného textového souboru primárně exportovaného z převodníků PDF formátu či výstupu OCR programu. Skripty textové dokumenty následně přefiltrují tak, aby se daly využít např. jako vstup do adaptivních rozpoznávačů textů nebo zkrátka pro účel získání čitelného textu složeného z vět dávajících smysl, a to bez chyb, překlepů, nežádoucích zbytkových znaků vzniklých extrahováním z grafů, tabulek, plovoucích obrázků, vzorečků a popř. i jiným zašuměným obsahem. Filtr pracuje s textem v různých světových jazycích, avšak kladen důraz je především na angličtinu.

V první kapitole budete seznámeni s obecnými problémy extrahovaného textu. Díky názorně vloženým ukázkám si uděláte potřebné povědomí o tom, co samotná práce vlastně řeší a k čemu je vhodné vyhotovený software použít.

Ve druhé kapitole se dočtete o vytvořených programovacích principech a o technologiích včetně nástrojů, které sada skriptů vyžaduje ke správnému spouštění.

Třetí kapitola Vás seznámí s technikami rozpoznávání znaků z obrazu OCR (Optical Character Recognition) a popisem technik, jak nedokonalosti právě OCR tato práce odstraňuje.

Ve čtvrté kapitole je popsaný a vyobrazený návrh, návaznost skriptů na sebe a způsob manuální kontroly a zásahu do pravidel, slovníků a konfigurace aplikace uživatelem.

Předposlední pátá kapitola ukazuje výsledky průběhu experimentálního testování. Objektivní zhodnocení úspěchu celého projektu se dá znázornit srovnáním s referenčním ručně vyhotoveným řešením. Stoprocentní shody bychom mohli dosáhnout tak, že bychom naučili náš program porozumět textu, resp. naučili jej uvažovat jako my, což je, z pohledu dnešní doby, naprosto nepředstavitelné, a to jak z hlediska výpočetního výkonu pro takovýto program, tak z hlediska vytvoření takovéhoto programu.

Poslední šestá kapitola je shrnutím celého projektu. Rozepsal jsem zde také pár myšlenek, jakým směrem by mělo případné vylepšení mé práce pokračovat.

Pro odlišení běžného textu této technické práce od názvů proměnných, funkcí, spouštěcích parametrů či přepínačů a regulárních výrazů je použit odlišný tzv. strojový styl písma.

1.1 Extrakce textu

Extrahování textu neboli získávání informací především z odborné literatury v elektronické podobě je v současnosti zcela běžná záležitost. Na síti je nepřeberné množství software, který umí vyčíst informace z libovolných formátů. Takováto extrakce je však komplikovanější proces. Konvertory různých druhů textových dokumentů pracují do určité míry spolehlivě, ale i přesto vzniká nechtěný šum např. v důsledku nedostatečně přesného rozeznání znaků OCR programem. Naskenovaný obrázek máme k dispozici v nízkém rozlišení bodů na palec (dots per inch - DPI) nebo text je napsán malým, třeba i ručně psaným písmem. Extrakcí dochází také k rozházení struktury textu nebo podivné rozmístění znaků. Takový formát písma je pak složité přesně detekovat a přečíst tak, aby výstup byl gramaticky správně a věta dávala smysl.

1.2 Vybrané problémové příklady extrakce

Na následující zkrácené ukázce výstupu extrakce z formátu PDF pomocí Unixového nástroje PDFTOTEXT s přepínačem `-nopgrbrk` si znázorníme, co je nevhodně vyhotoveno. Pro lepší orientaci jsou na obrázcích[1] screenshotů vstupů a výstupů řádky dodatečně očíslovány.

```
 1 Robust Vocabulary Independent Keyword Spotting
 2 with Graphical Models
 3 Martin W¨ llmer 1 , Florian Eyben 2 , Bj¨ rn Schuller 3 , Gerhard Rigoll 4
 4 o
 5 o
 6 Institute for Human-Machine Communication, Technische Universit¨ t M¨ nchen, 80333 M¨ nchen, Germany
 7 a u
 8 u
 9 1
10
11 woellmer@tum.de
12 2
13 eyben@tum.de
14 3
15 schuller@tum.de
16 4
17 rigoll@tum.de
18
19 Abstract—This paper introduces a novel graphical model
20 architecture for robust and vocabulary independent keyword
21 spotting which does not require the training of an explicit
22 garbage model. We show how a graphical model structure for
23 phoneme recognition can be extended to a keyword spotter that
24 is robust with respect to phoneme recognition errors. We use a
25 hidden garbage variable together with the concept of switching
26 parents to model keywords as well as arbitrary speech. This
27 implies that keywords can be added to the vocabulary without
28 having to re-train the model. Thereby the design of our model
29 architecture is optimised to reliably detect keywords rather than
30 to decode keyword phoneme sequences as arbitrary speech,
31 while offering a parameter to adjust the operating point on the
32 Receiver Operating Characteristics curve. Experiments on the
33 TIMIT corpus reveal that our graphical model outperforms a
34 comparable Hidden Markov Model based keyword spotter that
35 uses conventional garbage modelling.
```

Obrázek 1.1: Výstup po převodu z PDF.

[1] Originální materiály ve formátu PDF obrázků 1.1 až 1.4 mi byly interně dodány.

Na řádku číslo 3 můžeme vidět neschopnost převodu přehláskovaných samohlásek. Samotná samohláska se nachází o jeden nebo více řádků níže. Stejný nedostatek je patrný i na 6. řádku. Podobná situace nastává i v textu či vzorečcích s horními i dolními indexy. Všechna vyznačená čísla jsou indexy jmen autorů dokumentu odkazující na jejich příslušné e-mailové adresy. Odstavec níže je v bezchybné podobě, ale celkové odřádkování a rozmístění dokumentu se nám ovšem nelíbí. Pro srovnání originální část dokumentu vypadá takto:

Robust Vocabulary Independent Keyword Spotting with Graphical Models

Martin Wöllmer [1], Florian Eyben [2], Björn Schuller [3], Gerhard Rigoll [4]

Institute for Human-Machine Communication, Technische Universität München, 80333 München, Germany

[1] woellmer@tum.de
[2] eyben@tum.de
[3] schuller@tum.de
[4] rigoll@tum.de

***Abstract*—This paper introduces a novel graphical model architecture for robust and vocabulary independent keyword spotting which does not require the training of an explicit garbage model. We show how a graphical model structure for phoneme recognition can be extended to a keyword spotter that is robust with respect to phoneme recognition errors. We use a hidden garbage variable together with the concept of switching parents to model keywords as well as arbitrary speech. This implies that keywords can be added to the vocabulary without having to re-train the model. Thereby the design of our model architecture is optimised to reliably detect keywords rather than to decode keyword phoneme sequences as arbitrary speech, while offering a parameter to adjust the operating point on the Receiver Operating Characteristics curve. Experiments on the TIMIT corpus reveal that our graphical model outperforms a comparable Hidden Markov Model based keyword spotter that uses conventional garbage modelling.**

I. Introduction

As an important discipline in the field of automatic speech recognition (ASR), keyword spotting has found many applications in recent years. For voice command detection, information retrieval systems, or embodied conversational agents, reliably detecting important keywords is often more important than attempting to capture the whole spoken content of an utterance. Hidden Markov Model (HMM) based keyword spotting systems [1], [2] usually require keyword HMMs and a *filler* or *garbage* HMM to model both, keywords and non-keyword parts of the speech sequence. Using whole word HMMs for the keywords and the garbage model presumes that there are enough occurrences of the keywords in the training corpus and suffers from low flexibility since new less flexible than vocabulary independent systems [4] where no information about the set of keywords is required while training the models.

Apart from the numerous HMM based approaches, more unconventional keyword spotting strategies, such as applying recurrent neural networks [5] or using discriminative learning procedures [6], [7] have been developed. The latter technique non-linearly maps speech features into an abstract vector space which has shown good performance, but requires much more computational power than HMM based methods.

In this paper we present a new graphical model (GM) design which can be used for robust keyword spotting and overcomes most of the drawbacks of other approaches. Graphical models offer a flexible statistical framework that is increasingly applied for speech recognition tasks [8], [9] since it allows for conceptual deviations from the conventional HMM architecture (as in [10] or [11] for example). The GM makes use of the graph theory in order to describe the time evolution of speech as a statistical process and thereby defines conditional independence properties of the observed and hidden variables that are involved in the process of speech decoding. Apart from common HMM approaches, there exist only a small number of works which try to address the task of keyword spotting using the graphical model paradigm. In [12] a graphical model is applied for spoken keyword spotting based on performing a joint alignment between the phone lattices generated from a spoken query and a long stored utterance. This concept, however, is optimised for offline phone lattice generation and bears no similarity to the technique proposed herein.

Obrázek 1.2: Obraz původního dokumentu.

Nesrovnalosti se vyskytují i např. u vzorců. Takové pozůstatky je třeba správně identifikovat a vymazat z textového dokumentu stejně tak jako citované zdroje, e-mailové adresy, popisky obrázků, legendy grafů, hodnoty tabulek apod.

First, let X be a discrete random variable counting degraded pixels in a square window B of size $n = (2r+1)^d$ in the image I. Then, the probability that at least k pixels are corrupted in B is given by

$$\mathbb{P}(X > k) = \sum_{i=k+1}^{n} \binom{n}{i} \xi^i (1-\xi)^{n-i}, \tag{14}$$

where $\binom{n}{i} = \frac{n!}{i!(n-i)!}$. For a fixed window radius r, it is straightforward to see

Obrázek 1.3: Úryvek dokumentu se vzorci.

Z výše uvedeného screenshotu extrahováním vznikne následující nepořádek:

```
897 First, let X be a discrete random variable counting degraded pixels in a square
898 window B of size n = (2r + 1)d in the image I. Then, the probability that at
899 least k pixels are corrupted in B is given by
900 n
901
902 P(X > k) =
903 i=k+1
904
905 n i
906 ξ (1 − ξ)n−i ,
907 i
908
909 (14)
910
911 n!
912 where n = i!(n−i)! . For a fixed window radius r, it is straightforward to see
```

Obrázek 1.4: Nevhodný výstup po extrakci z PDF.

Vzorec představující jistou pravděpodobnost výskytu poškozených pixelů je nesmyslně rozmístěn na celkem osm řádků. Celá originální část obrazu dokumentu je přitom rozložena pouze na pět řádků. Řádky č. 900 až 907 z textu vyžadujeme vymazat včetně obou vzorečků na řádcích č. 898 a 912 tak, aby vznikl námi požadovaný souvislý text složený ze souvětí. Označení vzorce „(14)“ na řádku č. 909 ztrácí zcela smysl, jelikož příslušně očíslovaný vzoreček vyzmizíkujeme. Zbytek obsahu dokumentu je posléze třeba slepit k sobě (smazat odsazení), abychom nemáme zbytečné prázdné místo.

2 Principy řešení

V této kapitole jsou rozepsány jednotlivé implementované postupy. Menší a srozumitelnější celky jsou pouze zmíněny, aby se čtenář dozvěděl o všech funkcích programu. V poslední části píši o nárocích a nutných požadavcích programu ke spouštění.

2.1 Vyřazení šumu

Vyskytující se šum je nejprve nutné odlišit od rozumného textu, vyhodnotit jeho míru a smazat. K tomuto účelu je vyhotoven algoritmus, který vypočítá míru pravděpodobného šumu `rozhodovaci_uroven`, srovná ji s konstantou, která byla nastavena testováním tak, aby výsledky byly co nejblíže referenčnímu řešení. Míra `rozhodovaci_uroven` se počítá pro každou větu a oblast zvláště takto

$$rozhodovaci_uroven = \frac{sentence_length}{(\sum Ki * Ci) * 100}$$

Rovnice 2.1: Výpočet míry šumu v textu.

kde K značí příslušný koeficient množiny znaků a C četnost výskytů množiny znaků i.

Je tedy podílem délky věty a vypočítaným ohodnocením v procentech. Ohodnocení je číslo zjednodušeně řečeno udávající rozložení obsahu věty neboli míru váhového zastoupení různých typů množin znaků. Čili jedná se tedy o formu lingvistické neurčitosti, které se říká fuzzy logika. Tato proměnná počítá zastoupení symbolů, malých písmen, velkých písmen, názvů a zkratek. Tyto počty se násobí konstantami dle toho, jak často jsou běžně součástí šumu. Malá písmena a zkratky tedy mají nejnižší konstantu násobku, čísla a velká písmena jsou násobena vyšším koeficientem. Názvy lokací, osob, společností atd. a povolené symboly, které se mohou běžně vyskytovat v textu (uvozovky pro přímou řeč, dvojtečka před přímou řečí, kulaté závorky, spojovací pomlčka, tečka, čárka, otazník, vykřičník) násobíme ještě vyšší konstantou. Symboly, kterými rozumíme vše ostatní, jsou vynásobeny nejvyšším násobitelem, jelikož v textu určitě nevyžadujeme lomítka, znaky nerovnosti,

závorky (složené, hranaté…), mřížky, přebytečné bílé znaky[2] a další znaky s ASCII[3] hodnotou vyšší než 122 apod. Zásah uživatele do koeficientů je možný. Pokročilý uživatel může konfigurační soubor upravit tak, jak si myslí, že by program mohl odmazávat lépe. Výchozí nastavení je výsledkem rozsáhlejšího testování a mělo by pro většinu dokumentů, formulářů a textů vypočítat nejideálnější hodnotu vůči porovnávané hranici vyhození. Výsledný součet těchto hodnot udává ohodnocení, potřebné k určení rozhodující úrovně. Pokud v krátké větě je větší zastoupení nežádoucích znaků, tak `rozhodovaci_uroven` bude nabývat velmi malých hodnot, neprojde konstantně nastavenou mezí a tím pádem skript celou větu nebo oblast vyřadí. Konstantu lze upravit tak, aby lépe vyhovovala uživateli. Jinými slovy pokud se uživateli zdá, že skript vyhazuje příliš často a větší část textu, než by předpokládal, mez jednoduše v konfiguračním souboru zmenší nebo naopak posune nahoru.

2.2 Zkratky

Skript umí vyhledat a zpracovat zkratky víceslovných termínů, institucí, spolků, podniků a čehokoli jiného. Obvykle v článku, pokud je používána nějaká zkratka častěji, je u prvního výskytu rozepsán celý její slovní zápis. Lexikální analyzátor [8] narazí na zkratku (zpravidla řetězec s alespoň dvěma velkými písmeny po sobě) a začne prohledávat zpátky předchozí text z bufferu. Srovná shodu velkých písmen se začátky předchozích slov a přečte si tak celou zkratku. Tu si pamatuje a nahrazuje veškeré výskyty v textu celým souslovím.

Pro případ, kdy v dokumentu nemáme vysvětlenou zkratku, je implementován slovník, do kterého se všechny nálezy ukládají, aby se časem každým zpracovaným dokumentem program rozšiřoval a zdokonaloval sám sebe.

Obvykle se zkratky zapisují pouze velkými písmeny, ale někdy narazíme i na nějakou třeba neoficiální zkombinovanou včetně malých písmen někde uprostřed řetězce. Může se jednat např. o předložku „of“ (Age of Empires - AoE). Skript je chytrý a zkratky si všimne. Stejná situace platí i pro opačný případ, kdy rozvinutý slovní zápis se skládá včetně slov (předložek), ale které ve zkrácené podobě neobsahuje příslušné velké počáteční písmeno (Faculty of Information Technology - FIT).

Zřídka se může v dokumentu vyskytnout jedna tatáž zkratka pro dva nebo i vícero významů (sousloví). Skript si však pamatuje vždy pouze jednu prvně rozepsanou alternativu, podle které přepíše všechny výskyty zkráceného zápisu.

---

[2] Bílý znak (whitespace character) - v informatice je to znak, který představuje prázdné místo, čili svislou nebo vodorovnou mezeru. Jsou to znaky běžně zadávané např. mezerníkem nebo tabulátorem.
[3] ASCII (American Standard Code for Information Interchange) - je kódová tabulka definující anglické znaky a jiné znaky používané v informatice.

2.3 Převod čísel

Abychom dosáhli co největšího množství čistého textu, je vhodné veškeré číslovky převést na odpovídající řetězce znaků. Oříznutý text od vzorečků, grafů, tabulek, obrázků apod. bude tedy nakonec i rozšířen „zpátky". Čísla jsou převedena na slovní zápis pomocí funkce `int2word()` převzaté ze zdrojového Python kódu Number to Word Convertor [4]. Čísla se převádí na anglické číslovky.

Ukážeme si, jak proběhne správný převod na výroku vzrůstu mzdy ve střední vrstvě obyvatel USA[4]:

Median income households saw real earnings go from 40,800 USD in 1967. By 2010, that was up to 49,400 USD, just a 21 percent increase. => Median income households saw real earnings go from forty thousand eight hundred United States Dollars in nineteen sixty-seven. By two thousand and ten, that was up to forty nine thousand four hundred United States Dollars, just a twenty-one percent increase.

2.4 Oblasti

K některým úpravám používá program rozdělení textu na jednotlivé oblasti, které se snaží vyhodnocovat, upravovat, spojovat a vyhazovat. Výsledek funguje na podobný způsob jako software IGP:OCR-CG Latin pro OCR, kterým jednoduše v obrázku označíte vodorovnými přímkami hranice mezi článkem samotným a zápatím či záhlaví. Odříznete tím pádem dvě nebo více oblastí, u nichž nemáte zbytečné nutkání digitalizovat obrázek, čímž předejte možnému zbytečnému zanesení šumu do samotného článku.

Princip nalézání oblastí funguje tak, že celý textový soubor znaků extrahovaný nástrojem PDFTOTEXT s parametrem `-layout` se převede do maticového pole `ltext` se souřadnicemi přesně tak, jak jsou fyzicky znaky v dokumentu reálně rozmístěny. Parametrem `-raw` získáme soubor s uchovaným tokem textu. Dále je definované pomocné pole `larr`, do kterého se ukládá hodnota znaků na příslušných souřadnicích z pole `ltext`. Pro všechny alfanumerické znaky se nachází v poli `larr` hodnota 1, pro bílé znaky 0 a pro všechny ostatní 3. Vznikají tedy řetězce těchto čísel, přičemž celý soubor se rozděluje na dílčí posloupnosti dle konstant toleranční vzdálenosti ve

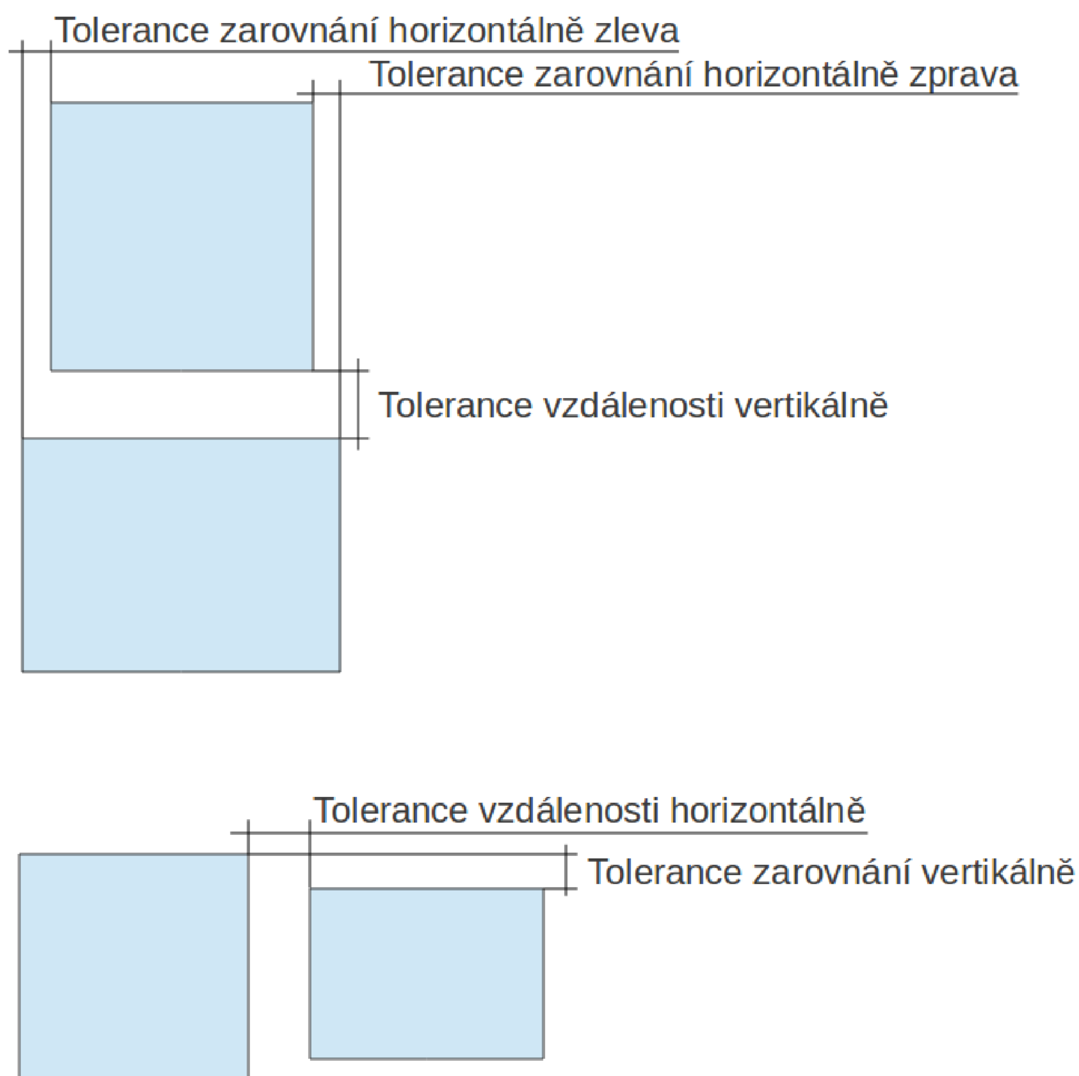
[4] Dostupné informace na URL: http://www.facethefactsusa.org/facts/income-gap-between-rich-middle-class-and-poor-widens/

vertikálním i horizontálním směru. Oblasti jsou také nalézány zarovnáním. Ukážeme si příklad korektního vyhledání oblastí na smyšleném textu se dvěma nadpisy a jedním odstavcem. Oblasti jsou vyznačeny modře. Zbytek textu jsou nuly představující mezery, tabulátory a nové řádky.

```
1  000000000000000111111111110
2  0
3  0
4  0
5  0
6  000000000000000111111111110110111111011011111011111011111101111110
7  0000000000000001110111111110111111130101111110111110
8  0000000000000001111010111111111110111110101111111111101111101011111110
9  0000000000000001111110111111111011111011111111 0
10 0000000000000001111111111011333111110111110111111011111101011111111 0
11 00000000000000011110111111101111110111111111111011111101011111111 10
12 0
13 000000000000000000000000000000010111011111110
```

Obrázek 2.1: Znázornění oblastí textu.

Vidíme, že na krátkém úseku textu se nachází tři pomyslné oblasti textu. Program je správně rozezná díky vertikální vzdálenosti alespoň ob řádek a zcela odlišným horizontálním zarovnáním z obou stran. Pro lepší znázornění je na obrázku Obrázek 2.2: Vyznačení tolerančních mezí mezi jednotlivými oblastmi. uvedený nákres, jak se tolerance vzdáleností a zarovnání oblastí vyhodnocují. Na oblasti je také použit princip vyřazení šumu popsaný v kapitole Vyřazení šumu. Tyto oblasti se dále rozdělují na co nejmenší části a rozhoduje se o jejich vyhození ze souboru, ve kterém máme uložený text s uchovaným tokem znaků.



Obrázek 2.2: Vyznačení tolerančních mezí mezi jednotlivými oblastmi.

Z levého horního rohu se program snaží narazit na oblast textu (posloupnosti jedniček a trojek), jakmile objeví, pokračuje dále na řádku. Objeví konec a zapamatuje si šířku x. Prohledá oblast směrem dolů, kde najde konec a hledá přesně v té stejné šířce s +- tolerancí zarovnání a vertikální vzdálenosti další oblasti stejným způsobem. Až přečte poslední oblast v šířce od začátku dokumentu zleva do x, posune se od x do y, kde y je konec další oblasti napravo. Jestliže žádnou nenajde, posune své hledání o řádek níže. Oblasti mezi sebou na řádku musí splňovat minimální rozestup horizontální vzdálenosti.

Díky tomuto principu program je schopen odlišit např. nadpis u každé kapitoly, záhlaví, zápatí, popisek grafu, zbytky z tabulek, rovnic, obrázků, rámečků apod. a jejich znakové složení. V podstatě kontroluje a filtruje kratší úseky textů, které se vykytují ve všech směrem kolem hlavních odstavců článků. Může se jednat i o vysvětlivky nebo poznámky pod čarou.

Nadpisy nejsou součástí věty a jsou samostatným celkem, který ve výsledku nechceme, tudíž každou kratší zpravidla jednořádkovou oblast dvojek neukončenou speciálním znakem v poli `larr` (tečkou) odstraníme.

2.5 Požadavky pro spouštění

Skripty jsou psány v unixovém shellu Bash, Perlu a jazyce Python verze 2.7.3. Kromě běžného operačního systému unixového typu (Linux, BSD) nemá program žádné extra požadavky ať už HW či SW. Čím rychlejším CPU a čtením disku bude PC disponovat, tím rychlejší bude běh programu.

Importované moduly pro regulární výrazy `re` a zpracování parametrů `getopt` jsou součástí Python. Problém může nastat kvůli absenci modulu `ply`, který bude nutné doinstalovat. Program nepoužívá a nevyžaduje běh operačního systému v grafickém uživatelském rozhraní. Spouštění, nastavení konfiguračního souboru nebo úpravu kteréhokoliv slovníků je možné z příkazové řádky editorem. Pro extrakci z PDF je vyžadován nástroj PDFTOTEXT (`sudo apt-get install pdftotext`), který však již je standardním balíčkem Linuxové distribuce Ubuntu. Pro převod z obrázků formátu .jpeg je vyžadován nainstalovaný Acroread a OCR nástroj Tesseract příkazy `sudo apt-get install acroread (tesseract)`. Obě dvě aplikace jsou k dispozici volně ke stažení.

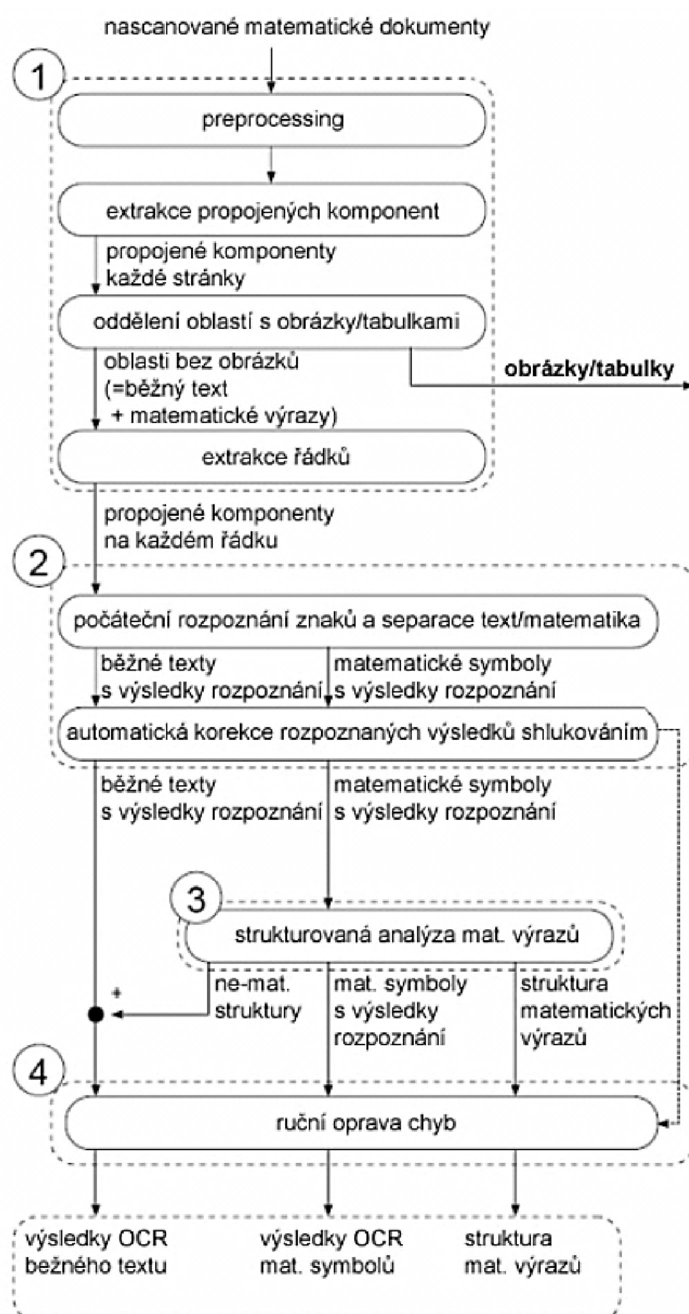
2.6 Ostatní

Data v typických Amerických formátech (MM-DD-YY, MM/DD/YY, MM/DD) nebo Britských, kde první dvojčíslí značí den v měsíci, program rozepíše na plné znění, tak jak se běžně vyslovuje.

Obrázky, tabulky a grafy se anglicky označují Figures, Tables a obvykle zkrácenými titulky „Fig." A „Tab." pod objektem. Proto je zřejmé, že v pravidelně odsazených oblastech pod nebo nad tímto označením bude „smetí". Tyto oblasti plus mínus pár řádků od označení je vhodné také odstranit.

3 OCR proces

Optical Character Recognition (OCR) standard vznikl již v roce 1966, přičemž programy i dnes běžně zaměňují některá písmena abecedy. Záleží na kvalitě vstupu, použitého rozlišení při digitalizaci, kontrastu snímaného objektu a stylu nebo rotace písma. Digitalizace pomáhá kromě možnosti strojového čtení a úpravy dokumentu také k rychlejšímu vyhledávání.



Obrázek 3.1: Workflow OCR. Převzato z [10].

Proces OCR probíhá v následujících několika krocích:

1. skenuje jednotlivé stránky do digitální podoby obrázku nejčastěji formátu jpeg
2. provede obrazové úpravy na získaném obrazu dokumentu
3. obraz rozdělí na oblasti tzv. segmenty
4. rozpoznávání jednotlivých oblastí a znaků
5. tzv. lexikální postprocessing, který je založen na jazyku textu a jeho lexikálních pravidlech
6. uložení výstupu do souboru námi zvoleného formátu

OCR rozpoznává každý znak zvlášť. Pokud je písmo stejně široké, je rozpoznání daleko jednodušší oproti běžně používanému proporcionálnímu písmu, u kterého se znaky navzájem dotýkají a nemají stejnou šířku. Svoji roli hraje také kontrast. Skenování probíhá v přednastavených typech dokumentů, s různým předpokládaným kontrastem pixelů v objektu, jakými jsou noviny, perokresba, fotografie apod. Uživatelská volba ulehčí OCR programu vhodněji použít režim zpracování obrazu tak, aby proces byl schopen podávat co nejlepší výsledky. Natočení písma nebo šum kolem textu také mate proces vyhodnocování. Samozřejmě nelze očekávat v sebelepším programu či nejpřesnějším režimu a vysokým rozlišení snímání vynikající výsledky, pokud vstupní obraz je rozmazán na zažloutlém pozadí s vybledlým textem (viz obrázek Obrázek 3.2: Rozmazaný starý kus novin. Převzato z URL.. S každým horším stupněm kvality obrazových dat chybovost výrazně roste a to zhruba exponenciálně. Na běžném čistém fyzickém dokumentu však dnešní nejdokonalejší OCR programy dosahují příznivou 97% až přes 99% úspěšnost rozeznání všech znaků. Vydavatelé obvykle rádi uvádějí svým produktům papírově nadsazenou hodnotu 99,9% úspěšného rozpoznání.

SPRINGFIELD GAZETTE.

Obrázek 3.2: Rozmazaný starý kus novin. Převzato z URL[5].

[5] http://jasonlefkowitz.net/wp-content/uploads/2012/05/newspaper.jpg

3.1 Implementace OCR filtru

Skript pracuje se svými interními slovníky, do kterých si ukládá databáze korektních slov, běžných defektů a návrhů změn. Každé nalezené slovo lexikálním analyzátorem [8] je vyhledáváno právě v těchto databázích. Slovíčka v korektním stavu jsou rozděleny na začínající velkým nebo malým písmenem `upper.wlist` a `lower.wlist`. Jestliže slovo začíná malým písmenem, je prohledáváno v databázi `lower.wlist`. Pokud se jedná o slovo na začátku věty nebo názvu či jméno, je vyhledáváno nejdříve v `upper.wlist` a následně v případě neúspěchu nalezení v `lower.wlist`. Opravuje se tím chybné rozpoznání velikosti písmen, neboť OCR např. zaměňuje běžně velké „S“ a malé „s“ nebo „V“ a v“. Obvykle malá písmena OCR zamění za jejich velké protějšky.

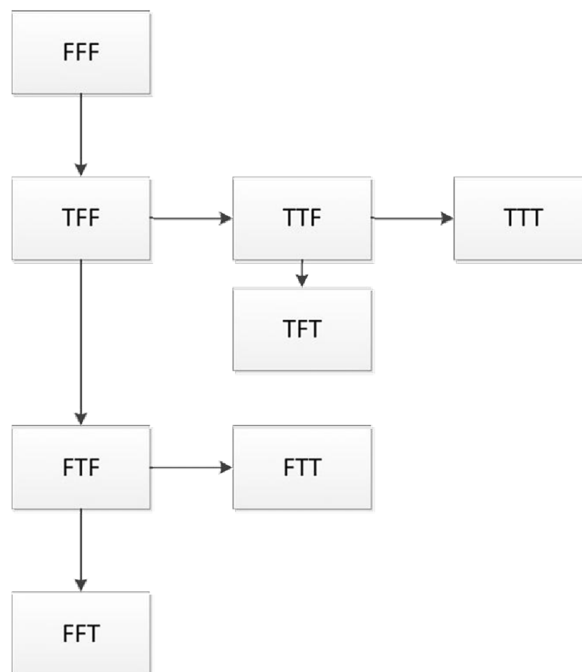
Jestliže se tedy slovíčko nachází v databázi se známými jmény lidí, společností, měst atd. (slovník se učí a automaticky sám rozšiřuje), je korektní a přechází se na řetězec následující v pořadí. Jinak se hledá v malých a případná shoda se ve výsledku přepíše malým písmenem.

Skript si ještě uchovává databázi slov s rizikovými znaky [11] tak, aby mohl uživateli navrhnout možnou změnu korektně vyhledaného řetězce, např. u slovíčka OCR programem vyhodnoceným jako „modern“, avšak kontextově správným slovem „modem“ nebo „day“ a „clay“. Tyto nedostatky jsou výsledkem nedokonalého alfanumerického systému z dob Římanů a Řeků, kteří jej stvořili. Některé kombinace znaků bohužel OCR nerozezná při určitém fontu písma, typografické techniky a kvalitě a režimu skenování. Vznikají tak texty s přehozeným významem, které kontrola pravopisu bohužel neodhalí. To je problém, pokud vyžadujeme digitalizovat desítky tisíc stránek, kdy ruční kontrola nepřipadá v úvahu. Odlišná a složitější situace nastává, kdy ani jeden ze slovníků neobsahují hledaný řetězec.

V poli omylů `oprava_ocr_lvl1` jsou zaznamenány záměny znaků (viz obrázek Obrázek 3.4: Seznam častých záměn znaků OCR [11]. [11]. Funkce `generujMoznaSlova()` rekurzivně volá sama sebe a na základě pole omylů vytváří možné nové varianty správného tvaru slova, které posléze nabídne uživateli k náhradě. Jestliže skript narazí např. na ukázkově smyšlený podřetězec „fff“, nenajde celé slovíčko v patřičném slovníku a v poli náhrad má uvedeno, že „f“ je zaměňováno s „t“, vygeneruje možné varianty a vyžádá si od uživatele tu správnou (včetně možnosti výchozí) nebo uživatel přikáže smazat všechny tyto řetězce z dokumentu.

Algoritmus generování nových tvarů slov vypadá takto:

```
def moznaSlova(self,slovo):
 def generujMoznaSlova(slovo, uroven=1, index=0):
  if index > len(slovo):
    return ""
  res=[slovo]
  nahrada=True
  while(index<len(slovo)):
    nahrada=False
    for item,value in self.oprava_ocr_lvl1.iteritems():
      if item == slovo[index:index+len(item)]:
            ind=index
            noveSlovo=slovo[:ind]+value+slovo[ind+len(item):]
            dalsi=generujMoznaSlova(noveSlovo, uroven, ind+len(value))
            res.extend(dalsi)
    index+=1
return res
```



Obrázek 3.3: Ukázka rekurze generování možných náhrad.

Funkce `generujMoznaSlova()` vytvoří někdy i několika desítkový seznam možných náhradních řetězců. Záleží na počtu často zaměňovaných znaků ve výchozím slově. Pro nález „terminator", ve

kterém dochází k záměně znaků z obrázku Obrázek 3.4: Seznam častých záměn znaků OCR [11].. „rm“ => „nn“; „m => rn“; „l“ => „i“; „t“ => „f“ a naopak, program vygeneruje níže uvedené možnosti, jelikož se zatím nález nevyskytuje v databázi.

```
['terminator',      'ferminator',      'fenninator',
'fennlnator',       'fennlnafor',      'fenninafor',
'ferrninator',      'ferrnlnator',     'ferrnlnafor',
'ferrninafor',      'fermlnator',      'fermlnafor',
'ferminafor',       'tenninator',      'tennlnator',
'tennlnafor',       'tenninafor',      'terrninator',
'terrnlnator',      'terrnlnafor',     'terrninafor',
'termlnator',       'termlnafor',      'terminafor']
```

Uživatel bude vyzván k ručnímu zásahu, avšak jelikož ani jeden vygenerovaný řetězec se ve slovníku nenachází, tak nedostane na výběr možnosti kromě přečteného „terminator“, pokud však neupravil konfigurační soubor, aby všechny tyto pravděpodobně nesprávné tvary mu byly nabídnuty. Samozřejmě pokud mu přepíšeme svůj vlastní řetězec, skript ho nahradí dle naší libosti a uloží si toto pravidlo pro každý další takový nález.
Komplikovaná situace nastává právě u kritičtějších znaků a matoucích slovíček.

```
carriage => carnage  severity => seventy entitles => entities
modern => modem      acids => adds       stories => stones
dock => clock        gentler => gender   title => tide
timed => tinned      rioting => noting   refrained => reframed
```

Zatím ne příliš rozšířená databáze výše uvedených pravidel je zpracovávána a po každém nálezu, ačkoli je slovo zcela korektně správně na vstupu, uživatel musí ručně vyhodnotit situaci. Program by musel znát kontext autorovy myšlenky. Můžou se objevit dokonce dva takové výrazy za sebou např. „modern modem“.

Při testování nástroje Unixového Tesseract OCR programu při snímání s rozlišením 400 DPI nebo menším a ze zdroje [11] byly zjištěny následující omyly, kdy OCR zaměňuje podobné znaky či kratší řetězce znaků. Jsou to potenciálně chybné záměny vytvářející však korektní jiná slova vyznačena níže.

ab → ah
ah → ab
ac → ae
ae → ac
ai → al
al → ai
aq → ag
ag → aq
ay → av
bb → hh
bb → hb
bc → be
be → he
he → be
be → bc
bi → bl
bl → bi
bo → ho
ho → bo
bu → hu
hu → bu
ca → ea
ea → ca
cd → ed
ed → cd
cf → of
of → cf

ci → cl
cl → ci
ci → d
cl → d
co → ca
cr → er
er → cr
cs → es
es → cs
cv → cy
db → dh
di → th
th → di
dl → di
di → dl
ei → el
el → ei
et → el
fc → fe
fc→ fo
fi → fl
fl → fi
fy → ty
ty →fy
gi →gl
gl →gi
gu →qu
qu →gu

hy →by
ia →la
la →ia
ic → ie
ie → ic
ie → le
ii → il
ii → u
il → ll
il → ii
io → lo
ld → id
li → ll
ll → li
ll → ll
lm → im
lm → lm
lt → lt
lv → iv
lv → lv
or → ur
or → ür
ot → et
et → ot
pi → pl
pl → pi
qa → ga
ga → qa
qo → go

go → qo
qs → gs
qu → gu
ro → ru
ru → ro
sa → se
tb → th
tt → ff
ub → uh
uc → ue
ue → uc
vs → us
wb → wh
nn → m
ri → n
rl → d
rn → m
tl → d
in → m
m → in
m → nn
m → rn
d → tl
d → cl
d → ci
n → ri
u → ii
a1 → al
al → a1

Obrázek 3.4: Seznam častých záměn znaků OCR [11].

3.2 Vliv DPI na úspěch rozpoznání

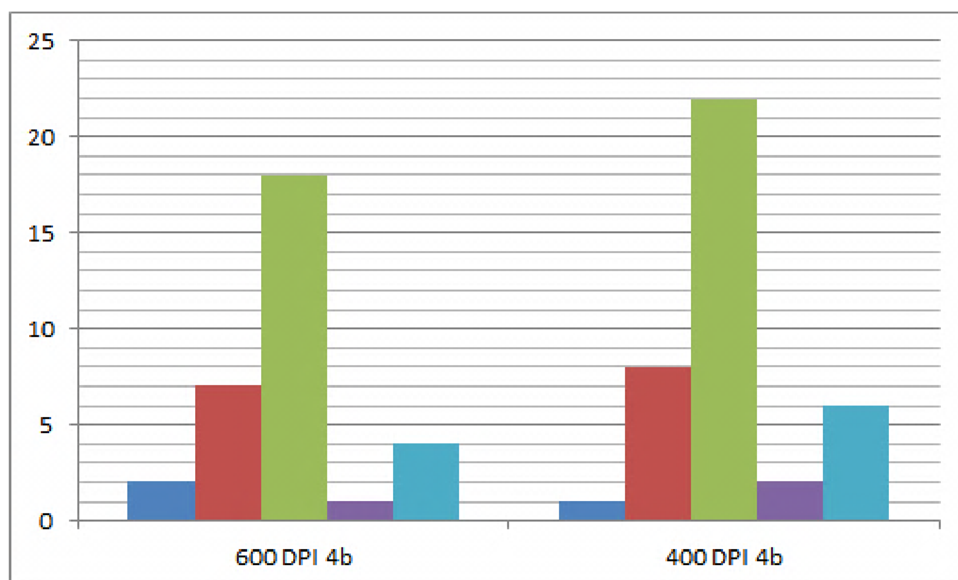
Počet bodů na palec snímaného obrazu logicky ovlivňuje výslednou kvalitu výstupu. Takovým nepsaným standardem se považuje rozlišení 300 DPI, které podává příznivé výsledky i s ohledem na přijatelnou časovou dobu vynaloženou získáváním obrazu. Nejnižší použitelnou hodnotou se považuje kolem 150 DPI. Rozdíl mezi 200 a 300 DPI je znatelný, ačkoliv na první pohled se nemusí zdát. 300 * 300 = 90 000 a 200 * 200 = 40 000 obrazových bodů na palec[6] čtvereční, což je více než dvakrát menší počet potřebných pixelů k detekci znaků v dokumentu. Více bodů umožní počítači pracovat v přesnějším režimu, protože má k dispozici více dat, na základě kterých může správně rozhodovat o tvaru znaku. Je nutno také podotknout, že nikdy nezískáme pomocí OCR lepší data, než je kvalita vstupů samotná.

Na následujícím zvětšeném obrázku Obrázek 3.5: 300 DPI nalevo oproti 200 DPI napravo. si můžete všimnout, jak snadno by mohlo dojít k záměně znaků „B“ a „8“ při použití nižšího rozlišení DPI.

[6] Velikost palce (z angl. inch) je definována jako jedna dvanáctina stopy, nebo také 2,54 cm.

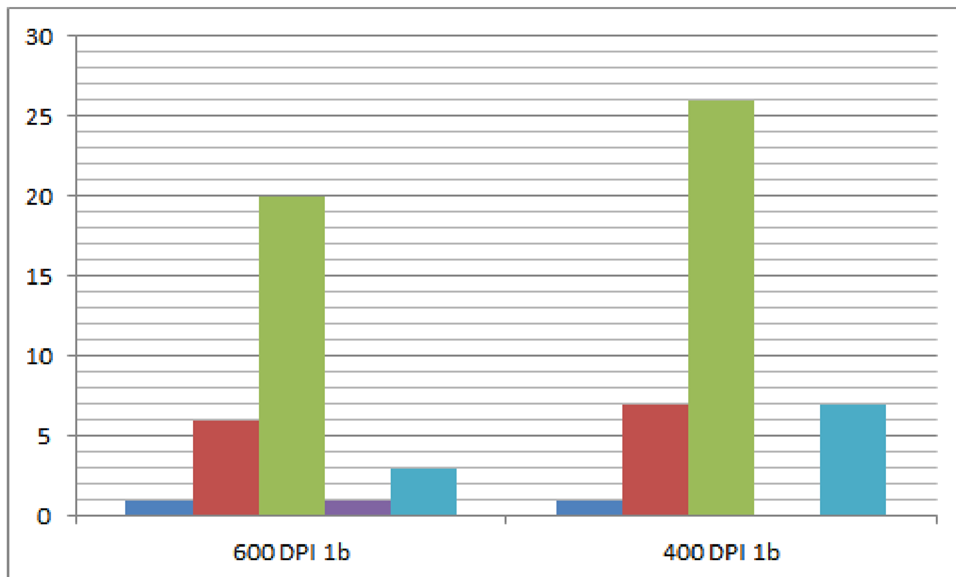
Obrázek 3.5: 300 DPI nalevo oproti 200 DPI napravo.

Určitý vliv má také barevná hloubka skenování. Znázorníme si na sloupcových grafech vliv právě barevné hloubky a DPI na počtu chyb ve výstupu komerčních programů FineReader 8 a InftyReader. Pro příklad bylo detekováno pět náhodných TIFF obrázků bez komprese, a to s anglickým i českým textem[7]. Chybovost byla následující:

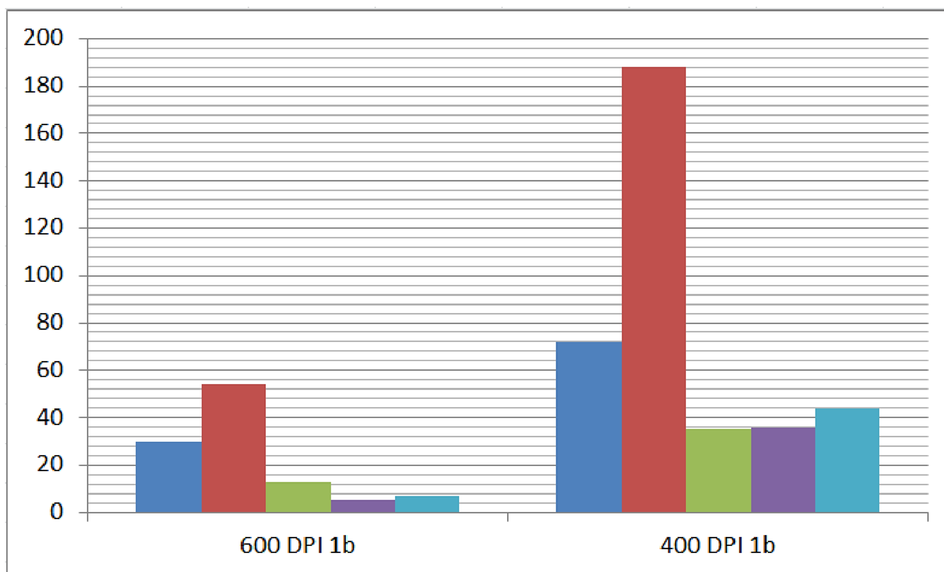


Obrázek 3.6: FineReader a závislost počtu chyb na rozlišení (4b color depth).

[7] Informace dostupné na http://is.muni.cz/th/60738/fi_m/dipl.pdf.



Graf 3.1: FineReader a závislost počtu chyb na rozlišení (1b color depth).



Graf 3.2: InftyReader a závislost počtu chyb na rozlišení (1b color depth).

FineReader oproti InftyReaderu si vede o poznání lépe a především s nižším nastavení rozlišení. Zde je vidět markantnější rozdíl oproti menšímu DPI. Je vidět, že použitím 600 DPI v IR docílíme znatelného zlepšení. Barevná hloubka nastavena na 4 bity se také pozitivně promítne na výsledku. Ačkoli takové kvalitnější snímání trvá delší dobu, je jednorázovou akcí a vyplatí se, než poté ručně vyhledávat a přepisovat velké množství znaků. Špatná kvalita snímání, natočené stránky v obraze a šum JPG snímků zvyšují náklady. Zejména na diakritiku a interpunkci mají tyto faktory katastrofální následky.

3.3 Nerozpoznání rozestupu mezi slovy

OCR má také problémy s odhadem velikosti mezer mezi slovy. Jinými slovy stává se, že dvě sousední slova chybně sloučí do jednoho. Na druhou stranu existují hlavně delší slovíčka, která se právě i z několika kratších skládají, což řešení tohoto nedostatku komplikuje. Proto je implementován algoritmus funkce `generujMoznaSlovaDelenim()`, který se snaží dělit slova na všech možných místech a vyhledávat podřetězce. Po každém znaku se řetězec pomyslně rozdělí na více částí. Kratší podřetězce pak vyhledává ve své databázi známých slov. Slovník `allwords.wlist` obsahuje desítky tisíc anglických slovíček a neustále se svým používáním rozšiřuje a učí.

Pro příklad si ukážeme slovo „background“. Na obrázku Obrázek 3.7: **Rozdělení slova na možné podřetězce.** jsou znázorněna místa, ve kterých program nalezne vhodné místo pro rozdělení. Jedná se o podřetězce back, ground, round a v informatice taky využívaný výraz ack, který se používá jako zkrácený zápis pro acknowledgement (potvrzení např. příchozího paketu v síti). Pokud řetězec je možno rozdělit celý beze zbytku, přidá se k možným řešením a uživateli budou tyto možnosti nabídnuty.

Obrázek 3.7: Rozdělení slova na možné podřetězce.

Situace může nastat i zcela opačná. OCR rozpozná více kratších řetězců namísto jednoho. Proto ve vylepšení mého programu by se mohla implementace ošetření tohoto teoretického předpokladu objevit.

3.4 FineReader

Profesionální OCR engine používá např. komerční produkt FineReader v aktuálně nejvyšší verzi 11, který umí rozpoznat text mezi téměř dvěma stovkami světových jazyků včetně diakritiky, podporuje

ukládání do výstupních formátů jako jsou doc(x), rtf, xls(x), pdf atd. a dokáže uchovat styl a strukturu textu. Jedenáctá verze slibuje až 30% zlepšení rozpoznávání pro tištěné obrazy v nižší kvalitě. V dnešní době digitálních fotoaparátů a mobilních telefonů či tabletů se nám FR určitě hodí. Snímání Vám usnadní průvodce ve třech krocích s intuitivním uživatelským rozhraním.

3.5 OCR OpenSource

Výborným kompromisem mezi cenou a kvalitním rozpoznáním obrazových dokumentů je již nějakou dobu volně dostupný Tesseract. Vyvíjela ho společnost Hewlett-Packard zpočátku pro komerční užití do roku 1994. Razantnější zlepšování algoritmů se poté však již nekonalo a v roce 2005 HP poskytl svůj program pod OpenSource licencí. Aktuálně ve vývoji spolupracuje Google a program vydává pod licencí Apatche 2.0. Tesseract engine obdržel v roce 1995 na University of Nevada Las Vegas hodnocení jako jeden ze tří nejlepších OCR programů, což je vynikající výsledek pro nekomerční software. Na druhou stranu většina běžných uživatelů se mu určitě kvůli chybějícímu GUI (Graphics User Interface) vyhne obloukem i přesto, že běží na Windows a neoficiálně na Mac OS X.
Tesseract pracuje pouze v příkazové řádce, nemá žádné grafické rozhraní a celkově je spíše jeho programová část, která je implementovaná v jazyce C++, použitelná pro jednodušší struktury textu. Právě jeho nedokonalosti využiji k testování implementace mého skriptu.

4 Návrh

Uvedeme si požadavky na funkčnost software. Program vyžaduje nastavení určitých proměnných. Konfigurací je však tolik, že je pro rychlejší spouštění zaveden konfigurační soubor, který je nejdříve doporučeno zkontrolovat a případně poupravit hodnoty a požadavky na běh skriptů. Průběh funguje tak, že před dokončením posledních kroků vyzve program uživatele, aby odsouhlasil nebo popřípadě změnil navrhované náhrady slov dle libosti. Skript tedy vygeneruje seznam změn do textového souboru, který uživatel následně zkontroluje a upraví. Z finální verze tohoto souboru se mezivýsledek přepíše na výstup. V následujících podkapitolách jsou podrobněji rozepsány principy skriptů. Předposlední část graficky vyobrazuje celé schéma, jak jednotlivé procesy postupují chronologicky po sobě.

Pro implementaci jsem využil především jazyka Python 2.7 [2]. Je to jazyk jednodušší, ale vykazuje efektivní výkony a nenáročnou implementaci. Psát v Pythonu přitom lze jak strukturovaně, tak objektově.

4.1 Vstup do filtru

Skript vyžaduje jakýkoliv dokument ve formátu plain text (obvykle přípona .txt). Není nutná nějaká další úprava nebo vnitřní struktura dokumentu. Program se snaží filtrovat a opravovat vše, co je jen možné nehledě na to, odkud textový soubor pochází. Samozřejmě ke vhodnému, plnému a smysluplnému využití této práce, hlavní skript předpokládá na vstupu soubor se špatným rozložením textu, se zbytky grafů, tabulek, rovnic, obrázků, formulí atd. z extrahovaného souboru typu PDF, špatně rozpoznanému výstupu OCR programu nebo webu (HTML). S ostatními strukturami např. texty ve značkovacím jazyce XHTML, LaTeX nebo kterémkoliv jiném, budou zpracovávány obdobně, avšak při implementaci na ně nebyl brán příliš velký zřetel, jelikož toto není předmětem této práce.

Skript se spouští s několika parametry a to `-i <inputfile>` a `-o <outputfile>`, kde první je vstupní soubor a druhý výstupní přefiltrovaný soubor. Argumentem `-s <file for OCR corrections>` předáme název souboru se seznamem pravidel náhrad pro OCR chyby. Přepínač `-h` vytiskne na standardní výstup nápovědu. Uživateli se doporučuje spouštět skript se soubory typu PDF, je však nutné nastavit konfigurační soubor, aby se nejdříve extrahoval text do holého textového formátu. Adobe Reader formát a konvertor PDFTOTEXT se využije k převodu do dvou odlišných struktur textu, které se využijí k implementaci oblastí popsané v kapitole Oblasti.

4.2 Základní filtr

Skript `my_pdf2text.sh` nejprve soubory rozdělí do stránek a dále až do posledního kroku stránky zpracovává. Hlavní `my_pdf2text.sh` volá ostatní skripty `rmv_emails.py`, `rmv_headers_footers.py`, `rmv_links.py`, které po řadě odstraňují emaily, záhlaví, zápatí, odkazy a citace. Kromě záhlaví, zápatí a citací skripty pracují s regulárními výrazy (RV). Kroky probíhají jeden po druhém. Základní filtr vrací a předává upravené texty dalším skriptům.

`[\w\-\.]+@[\w\-\.]+\.\w{2,5}` - RV pokryje emailové adresy

`([\w]+://|www\.)[\w\.\-/#&=\?+]+` - RV pokryje odkazy

4.3 Skripty, tvorba pravidel a slovníků

Na řadě je proces tvorby a použití slovníků a pravidel především pro nedostatky rozpoznávačů OCR. Dle zvoleného přepínače v konfiguračním souboru se spustí chronologie skriptů buď pro univerzální vstup ve formátu holého textu nebo pro dokument typu PDF či obrázek. Pro PDF se očekává jiná extrahovaná struktura textu a využije se pro tento typ souboru princip odstraňování na základě oblastí v kapitole Oblasti tak, aby se jednoznačně dala určit nesourodost textu. V každém nutném případě interaktivity s uživatelem během procesu filtrování, program otevře patřičný soubor s pravidly úprav nebo slovníkem v textovém editoru. Preferovaný editor je možné změnit v konfiguračním souboru před každým spuštěním programu. Jako výchozí textový editor je přednastaven Vi, neboť je součástí většiny Linuxových distribucí. Gramatická pravidla, která pojímají aktuálně přes šedesát tisíc korektních slovíček, databáze zkratek a náhrad a výslovnostní slovník, do kterého se zapisuje anglická výslovnost všech nalezených slov ve vstupních dokumentech, se ukládají globálně a každým vstupem se neustále rozšiřuje. Je tedy vhodné tyto soubory s příponami .wlist přenášet a zálohovat. Výslovnostní slovník má svou databázi každého korektního slova, na které kdy program narazil. Pro neznámé slovíčko program vygeneruje jeho výslovnost pomocí programu Sequitor a uloží ji.

4.4 Konečný filtr

Pro adaptivní rozpoznávače či detektory řečové aktivity je vhodné upravit výstupní text. Ten by pro takové cíle neměl obsahovat cokoliv kromě klasické anglické abecedy. V posledním kroku skript odstraní veškeré znaky kromě písmen a mezer mezi slovy. Navíc převede všechna velká písmena na

malá včetně začátku vět. Tato možnost lze vypnout v konfiguračním souboru. Může a nemusí se tato poslední fáze uživateli hodit. Záleží na účelu spuštění aplikace.

4.5 Uživatelská konfigurace

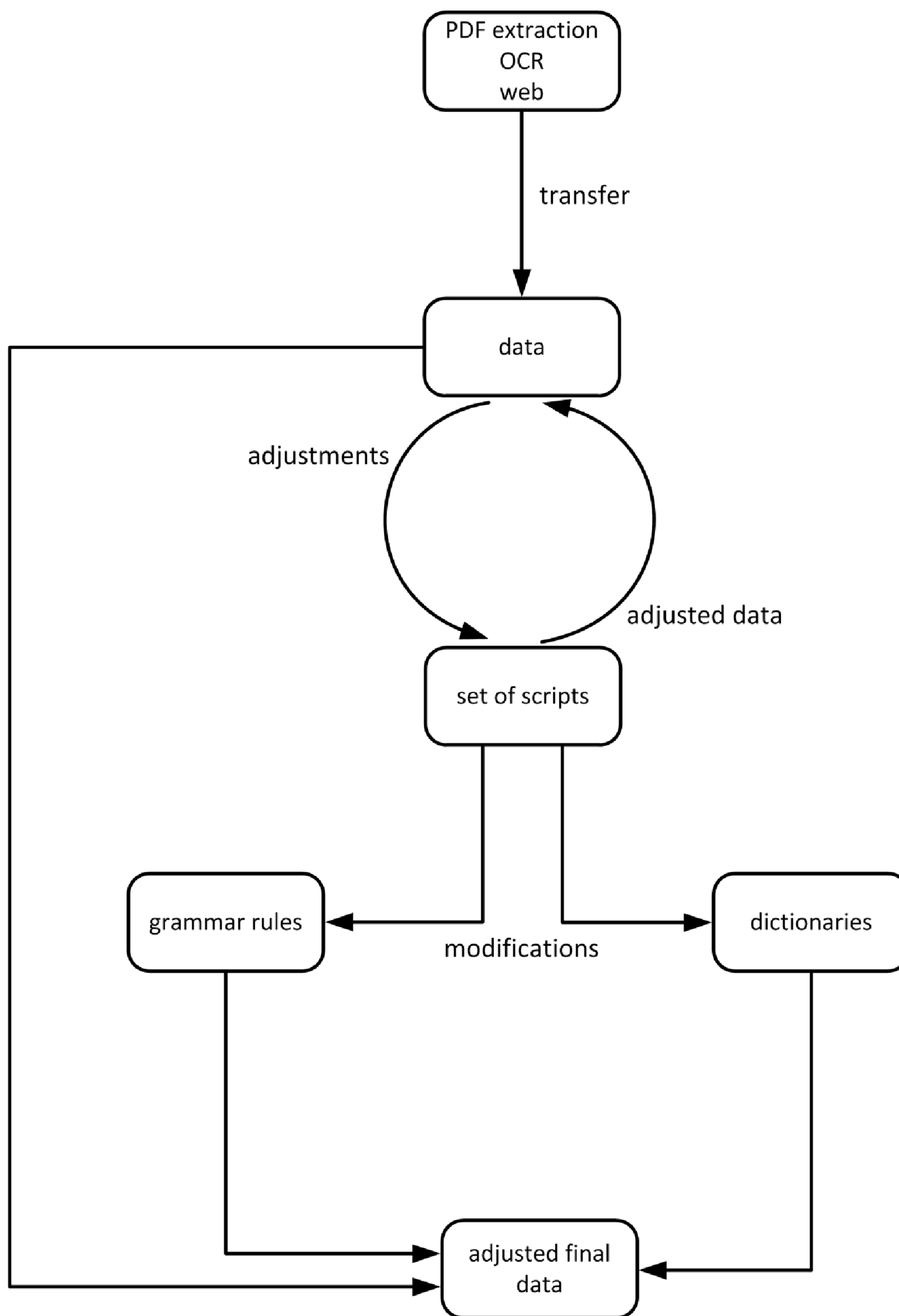
Pro filtrování z výstupu OCR je vhodné uživatelsky konfigurační soubor upravit tak, aby skript vyhledával a opravoval právě chyby tohoto druhu. Skript pracuje s interními slovníky a vyhledává každé slovo ve své databázi známých řetězců. Jestliže najde shodu, slovo je v pořádku a pokračuje na další. V opačném případě si poznamenává řetězec do souboru změn. Po tomto kroku vyzve hlavní skript uživatele, aby překontroloval soubor změn a vybral si z možných variant správnou korekci nebo vytvořil slovo nové. Skript následně nahradí chyby dle libosti uživatele.

Některé funkce tohoto implementovaného postupu jsou relativní a pracují s různými vstupy odlišně. Konfiguračním souborem můžeme chování do určité míry měnit, aniž by uživatel rozuměl kódu a přepisoval konstanty či proměnné. Otestuje si na prvním typu vstupního dokumentu, které hodnoty budou pro něho podávat ideální výsledky. Program pak nebude vyhazovat zbytečně moc nebo naopak moc málo textů. Pro další podobné dokumenty hodnoty již jen ponechá.

parametr	**popis [možnosti]**
use_OCR	zapnout/vypnout korekci OCR [0/1]
interactive	zapnout/vypnout vyžádání kontroly a editace uživatele [0/1]
auto_open	zapnout/vypnout manuální otevírání souboru [0/1]
text_editor	nastavení spouštěného textového editoru [vi, gedit, vim, kate, sublime...]
keep_unknown_words	defaultně ponechávat nenalezená slova (OCR korekce) [0/1]
hide_unknown_suggestions	skryje uživateli neznámé vygenerované návrhy [0/1]
knockout_limit	nastavení hranice pro vyřazení vět a oblastí [x]
final_filter	vyhodí vše kromě abecedy a přepíše na malá písmena [0/1]
coefficient_lower	koeficient malých písmen [x]
coefficient_upper	koeficient velkých písmen [x]
coefficient_names	koeficient názvů [x]
coefficient_abbreviation	koeficient zkratek [x]
coefficient_numbers	koeficient čísel [x]
coefficient_symbols	koeficient speciálních symbolů [x]
coefficient_symbols_permitted	koeficient povolených symbolů [x]

Tabulka 4.1: Konfigurace aplikace.

4.6 Schéma návrhu



Obrázek 4.1: Schéma průběhu filtrace.

5 Experimentování

V této kapitole se podíváme, jaké výsledky podává program. Testovat ho budu na různých vstupních datech a srovnám je. Jelikož budu porovnávat výsledky jako textové soubory mezi sebou a potřebuju znát znakové rozdíly, využiji k tomu vhodného algoritmu Levenshtein Distance [14] neboli Levenshteinovy vzdálenosti. V Perlu implementovaným jednoduchým skriptem s využitím modulu `Text::LevenshteinXS` snadno zobrazím nesourodost mezi srovnávanými texty. Levenshteinova vzdálenost vyjadřuje rozdíl mezi dvěma řetězci. LV definuje tedy minimální počet znaků, které je třeba smazat, nahradit nebo přidat jednomu řetězci tak, aby byly oba porovnávané řetězce totožné. Objektivně můžu takto změřit rozdíl a popsat výkonnost mezi řešeními. Výsledky promítnu do názorných grafů.

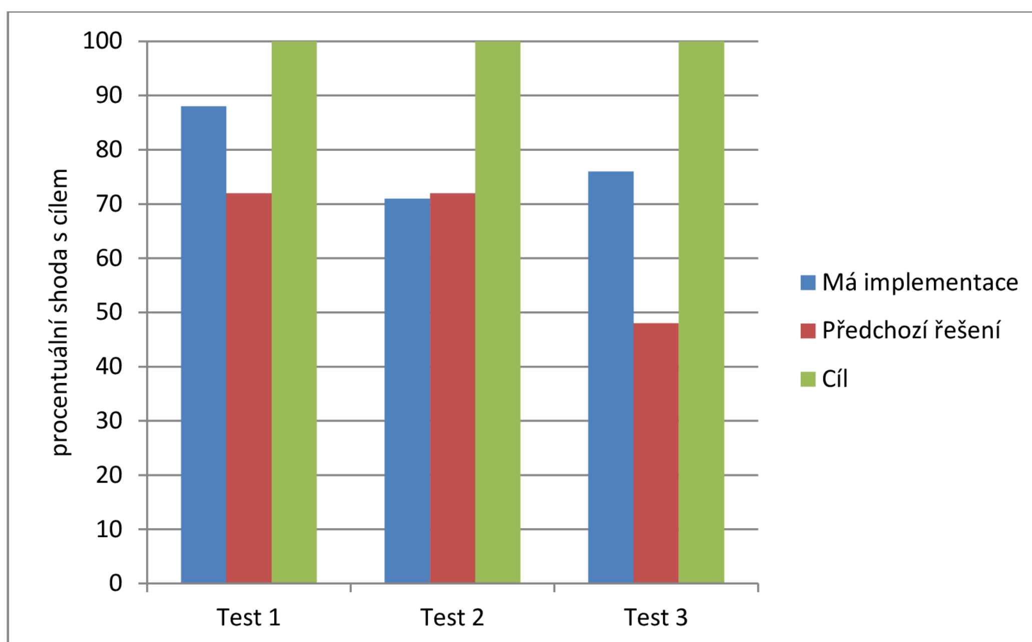
5.1 Srovnání s předchozím řešením

Můj software má za účel zdokonalit řešení, kterým se zabýval Ing. Igor Szöke Ph.D. ve své práci. Pan Ing. Igor Szöke Ph.D. je rovněž i mým vedoucím bakalářské práce. Tudíž jsem měl možnost přímé konzultace v oblasti filtrování textu. Cíle mého software a předchozí řešení se však mírně liší v některých ohledech, nicméně pokusíme se je srovnat co nejobjektivněji a znázornit, jak si můj program vede a v čem je lepší, popřípadě pokud v něčem zaostává.

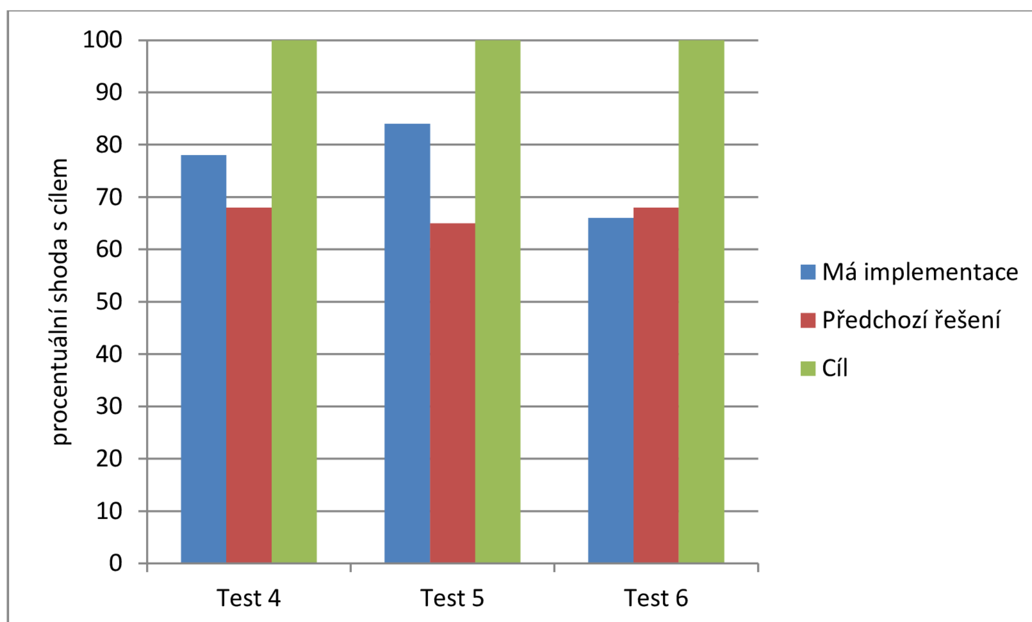
5.2 Srovnání s referenčním řešením

Rozdíl mezi mým software a předchozím programem znázorním oproti referenčnímu textu. Referenční řešení je má manuálně vyhotovená filtrace odborných textů extrahovaných z PDF publikací. Originální dokumenty obsahovaly nepřeberné množství grafů, tabulek, obrázků, vzorečků, citací, emailů, OCR překlepů apod. Během vývoje mého programu jsem se právě tomuto řešení snažil co nejvíce přiblížit, jelikož je zdrojem dokonalého žádoucího výstupu filtrace.

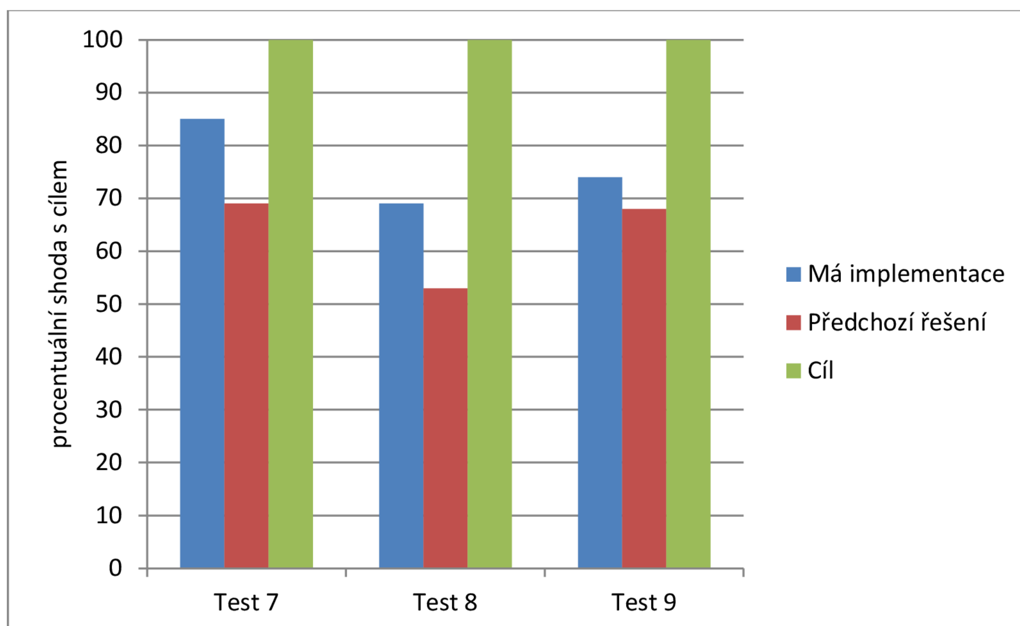
5.3 Testy



Graf 5.1: Testy PDF extrakce.



Graf 5.2: Testy OCR chybovosti.



Graf 5.3: Testy univerzální.

Testování probíhalo na odborných dokumentech a publikacích ve formátu PDF, které mi byly interně dodány. U aplikace byla zapnuta OCR korekce a nastavena výchozí konfigurace. Z dokumentů byl extrahován text konvertorem PDFTOTEXT nebo nástrojem Acroread a OCR aplikace Tesseract. Z grafů je zřejmé, že program si v porovnání s předchozím řešením nevede špatně a ve většině testů podává lepší výsledky. Některé testy dopadly velmi těsně. Pravděpodobně v oněch testovacích dokumentech jsou názvy kapitol, nadpisy, rámečky apod. ve struktuře textu umístěny tak nešťastně, že detekce oblastí je neodhalí správně, tudíž projdou sítem. Domnívám se, že pečlivým otestováním různých konfigurací koeficientů množin znaků (viz kapitola 3.1) a změny hranice vyhazování na přísnější hodnotu pro každý takový vstup by měly pozitivní vliv a program by podával o poznání lepší výsledky pro konkrétní případy.

6 Závěr

Předmětem této práce bylo pomocí sady skriptů zdokonalit převod různých typů dokumentů do čistě textové podoby (plain text). Svou práci jsem v této technické zprávě ukázal a popsal zde principy řešení. Snažil jsem se popisovat techniky a problematiku stručně a výstižně i pro čtenáře, který nemá zkušenosti se zpracováním textu nebo detekci znaků. Program byl navržen tak, aby jeho zacházení nebylo příliš uživatelsky náročné a zároveň podával co nejlepší výsledky. Závěrečné alfa-testování ukázalo několik nedostatků, které však byly včas odstraněny i přes časovou tíseň před koncem data odevzdání. Já si tak tímto pro mě větším projektem odnesl cenné zkušenosti.

Skripty se podařilo vypracovat tak, aby extrahovaný textový soubor vyčistily, výsledný text byl čitelný, dával smysl (při nejhorším náznak) a neobsahoval zbytky různě vyskytujících se znaků z převodu grafů, seznamy literatury, obsahů tabulek, zbytků vzorců, popisků obrázků apod. S nedokonalostí OCR či extrahováním textu si můj software poradí a pozitivní výsledky experimentování se podařilo přednést v páté kapitole.

Pohled dalšího vývoje filtrace by se měl zaměřit na zdokonalení. Prosté rozeznání je na vysoké úrovni a podává slušné výsledky, ale na obrazových předlohách v horší kvalitě se současnou technikou toho moc nevykouzlíme. Mohl by se navrhnout program, jehož engine by byl schopen chápat to, co čte nebo alespoň by si uchovával jakýsi přehled o kontextu textu. Samozřejmostí by byla práce se syntaxí a sémantikou různých jazyků. Systém by byl však náročný a musel by mít povědomí o souvislostech mezi souvislosti, přičemž ty by byly tvořeny dynamicky. Zdokonalení přepisu rukopisu do strojem čitelné podoby by se určitě také využilo nebo zlepšení rozpoznání exotičtějších fontů písma.

Literatura

[1] Herold, Helmut: *awk & sed - Příručka pro dávkové zpracování textu*. vydání první. Brno: Nakladatelství Computer Press Brno, 2004, ISBN 80-251-0309-9.

[2] The Python Tutorial. Dokumenty dostupné na URL http://docs.python.org/3/tutorial/index.html, [cit. 2013-04-10].

[3] Bash Reference Manual. Dokumenty dostupné na URL http://www.gnu.org/software/bash/manual/bashref.html, [cit. 2013-04-10].

[4] Number to Word Convertor (Python). Zdrojový kód dostupný na URL http://www.daniweb.com/software-development/python/code/216839/number-to-word-converter-python, [cit. 2013-04-10].

[5] Pilgrim, M.: Dive Into Python 3 [online]. http://getpython3.com/diveintopython3, 2011 [cit. 2013-03-10].

[6] Python Software Foundation: Useful Modules [online]. http://wiki.python.org/moin/UsefulModules, 2012 [cit. 2013-03-10].

[7] Python Software Foundation: Python 3.0 Release [online]. http://www.python.org/getit/releases/3.0, 2013 [cit. 2013-03-15].

[8] PLY (Python Lex-Yacc) [online]. http://www.dabeaz.com/ply/ply.html, 2011 [cit. 2013-04-15].

[9] Regular expression operations [online]. http://docs.python.org/2/library/re.html, 2013 [cit. 2013-04-25].

[10] Digitalizace dokumentů na fakultách Masarykovy univerzity v Brně [online]. https://is.muni.cz/th/342264/ff_b/Plny_text_prace_otevrelova.pdf, 2013 [cit. 2013-05-04].

[11] OCR Production Nightmares [online]. http://www.infogridpacific.com/blog/igp-blog-20130317-ocr-production-nightmares.html, 2013 [cit. 2013-05-05].

[12] Why is OCR at 300 dpi a Standard? [online]. http://scansnapcommunity.com/tips-tricks/1652-why-is-ocr-at-300-dpi-a-standard/, 2010 [cit. 2013-04-20].

[13] The Fourth Annual Test of OCR Accuracy [online]. http://stephenvrice.com/images/AT-1995.pdf, 1995 [cit. 2013-04-21].

[14] Text::LevenshteinXS - An XS implementation of the Levenshtein edit distance [online]. http://search.cpan.org/~jgoldberg/Text-LevenshteinXS-0.03/, 2004 [cit. 2013-04-29].

Seznam příloh

Příloha 1. Plakát

Příloha 2. Obsah CD

Plakát

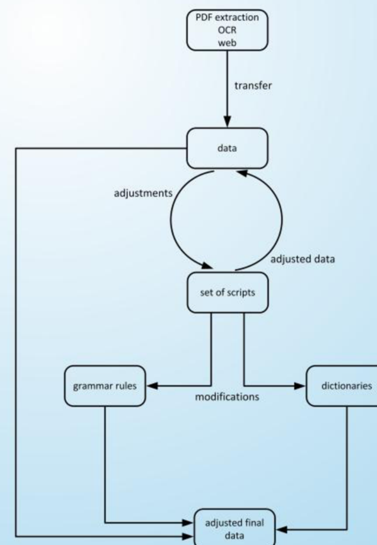
Text filter and corrector

This software helps you to improve the transfer of various types of documents into fully text.

- cleans (extracted) document

- the resulting text is readable, make sense and does not contain any residues of various characters appearing from the transfer of graphs, tables, formulas, etc.

- works universally and does not require input arised only by OCR tools or converting from PDF or web!



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Author: Filip Lehnert
Supervisor: Ing. Igor Szöke Ph.D.
Faculty of Information Technology, Brno University of Technology, 2013

Obsah CD

- Zdrojové kódy implementovaného programu, konfigurační soubor a slovníky v adresáři /src
- Zdrojový soubory této technické zprávy v adresáři /thesis/src
- Technická zpráva ve formátu PDF v adresáři /thesis
- Zdrojový soubor .cdr plakátu v adresáři /poster/src
- A2 plakát ve formátu PNG k tomuto projektu v adresáři /poster