Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozpoznanie fontu v dokumentoch

Bakalárska práca

2017
Dušan Javorek

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozpoznanie fontu v dokumentoch

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Elena Šikudová, PhD.

Bratislava, 2017
Dušan Javorek

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Dušan Javorek
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Rozpoznanie fontu v dokumentoch
Font recognition in documents

Cieľ: Naštudovať teóriu rozpoznávania znakov. identifikovať príznaky, podľa ktorých sa dajú rozlíšiť fonty. Rozpoznať font na naskenovanom dokumente na základe identifikovaných príznakov. Porovnať s dostupnou voľnou aplikáciou.

Vedúci: RNDr. Elena Šikudová, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 27.10.2016

Dátum schválenia: 02.11.2016 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Ďakujem svojej školiteľke RNDr.Elene Šikudovej, PhD. za pomoc, čas a cenné rady, ktoré mi poskytla počas tvorby tejto bakalárskej práce. Taktiež ďakujem mojej rodine za trpezlivosť a za ich neustálu psychickú podporu.

Abstrakt

V tejto práci predstavujem algoritmus na detegovanie fontu, ktorý je plne automatizovaný a spadá do kategórie problému Visual Font Recognition, ktorý je veľmi zanedbavaný komunitou zaoberajúcou sa počítačovým videním. Dal by sa nazvať aj Canny-SURF Font Recognition program, kedže využíva dva hlavné algoritmy: Canny Edge Detection a Speeded-Up Robust Features na zisťovanie fontu. Bol testovaný na 40 snímkach obsahujúcich každá jeden font s textom, ktorý obsahoval len písmená z anglickej abecedy. Fonty boli hlavne zo sans a serif-sans rodín. Na testovacích dátach dosahoval úspešnosť detekcie od 85% až po 92.5% s priemernou úspešnosťou 89.16%.

Kľúčové slová: Visual Font Recognition Problem, Canny-SURF, Detekcia fontu

Abstract

In this work I present a font recognition program, which is fully automatic and is classified under widely neglected Visual Font Recognition problem by community working on visual recognition. Algorithm recognizing font can be marked as Canny-Edge based as it is using the Canny Edge Detection with the Speeded-Up Robust Features extraction. It was tested on dataset containing 40 images, each image had text containing only letters from the English alphabet in different fonts. These fonts were picked from mainly two families: sans and serif-sans. At the end the successful recognizing rate was ranging from 85% to 92.5% with average of 89.16%.

Keywords: Visual Font Recognition problem, Canny-SURF, Optical Font Recognition

Obsah

Zoznam obrázkov

Úvod

Cieľom tejto práce bolo vytvoriť automatizovaný systém na detegovanie fontu. Tento systém mal detegovať font na naskenovaných dokumentoch, ktoré mohli byť aj fotky zo sveta, nie len obrazy čistého textu, takže mal byť dostatočne robustný aby vedel text lokalizovať a vedel pracovať s textami v rôznych farbách na rôznych pozadiach a aj na nekvalitných snímkach.

Vývoj tohoto algoritmu spadá pod problém Visual Font Recognition, ktorý sa sústreďuje na detegovanie fontov na fotkách a je dosť zanedbávaný komunitou zaoberajúcou sa počítačovým videním. Detegovanie fontu na obrazoch zo sveta je náročné z hľadiska počtu faktorov, ktoré ho môžu ovplnyvniť. Medzi najhlavnejšie patria: kvalita snímky, povrch textu a nasvietenie textu. Tieto faktory môžu ovplyvniť správne lokalizovanie, určenie písmena, alebo samotné detegovanie príznakov pomocou ktorých sa fonty rozpoznávajú.

Aj keď v poslednej dobe nastal veľký progres v detegovaní písmen pomocou OCR, tieto algoritmy zvyčajne pracujú len na štandartných tvaroch písmen a nie sú veľmi efektívne na tých neštandartných, ktoré zvyčajne chceme zistiť. Preto, súčasné systémy (napr. `www.whatfontis.com`) spoliehajú na používateľa aby detegoval, prípadne potvrdil písmená na fotke. Dokonca aj so zásahom používateľa, tieto systémy nie sú dosť robustné na zmyslupné praktické použitie. Hlbšie skúmanie toho problému by mohlo prispieť aj k vylepšeniu OCR algoritmov, kedže tieto problémy sú si veľmi podobné.

Implementácia samotného algoritmu je popísaná v kapitole 2 a výsledky dosiahnuté na testovacích dátach v kapitole 3. V tejto kapitole sú popísané aj možné vylepšenia algoritmu. Nakoniec, dosiahnuté výsledky sú porovnané s vybranými systémami v kapitole 4.

Kapitola 1

Základné pojmy

Táto kapitola stručne uvedie problém známy ako Visual Font Recognition, definuje a vysvetlí základné pojmy používané pri detekcii písmen a rozpoznávaní fontu.

1.1 Visual Font Recognition

Na definovanie problému Visual Font Recognision sa najskôr musíme zoznámiť s problémom Optical Character Recognition (ďalej len `OCR`) a pojmom Font.

1.1.1 OCR

OCR je technológia umožňujúca konvertovanie tlačených, alebo písaných znakov v obraze do textovej, editovateľnej formy. Program konvertuje obraz automaticky, alebo sa musí naučiť rozpoznávať znaky. Používa sa hlavne na digitalizovanie dokumentov, aby sa v nich dalo lahšie vyhladávať, mohli byť upravené, digitálne uskladnené a používatelia si ich mohli pozrieť online.

1.1.2 Font

Pojem Font predstavuje špecifickú kombináciu vlastností písmena ako sú napríklad: sklon, hrúbka, šírka, pätky (serif) (obr. 1.1). Podobné vlastnosti sa zoskupujú do rodín, kde sa fonty líšia len drobnými zmenami. V počítači je najčastejšie font reprezentovaný súborom vo formáte TrueType Font, alebo OpenType Font.

TrueType Font (TTF) je formát, ktorý vznikol v roku 1991 vyvinutý firmou Apple Computer, ako jednoduchý formát pre operačný systém vyvíjaný touto firmou. Neskôr firma Apple Computer TTF formát zadarmo licencovala firme Microsoft[3]. Tento formát má výhodu v tom, ako presne vie vykresliť znak lubovoľnej veľkosti, v počítači je reprezentovaný príponou .ttf na operačnom systéme Microsoft Windows a .dfont na macOS.

Obr. 1.1: Pätka (serif), znázornený červenou farbou, je štrukturálny detail na konci ťahu, ktorý tvorí písmena a symboly.

OpenType Font (OTF) je formát vyvinutý firmami Microsoft a Adobe v roku 1997 ako vylepšenie formátu TTF a dnes je hlavným používaným formátom[2]. Tento formát ma výhodu v tom, že je medzi-platformový (macOS a Windows), má omnoho väčší limit znakov (až 64 tisíc), a taktiež prináša nové vlastnoti ako možno definovať font. V počítači je reprezentovaný súborom s príponou .otf.

1.1.3 Visual a Optical Font Recognition

Problém Optical Font Recognition je snaha automaticky určiť, čo najpresnejšie, font skenovaného dokumentu. Pri súčasnej existencií stoviek tisíc fontov táto úloha nie je vôbec triviálna.

Na druhú stranu Visual Font Recognition problém má za úlohu určiť čo najpresnejšie font na snímkach zo sveta [9]. Tým sa stáva oveľa zložitejším, kedže musí brať do úvahy aj faktory ako sú materiál a povrch písmen, nasvietenie, alebo kvalita snímky. Všetky tieto faktory ovplyvňujú úspešnosť s akou je font detegovaný.

1.2 Príznaky

Príznakom v oblasti počítačového videnia sa rozumie výsledok merania, ktoré kvantifikuje nejakú vlastnosť objektu. Príznaky sa reprezentujú pomocou príznakového vektora[12].

Príznaky vzhľadom na úroveň ich abstrakcie delíme na:

- **Nízkoúrovňové príznaky** opisujú základné vlastnosti objektov ako tvar, farbu alebo textúru atď.
- **Stredoúrovňové príznaky** vzniknú spojením nízkoúrovňových príznakov ako napríklad použitá paleta farieb, súbor zakladných tvarov v obraze a pod.
- **Vysokoúrovňové príznaky** sú príznaky najvyššej úrovne, ktoré popisujú sémantiku zobrazenej scény.

Nasledujúca časť definuje niekoľko základných pojmov z nízkoúrovňových príznakov.

Obr. 1.2: Snímka rodinného domu segmentovaná na rôzne oblasti.

1.3 Segmentácia a príznaky tvaru

Segmentácia je proces vedúci k rozdeleniu obrazu do jednotlivých, vzájomne disjunktných oblastí. Rozpoznané oblasti sú následne zatriedené do určitých disjunktých podmnožín - tried (obr. 1.2). V prípade, že je zaujímavá len jedna trieda, alebo jedna skupina tried, je výsledkom binárny obraz kde sú tieto triedy oddelené od pozadia, ktoré je reprezentované nulami. Oblasti sú segmentované podľa rôznych kriterií. Najčastejšie sú to podobné odtiene farieb, alebo textúry a ich hranice[12].

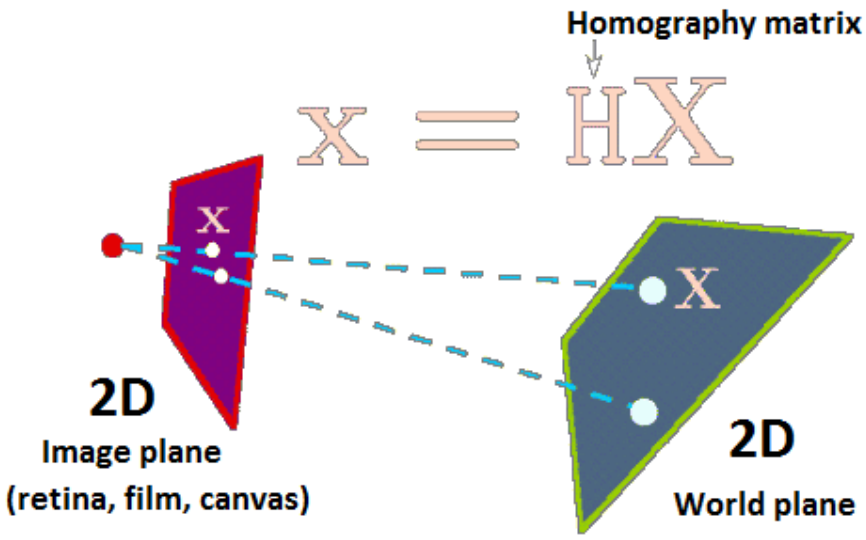
V týchto oblastiach sa následne detegujú príznaky (napr. tvaru, farby, textúry). Nás budú najviac zaujímať príznaky tvaru, ktoré sú:

- **Hrana** je súbor bodov, ktoré oddeľujú jeden objekt od druhého. Pod objektom si môžeme prestaviť napríklad okno na budove, oblak na oblohe, ale aj jeden odtieň farby v obraze.
- **Roh a zaujímavé body** sú body kde hrana výrazne mení smer. Tieto body môžu tiež popisovať zakrivenie hrany a smer jej zakrivenia.

1.4 Ostatné pojmy

Ostatné pojmy potrebné na porozumenie fungovania programu.

- `Keypoint` definuje zaujímavý bod v obraze (napr. hranu). Definuje bod, kde nezáleží na rotácí, veľkosti či posunu obrazu. To znamená, že algoritmy hľadajúce tieto body ich nájdu v obraze, ktorý mohol byť otočený, zmenšený/zväčšený, alebo posunutý.
- `Deskriptor` je vektor dĺžky 128 bitov definujúci konkrétny keypoint a jeho blízke okolie. Pomocou deskriptorov vieme keypointy porovnávať.



Obr. 1.3: Ukážka transformácie bodu x v rovine Image plane, na bod X v rovine World plane pomocou matice transformačnej matice H.

- `Matchovanie` keypointov je proces porovnávania a hľadania tých istých keypointov v dvoch rôznych obrazoch.
- `Homografia` v počítačovom videní definuje projekčnú transformáciu útvaru, ktorá priradí množine bodov ležiacej v jednej rovine, množinu bodov ležiacej v druhej rovine. Tieto body sú bodmi toho istého objektu v dvoch rôznych obrázkoch, ktoré sú zväčša dvoma rozličnými projekciami tohoto objektu. Výsledkom výpočtu je matica homografie (obr. 1.3). V našom prípade sú objekty písmená, definované množinou keypointov. Na výpočet používame metódu RANSAC (RANdom SAmple Consensus), ktorý nám keypointy rozdelí na tzv. *inliners* a *outliners*. Inliners sú body, ktoré „pasujú“do vypočítanej homografie a outliners sú všetky ostatné body.

Kapitola 2

Návrh a implementácia

Realizácia implementácie prebieha v týchto 3 nasledovných krokoch:

- nájdenie voľne dostupného OCR a algoritmu na predspracovanie obrazu
- nájdenie voľne dostupnej databázy pre testovacie dáta a databázu fontov
- implementácia algoritmu na rozpoznanie fontu

Program je implementovaný v jazyku C++, kompilovaný kompilátorom programu Visual Studio 2015. Všetky dôležité algoritmy, až na OCR, a taktiež všetky dátové typy a operácie týkajúce sa spracovania obrazu používa z knižnice OpenCV3.2 [6][10]. Nebol navrhnutý ako real-time program, takže sa moc nedbalo na optimalizácie. Celý program funguje nasledovne (obr. 2.1):

V prvej časti sa vstupný obraz nasegmentuje a oddelí sa text od pozadia, následne tieto oblasti s textom slúžia ako vstup pre OCR algoritmus, ktorý deteguje konkrétne slovo v oblasti a posunie tieto slová nášmu algoritmu.

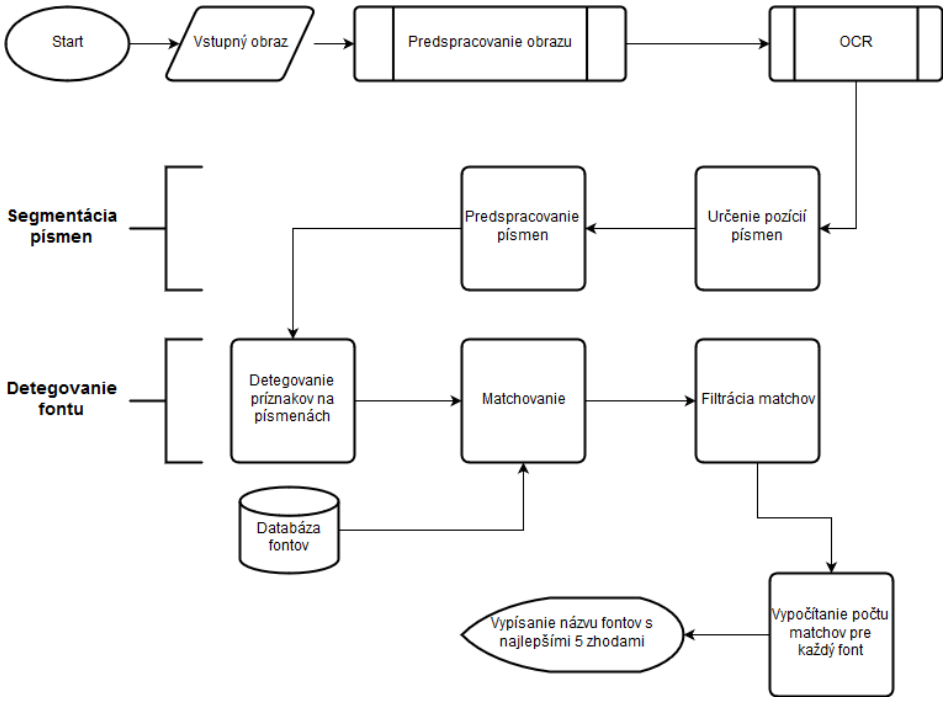
Druhá časť zahŕňa detegovanie príznakov jednotlivých písmen v slove pomocou nášho algoritmu a vyhodnotenie najlepších piatich zhôd s databázou fontov. Ako výstup algoritmu aj celého programu bude názov týchto top 5 fontov zoradených od najlepšej zhody.

2.1 OCR a algoritmus na predspracovanie obrazu

Kedže tieto algoritmy neboli hlavným cielom práce, požiadavkou bolo nájsť voľne dostupné a spoľahlivé algoritmy za cenu rýchlosti.

2.1.1 Predspracovanie obrazu

Na predspracovanie obrazu bol vybraný algoritmus z knižnice OpenCV, vyvinutý L. Neumannom a J. Matasom v roku 2012 [14], pre svoju robustnosť a fakt, že je súčasťou



Obr. 2.1: Vývojový diagram programu.

OpenCV. Algoritmus deteguje text zo setu External Regions (ERs), ktoré predstavujú skupiny s najpravdepodobnejším výskytom písmen. Detegovanie je rozdelené do dvoch fází.

V prvej fázi je každému regiónu priradená pravdepodobnosť, že tento región je písmeno, obsahuje alebo je časť písmena. Do druhej fázi sú vybrané len regióny s najväčšou pravdepodobnosťou, kde je na ne aplikovaný výpočtovo náročnejší algoritmus. Nakoniec sú tieto regióny spojené do skupín a skupiny s najväčšou pravdepodobnosťou výskytu písmen sú vybrané.

Tento algoritmus veľmi dobre funguje aj na nekvalitných alebo zle nasvietených snímkach. No výstupné skupiny obsahujúce písmená moc dobre nezachovávajú dôležité príznaky tvaru písmen a písmená sú väčšinou trochu deformované. Čo nevadí pokiaľ ide len o rozpoznanie písmen, ale vadí pri rozpoznaní fontu.

2.1.2 OCR

Pre OCR algoritmus bola požiadavka aby bol dostatočne výkonný, open-source a s čo najmenšou chybovosťou. Ako najlepší sa ukázal byť Tesseract vyvíjaný firmou HP naskôr ako PhD projekt v HP Labs, potom ako prototyp, a neskôr bol vydaný verejnosti [15]. Tesseract pracuje na skupinách, ktoré dostane od algoritmu na predspracovanie obrazu. Výstupom z tesseractu je slovo a percentuálna istota, že dané slovo je naozaj to, ktoré je v skupine. Všetky slová s istotou menšou ako 45% a slová kratšie ako 4 znaky a istotou menšou ako 68% sú ignorované.

2.2 Databázy

2.2.1 Databáza fontov

Databáza fontov musela obsahovať dostatočne širokú paletu fontov a obrazy písmen v príslušnom fonte. Taktiež musela obsahovať predpočítané dáta a názov fontu pre každé písmeno. Keďže nebolo možné nájsť takúto databázu, algoritmus, ktorý ju generoval musel byť vytvorený. Tento algoritmus berie ako vstup TrueType a OpenType fonty a vygeneruje zložky s názvom fontov. V tejto zložke je 125 súborov: 62 obrazov znakov, ktoré sú písmená (a-z, A-Z) a čísla (0-9), 62 xml súborov obsahujúcich dáta ku každému znaku a jeden textový súbor obsahujúci informáciu či tento font bol predpočítaný. Každý obraz znaku a k nemu prisluchajúci predpočítaný dátový súbor je pomenovaný číselnou reprezentáciou znaku v tabuľke ASCII. Súbory obsahujúce jednotlivé fonty sú uložené v samostatnej zložke _FONTS, textový názov fontov je uložený v textovom súbore list.txt v zložke _Config. Vždy keď sa pridá nový font, algoritmus zistí, či každý súbor v zložke _FONTS má meno zapísané v list.txt, takto nájde novo pridaný font, vytvorí preňho novú zložku a vygeneruje obrazy znakov a data.txt súbor.

Samotné predpočítavanie dát a generovanie xml prebieha v druhej časti tohoto algoritmu. V tejto časti algoritmus prebehne všetky zložky fontov a ak nájde v data.txt informáciu, že pre tento font neboli dáta vypočítané, spustí sa v tejto zložke. To znamená, pre každý obraz znaku vygeneruje xml súbor s dátami opisujúcimi tento znak (bližšie sú tieto dáta opísane v sekcii 2.3.2).

2.2.2 Testovacia databáza

Databáza na testovanie musela obsahovať obrazy z reálneho sveta, kde v každom obraze musel byť text v jednom fonte a ku každému obrazu priložený názov fontu. Problém ale bol nájsť takúto voľne dostupnú databázu, tak musel byť vytvorený algoritmus, ktorý ju generoval. Ako pozadie bol vybraný náhodný snímok, kde bol následne odstránený pôvodný text (obr. 2.2). Algoritmus načíta fonty z databázy fontov a pre každý font vygeneruje obraz s textom a textový súbor s názvom použitého fontu (obr. 2.3).

2.3 Implementácia algoritmu

Algoritmus sa skladá z niekoľko častí. Hlavnými sú predspracovanie obrazu, OCR, segmentovanie písmen a detekcia fontu. Na ukážku budeme pracovať s náhodným obrazom z testovacích dát (obr. 2.3).

Algoritmus na predspracovanie obrazu sme popísali v sekcii 2.1.1, jeho výstupom je vektor štvoruholníkov, ktoré definujú oblasti kde sa detegoval text. Text sa deteguje horizontálne, takže v prípade keby slovo HOTEL bolo napísané vertikálne, dostaneme

Obr. 2.2: Vľavo je náhodný obraz s pôvodným textom vybraný ako pozadie do testovacej databázy, vpravo obraz po vymazaní pôvodného textu.

Obr. 2.3: Príklad obrazu, ktorý vygeneruje algoritmus testovacej databázy. Pôvodný text bol nahradený textom "The Quack 10"vo fonte Bottle Party. Tento obraz budeme zároveň používať ako vstup pre náš algoritmus.

Obr. 2.4: Binárna matica prekreslená do obrazu, ktorú dostaneme po predspracovaní obrazu. Matica obsahuje dve slová.

vektor s piatimi štvoruholníkmi, kde v každom bude jedno písmeno. Pomocou tohoto vektora sa zo vstupného obrazu vyrežú časti, ktoré obsahujú text. Tieto časti sú uložené v binárnej matici, kde pozadie je reprezentované nulami a text jednotkami (obr. 2.4). Pojem matica v knižnici OpenCV znamená dátová štruktúra, ktorá reprezentuje obraz. Táto matica sa ďalej posunie OCR na detekciu textu.

OCR, popísané v kapitole 2.1.2, dostane ako vstup binárnu maticu z algoritmu na predspracovanie obrazu, ktorá sa nareže na jednotlivé slová a výstupom je vektor textov a vektor istôt. Pre lahšie manipulovanie s maticami a ich príslušnými vektormi som si vytvoril vektor štruktúr `vector<Mats> group_imgs`, ktorý sa napĺna vždy keď OCR nájde slovo s dostatočnou istotou.

```
1 struct Mats {
2         Mat group; // Image stored
3         string word; // Text detected on the image
4         vector<float> confidence;       // Confidence of the text
              detected
5         bool active = false;    // True if this object contains
              text
6         ...
7 };
```

Štruktúra Mats obsahuje premenné:

- `Mat group` je matica, ktorá obsahuje príslušný text zapísaný binárne, kde pozadie je reprezentované nulou a písmená jednotkou. Matica obsahuje všetky slová v jednom riadku.
- `string word` je text v matici, ktoré dostaneme z OCR. V prípade viac slovného textu, premenná obsahuje tieto slová za sebou, bez medzery.
- `vector<float> confidence` je vektor istôt. Každá položka vektora je istota pre jedno slovo v danom texte.
- `bool active` je premenná, ktorá sa využíva pri spracovaní štruktúry neskôr v programe. Hovorí o tom či daná matica obsahuje text s dostatočnou istotou, pretože algoritmus na predspracovanie obrazu môže vrátit aj matice bez textu, alebo OCR môže detegovať text aj tam kde v skutočnosti nie je.

Obr. 2.5: Obraz vystrihnutý presne podľa danej binárnej matice prekonvertovaný do odtieňov šedej.

2.3.1 Segmentácia písmen

V tejto časti algoritmu sa detegujú pozície písmen a písmená sa konvertujú do formátu, na ktorom sa jednoduchšie detegujú príznaky tvaru.

Vstup do tejto časti je `vector<Mats> group_imgs` a obraz vystrihnutý zo vstupného obrazu, tak aby reprezentoval presne tú istú časť, ktorú obsahuje príslušná matica jedného group_imgs (obr. 2.5), v odtieňoch šedej. Pre jednoduchšie pochopenie pomenujme si jeden prvok vektora group_imgs groupBI (BI pre binary image) a k nemu prislúchajúcu maticu vystrihnutú zo vstupného obrazu groupGI (GI pre greyscale image).

Najskôr sa odfiltrujú všetky falošné objekty vektora, to znamená ak `groupBI.active` je true a najvyššia istota v `groupBI.confidence` je väčšia ako 68% a priemerná istota je väčšia ako 60%, tak groupBI obsahuje text s dostatočnou istotou aby ho algoritmus mohol ďalej spracovávať. Výška matice (`groupBI.group` (obr. 2.4)) sa zmení na 350 pixelov a šírka sa prispôsobí, aby si matica zachovala stále rovnaký pomer. Táto zmena veľkosti je potrebná na zväčšenie presnosti detegovania príznakov. V takto zmenenej matici sa spustí algoritmus `findContours` [6], ktorý pre každé písmeno v matici vráti body, ktoré ohraničujú toto písmeno. Je dôležité spúšťať findContours na binárnej matici, pretože vtedy sme si istý že vrátené body budú naozaj ohraničovať konkrétne písmeno a nie nejaký náhodný zhluk podobných pixelov. Pomocou týchto bodov nájdeme presnú pozíciu písmen v binárnej matici.

Následne sa na rovnakú šírku a výšku zmení aj groupGI, a na groupGI sa spustí algoritmus `canny` [8], ktorý nájde hrany objektov (obr. 2.6). Dôvod prečo sa canny spúšťa na groupGI obraz a nie na groupBI.group je taký, že groupBI.group obsahuje zdeformovaný tvar písmen z algoritmu na predspracovanie obrazu. Túto deformáciu môžeme pozorovať, ak porovnáme písmená `e` a `Q` medzi obrázkami 2.4 a 2.5. Z matice, ktorú dostaneme z canny algoritmu, ďalej vystrihneme konktrétne písmená, kedže poznáme presnú pozíciu písmen v binárnej matici a obe matice majú rovnaké rozmery aj obsah, a uložíme ich samostatne do štruktúry Mats. Opäť pre jednoduchosť budeme ďalej pracovať len s jedným písmenom, nazvyme si ho `P1`.Ku každému písmenu je priradený aj jeho znak v ASCII tabuľke. Teda `P1.group` obsahuje obraz jeho obrysu a `P1.word` obsahuje jeho textovú podobu. Ostatné zložky štruktúry nás už nezaujímajú.

Na takýchto samostaných písmenách sa detegujú príznaky tvaru a vyhodnocuje sa

Obr. 2.6: Hranice písmen, ktoré sme dostali pomocou algoritmu canny.

font.

2.3.2 Detekcia fontu

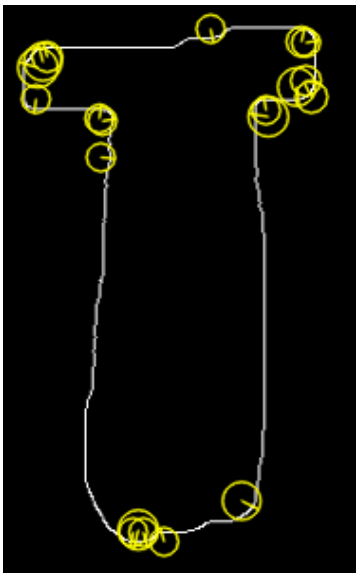
V tejto časti algoritmu sa na jednotlivých písmenách detegujú príznaky tvaru a vyhodnocuje sa font s najlepšou zhodou.

Na detegovanie príznakov používame funkciu Speeded-Up Robust Features (SURF) [5], ktorá k danému obrazu vráti vektor keypointov (obr. 2.7). SURF hľadá tzv. *b*loby - body, ktoré sú tmavšie, alebo svetlejšie ako ich okolie, presnejšie, determinant Hessovej matice druhých derivácií dosahuje v týchto bodoch maximum. Následne pre tieto body vypočíta deskriptory, ktoré popisujú ako sú distribuované intenzity pixelov v okolí keypointu. Tým, že detegujeme keypointy len na obrysoch písmen a nie na celých, pôvodných alebo binárnych tvaroch, zaručujeme, že sa SURF sústredí len na tvar písmena, nie na jeho hrúbku či nedokonalosti povrchu pri odfotení (čo môže byť napríklad špina na písmene, alebo nerovný povrch písmena). Keypointy každého písmena sa následne rozdelia na šesť častí pre presnejšie matchovanie (obr. 2.8). Ďalej sa z databázy fontov, z každého fontu, načítajú dáta konkrétneho písmena. Tieto dáta obsahujú keypointy už rozdelené do šiestich častí a taktiež deskriptory keypointov, detegované a vypočítané tým istým algoritmom.

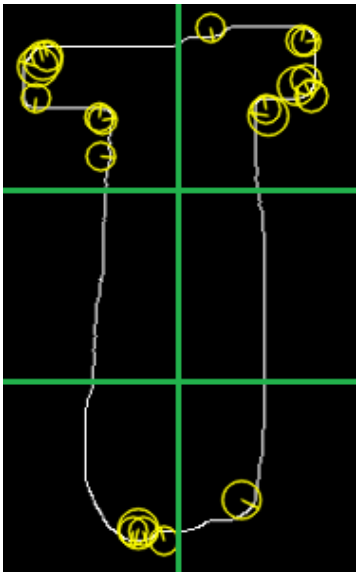
Samotné matchovanie deskriptorov keypointov zabezpečuje Brute-Force matcher[1] (obr. 2.9). Deskriptory sa porovnávajú zvolenou metrikou v 128 rozmernom priestore. Brute-Force matcher vráti dvojice keypointov, ktoré sú si najbližšie. Čo znamená, že vyfiltruje obe množiny a vráti nám len tie keypointy, pre ktoré existuje spojenie. No tieto spojenia stále nemusia byť korektné, veľa z nich je falošných (tz. nie sú paralelné s tými pravými), treba ich daľej filtrovať. Tvary, ktoré sa veľmi odlišujú od porovnávaného tvaru, vrátia viacej falošných spojení, ako viacej podobné tvary (obr. 2.9).

Filtrovanie falošných matchov prebieha pomocou funkcie `findHomography`, ktorá nájde transformáciu bodov jednej množiny podľa bodov druhej množiny (na obrázku 2.10 sú znázornené body a ich tranformácie spojené úsečkou). Taktiež identifikuje body, ktoré so svojimi projekciami spĺňajú transformáciu definovanú maticou homografie. Tým vieme odstrániť body, ktoré síce majú svoj pár, ale nie sú transformované určenou homografiou (obr. 2.11).

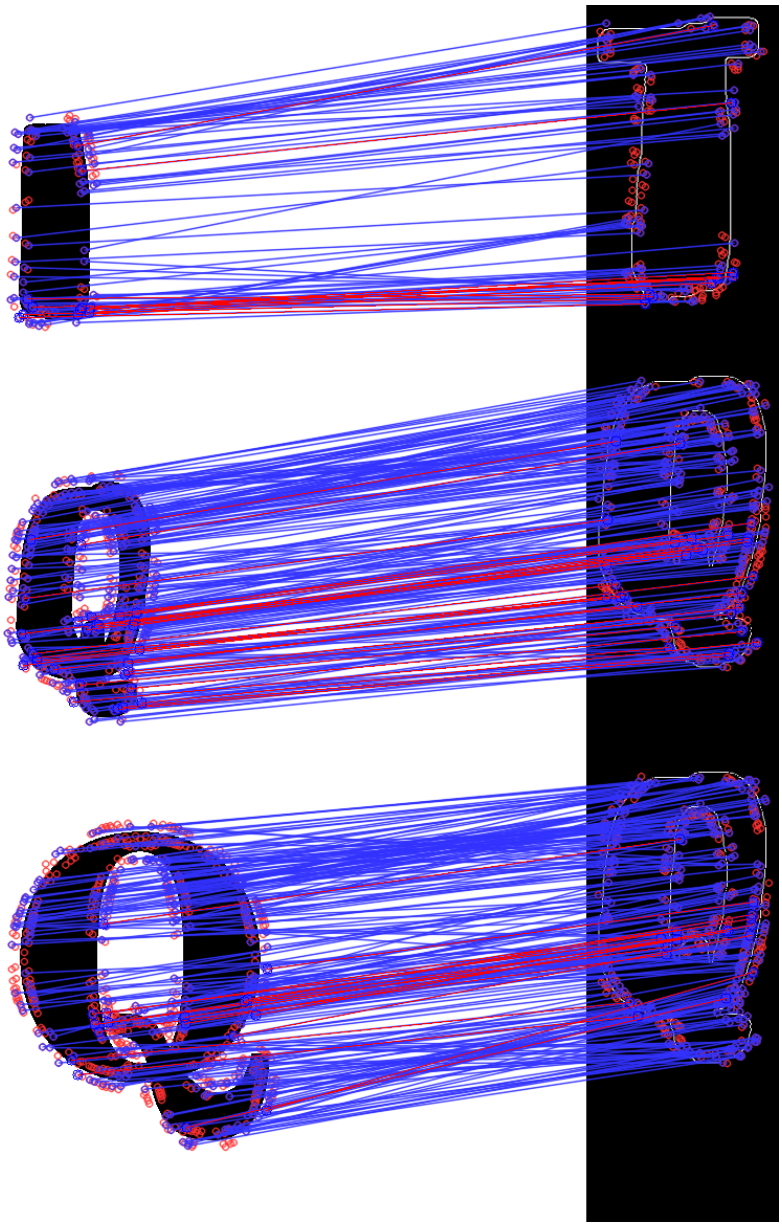
Nakoniec, každý font začína s počtom správnych keypointov rovných nule. Spočí-



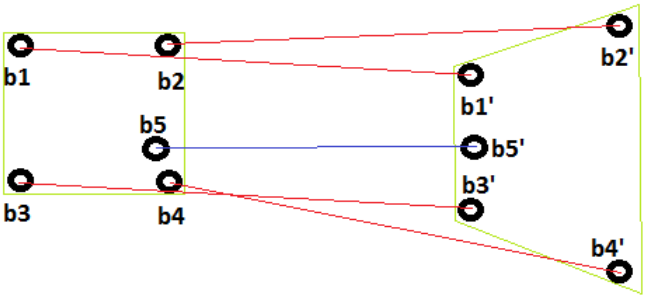
Obr. 2.7: Prvých 20 keypointov detegovaných na písmene T pomocou algoritmu SURF, polomer kruhu reprezentuje ich veľkosť a úsečka zo stredu k obvodu reprezentuje ich smer.



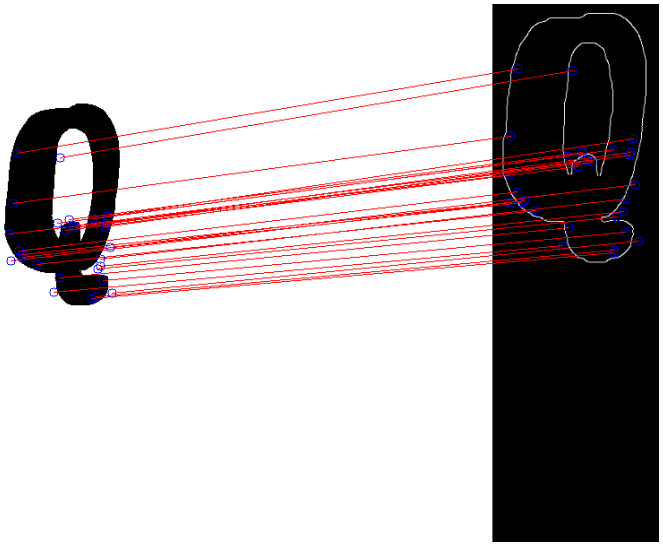
Obr. 2.8: Keypointy každého písmena sa rozdelia na šesť častí pre presnejšie matchovanie.



Obr. 2.9: Na obrázku môžme vidieť tri prípady matchovania Brute-Force funkciou. Správne spojenia sú znázornené červenou a falošné modrou. Z hora dole: V tomto prípade OCR zle rozoznalo 'T' a vyhodnotilo ho ako 'i', stále ale Brute-Force matcher našiel nejaké spoločné vlastnosti. V strede vidíme spojené 'Q' z databázy fontov (na ľavo) s 'Q' z testovacieho obrázku (na pravo) v rovnakom fonte. Môžeme vidieť, že veľa spojení je falošných, toto je spôsobené drobnými odchyľkami v obrysoch písmena. Na poslednom obrázku vidíme 'Q' z databázy fontov v inom fonte ako je to na pravo. Môžeme pozorovať menší výskyt správných spojení.



Obr. 2.10: Ukážka ako funguje findHomography. Ak by sme mali dve množiny bodov: B1 = {b1, b2, b3, b4, b5}, B2 = {b1', b2', b3', b4', b5'} a b1 by bolo spojené s b1', atď. findHomography by nám vrátila maticu, ktorá by transformovala zelený štvoruholník okolo množiny bodov B1 na zelený štvoruholník ohraničujúci množinu B2. Navyše by ignorovala spojenie b5-b5', pretože spojenie nevyhovuje tejto homografii.



Obr. 2.11: Príklad koľko nakoniec zostane keypointov po filtrácií funkciou findHomography. Na obrázku je písmeno 'Q' v rovnakom fonte na ľavo aj na pravo. Červenou sú znázornené spojenia vyhovujúce vypočítanej homografii.

tame koľko párov keypointov bolo transformovaných homografiou pre dané písmeno a pripočítame k počtu keypointov daného fontu. Keď spracujeme všetky písmená zoradíme fonty od najväčšieho počtu keypointov a vypíšeme najlepších 5 výsledokov.

Kapitola 3

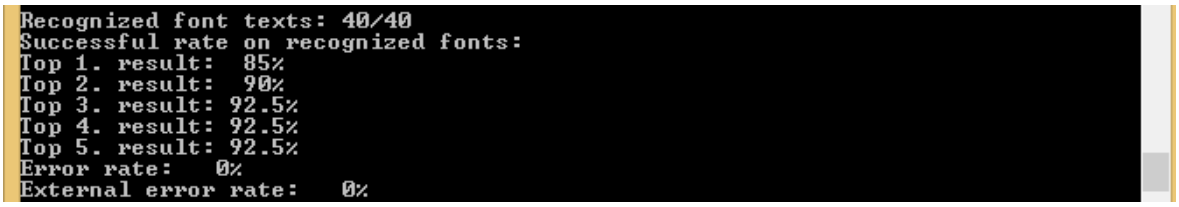
Vyhodnotenie

Nasledujúca kapitola popisuje aké výsledky sme dosiahli a aké možné vylepšenia by sme mohli aplikovať aby sa dosiahnuté výsledky zlepšili.

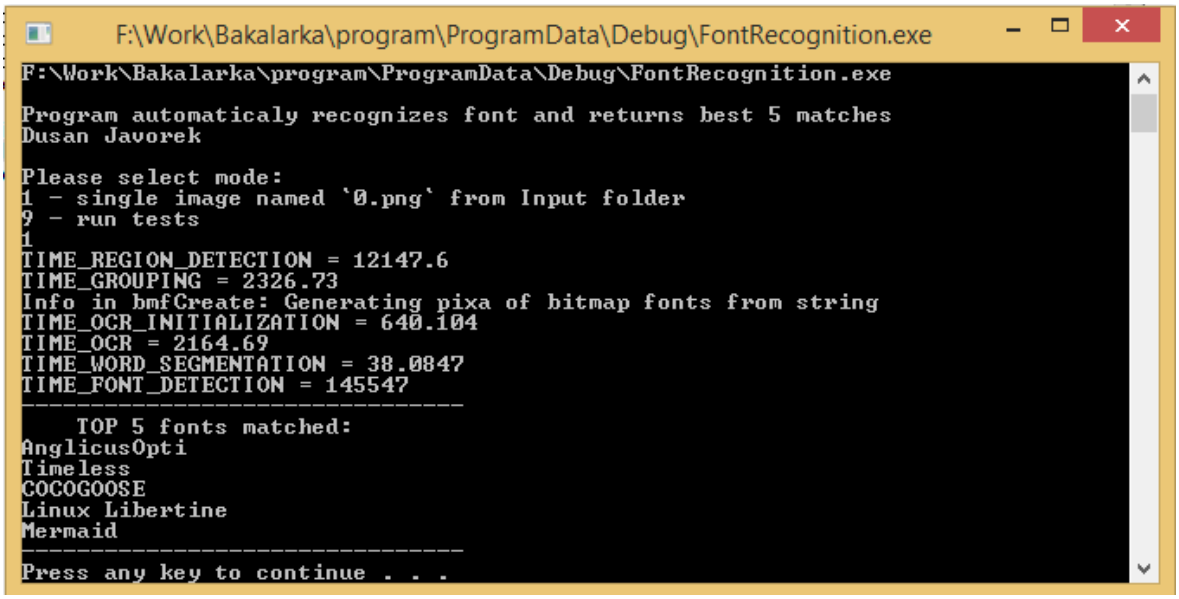
3.1 Testovanie a výsledky

Testovanie aplikácie prebiehalo porovnávacím algoritmom medzi názvom fontu, ktorý sme dostali z programu a názvom fontu v testovacej databáze. Aplikáciu možno prepnúť do testovacieho módu, ktorý automaticky zistí font na každej snímke z testovacej databázy popísanej v podkapitole 2.2.2, zapíše výsledok do textových súborov a vyhodnotí výsledok. Ak nastane chyba ešte pred krokom zisťovania fontu, program zapíše `SEGMENTATION_OCR_FAIL`, ak nastane chyba počas zisťovania fontu zapíše `NOT_ENOUGH_KEYPOINTS_FAIL` namiesto názvu fontu do výstupného súboru. Testovací algoritmus potom porovná dáta z výstupných súborov s dátami súborov v testovacej databáze a vypíše percentuálnu úspešnosť fontov, ktoré boli počítané, a percentuálnu chybu, kde program ako výstup vrátil `NOT_ENOUGH_KEYPOINTS_FAIL`. Výstupné dáta s reťazcom `SEGMENTATION_OCR_FAIL` nás nezaujímajú, pretože ide o chybu externého algoritmu, ktorú nemôžeme ovplyniť. Naviac testovací algoritmus vypíše aj úspešnosť s ktorou program, síce nevypočítal konkrétny font ako top 1, ale trafil sa do určitého miesta. To znamená, kedže program vypisuje najlepších 5 fontov, algoritmus vypíše percentuálnu úspešnosť toho, že sa správny font nachádza do konkrétneho miesta. Napríklad ak program vypočítal správne 15/20 fontov, jeho top 1 úspešnosť bude 75%, ak 2 z tých 5tich zle vypočítaných boli na 2. mieste vo výstupe, top 2 úspešnosť bude 85% (uhádol 17/20 do druhého miesta), atď.

Testovacie dáta pozostávali zo 40 fontov, obsahujúce niekoľko rodín, väčšinou ale sans a sans-serif, a niekoľko fontov v jednej rodine. Výsledok bol: náš program dokázal rozpoznať fonty s top 1 úspešnosťou 85% a percentuálnou chybou 0%, úspešnosť rástla až do top 3, kde dosiahla maximum 92.5% (obr. 3.1. Čo znamená, že prie-



Obr. 3.1: Výstup programu po zbehnutí testov. Celková úspešnosť dosiahla až 92.5%.



Obr. 3.2: Výsledok testovania náhodného obrazu prebehol úspešne. Top 1 vypísaný font sa zhoduje s fontom, ktorý zistila aj stránka whatfontis.

merná úspešnosť bola 89.16%. Všetky potiencionálne fonty, pre ktoré by program vrátil `SEGMENTATION_OCR_FAIL` boli z testovacej databázy odstránené. Väčšinou išlo o fonty, ktoré definovali písmo podobné písanému a teda písmená neboli štandartného tvaru.

Algorimus bol testovaný aj na dátach, ktoré neobsahovali text v presnom fonte z databázy a očakávala so, že algoritmus vypíše najbližší podobný font. Postup bol nasledovný: najskôr sa vybral náhodný obraz na testovanie a použila sa stránka whatfontis.com na určenie najbližsieho fontu. Táto stránka vypíše najlepších 100 výsledkov pre daný obraz. Z týchto 100 fontov sa vybral jeden, ktorý sa podobal najviac a pridal sa do databázy fontov. Následne bol spustený náš program s tým istým obrazom a očakávalo sa, že program vypíše práve novo stiahnutý font ako svoj top 1 výsledok (obr. 3.2). Výsledkom bolo, že program dokázal správne určiť najbližší podobný font v 5 z 5tich prípadoch.

3.2 Možné vylepšenia a optimalizácia

Program nebol navrhnutý pre real-time počítanie, takže neobsahuje veľa optimalizacií. Výpočet jednej snímky je závislý od počtu fontov v databáze, veľkosti snímky a množstva textu na nej. Vyhodnotenie jednoho písmena v priemere trvá 200 až 400 ms / font. To znamená, že pri 40tich fontoch vyhodnotenie jedného písmena trvá v priemere 12 sekúnd. Celý program na jednej testovacej snímke s 40 fontami (obr. 2.3) a s rozmermi 640*480 a počtu písmen maximálne 10, zbehne do 200 sekúnd. Kde segmentácia trvá do 20 sekúnd a vyhodnocovanie fontov do 160 sekúnd. Pri optimalizácii teda treba optimalizovať hlavne tieto dve časti programu.

Tiež treba spomenúť pamäťovú náročnosť programu. Každý font v databáze fontov obsahuje približne 30 MB dát, ak by sme mali vstupný obraz ktorý by obsahoval každé písmeno, do pamäte by sa muselo načítať 30mb / font. Čo už pri 200 fontov nie je moc reálne. Mohli by sme nastaviť algoritmus tak, aby počítal písmená pre jeden font a potom pamäť odalokoval, čo by ale znamenalo, že by sme viackrát pristupovali do disku a tým pádom ešte viac spomalili beh programu. Ďalšou možnosťou by bolo zmenšit objem dát v databáze, čo znamená zmenšit počet keypointov pre jednotlivé písmená, čo by mohlo viesť k viacej nepresným výsledkom.

Čo sa týka presnosti určovania fontu, program by sa dal ďalej vylepšiť tým, že k počtu spojených keypointov, ako meradlo presnosti, by sa pridal parameter, ktorý by popisoval rozdiel medzi tvarom písmena z databázy a jeho vypočítanou transformáciou. Čím menší, tým lepšie. Takže výsledný font by bol vybraný s najväčším počtom spojených keypointov a zároveň s najmenšou tranformáciou. Taktiež by vylepšilo presnosť natrénovať OCR na viacej dátach aby vedel rozpoznať aj neštandartné fonty. Na aplikáciu týchto vylepšení som bohužial nemal už dostatok času.

Kapitola 4

Existujúce implementácie

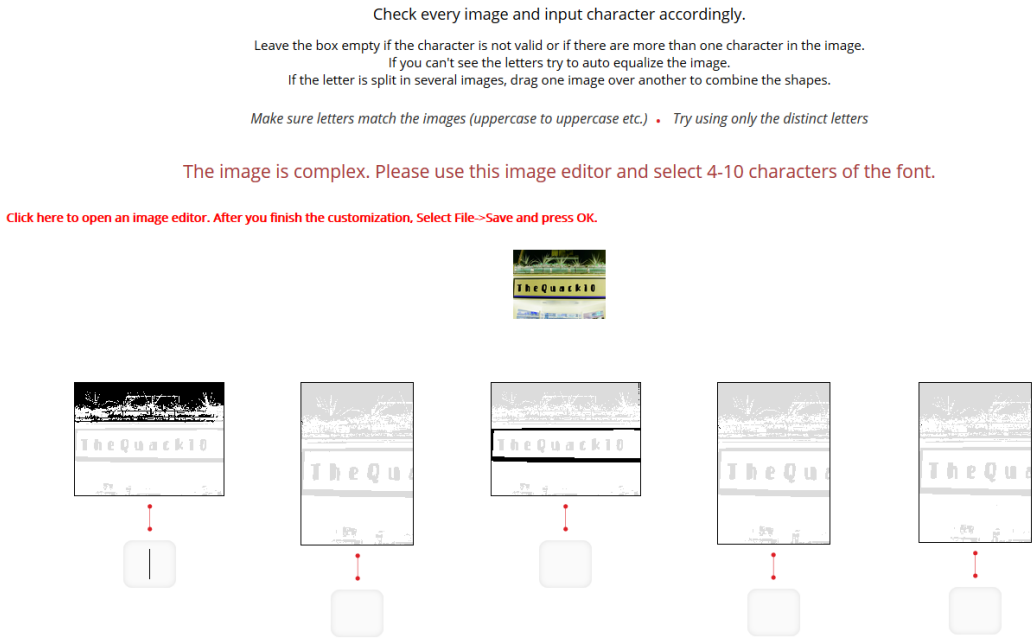
Táto kapitola poskytuje podrobnejší prehľad už existujúcich algoritmov na rozpoznanie fontu a ich porovnanie s mojou implementáciou.

Keďže požadavkou bol úplný automatizovaný program, porovnávanie so stránkami ako `www.whatfontis.com`, alebo `www.fontsquirrel.com` nie je veľmi relevantné, keďže tieto stránky nechávajú na užívateľa aby označil a detegoval text (obr.4.1), taktiež neposkytujú žiadnu dokumentáciu ich testovania. Viacej robustnejšie algoritmy sú ale väčšinou nedostupné verejnosti a jediná cesta, ako ich môžeme porovnať, je pomocou publikácií, v ktorých boli predstavené. Tieto algoritmy boli testované na vlastných databázach, odlišných od našej, no boli testované rovnakým spôsobom. Veľa algoritmov bolo vyvinutých na detegovanie fontov v dokumentoch, kde nie sú písmená zle nasvietené, alebo nekvalitné.

Jeden z algoritmov na detegovanie fontu na snímkach zo sveta je popísaný v dokumente Large-Scale Visual Font Recognition [9], vyvinutý univerzitou v Missouri pod záštitou Adobe v roku 2014. Tento algoritmus bol navrhnutý aby detegoval aj tie najemnejšie detaily medzi fontami pomocou upraveného SIFT algoritmu. Ich databáza fontov obsahovala 2420 rôznych variácií fontov zo 447 rodín.Testovanie prebiehalo na databáze, ktorá obsahovala obrazy 1000 najpoužívanejších anglických slov v každej variácii fontu a taktiež reálne snímky pozbierané z rôznych fór. Avšak všetky tieto testovacie obrazy boli orezané aby obsahovali len text, ktorý bude detegovaný. Výsledkom bolo, že na top 1 úspešnosť bola 72.50%, a vyšplhala sa až na 96.87% v top 10. Ich top 5 úspešnosť bola 93.45%, nás vedie k priemernej 87.61% úspešnosti. V porovnaní s našim, ktorý mal priemernú úspešnosť 89.16% no na veľmi malej databáze fontov oproti tomuto algoritmu.

Ďalšie zaujímavé algoritmy detegujú font zo skenovaných dokumentov. Za zmienku stoja:

Algoritmus popísaný v dokumente FONT FINDER: VISUAL RECOGNITION OF TYPEFACE IN PRINTED DOCUMENTS [7], vyvinutý univerzitou v Surrey v roku



Obr. 4.1: Stránka www.whatfontis.com necháva na užívateľovi detegovanie písmen, čo napomáha ku väčšej úspešnosti, no neprináša plnú automatizáciu. Po nahratí obrázku na stránku, nám ponúkne zoznam vysegmentovaných oblastí. Užívateľ musí oblasti prejsť a do políčok pod oblasťou zadať znak písmena, ak oblasť predstavuje písmeno.

2015, vo Veľkej Británii. Tento algoritmus nepotrebuje ručnú lokalizáciu textu, podobne ako náš program. Na matchovanie používa Histogram of Oriented Gradients (HOG) príznaky [11]. Ich databáza fontov obsahovala 1000 rôznych fontov a testovacia databáza náhodnú podmnožinu množiny, ktorá pozostávala z päť ukážok každého veľkého písmena v každom fonte. Ich priemerná úspešnosť bola 93.4%.

A algoritmus na detegovanie fontu na Farsi (perzských) dokumentoch popísaný v dokumente Farsi font recognition based on Sobel–Roberts features [13]. Na matchovanie používali vlastné príznaky, ktoré boli extrahované z textúr, ktoré dostali pomocou Sobel Edge detector a Robers Cross detector algoritmov. Tieto príznaky popisujú horizontálne úseky textu v jednom riadku, takže nepoužívajú klasický prístup matchovania písmeno-písmeno. Ich databáza fontov obsahovala 10 fontov a testovacia databáza 5000 rôznych dokumentov v týchto 10 fontoch. Ich priemerná úspešnosť bola 94.16%.

Záver

V tejto práci sme predstavili program na detegovanie fontu na skenovaných dokumentoch a fotkách. Tento program rieši problém Visual Font Recognition zanedbávaný komunitou zaoberajúcou sa počitačovým videním. Program, na detegovanie fontu, používa algoritmy Canny Edge detection a Speeded-Up Robust Features. Bol testovaný na testovacích dátach, ktoré obsahovali texty v 40 fontoch z hlavne dvoch rodín sans a serif-sans. Všetky texty použité na testovanie sa skladali len z písmen anglickej abecedy a čísel 0-9.

Cieľom tejto práce bolo úspešne rozpoznať font na naskenovanom dokumente a porovnať ho s vybranými aplikáciami. Na testovacích dátach program dosahoval úspešnosť medzi 85% až 92.5% s priemernou úspešnosťou 89.16%. Bol porovnaný s algoritmom vyvinutým univerzitou v Missouri, ktorý sa priamo zoberal problémom VFR a dvomi vybranými algoritmami, ktoré sa zaoberali problémom OFR. V porovnaní s algoritmami, ktoré riešili problém OFR si náš algoritmus viedol horšie v priemere o 4% a v prípade algoritmu vyvinutého na univerzite v Missouri si viedol o 1.5% lepšie. Treba ale spomenúť, že naše testovacie dáta boli rádovo menšie a je teda možné, že po dôkladnejšom testovaní naša priemerná úspešnosť môže klesnúť.

Program by sa do budúcna dal vylepšiť aby dosahoval väčšiu úspešnosť a taktiež by sa dal optimalizovať či už po časovej, alebo po pamäťovej stránke.

Literatúra

[1] Common interfaces of descriptor matchers. `http://docs.opencv.org/3.0-beta/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html`. Accessed: 2017-05-10.

[2] Opentype font. `https://www.microsoft.com/en-us/Typography/WhatIsOpenType.aspx`. Accessed: 2017-05-07.

[3] Truetype font history. `https://www.microsoft.com/typography/TrueTypeHistory.mspx`. Accessed: 2017-05-07.

[4] Michael Angelov. *Rozšířená realita v reklamě*. PhD thesis, FIT VUT v Brně, 2011.

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[6] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. Ö'Reilly Media, Inc.", 2008.

[7] Tu Bui and John Collomosse. Font finder: Visual recognition of typeface in printed documents. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 3926–3930. IEEE, 2015.

[8] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[9] Guang Chen, Jianchao Yang, Hailin Jin, Jonathan Brandt, Eli Shechtman, Aseem Agarwala, and Tony X Han. Large-scale visual font recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3598–3605. IEEE, 2014.

[10] Ivan Culjak, David Abram, Tomislav Pribanic, Hrvoje Dzapo, and Mario Cifrek. A brief introduction to opencv. In *MIPRO, 2012 proceedings of the 35th international convention*, pages 1725–1730. IEEE, 2012.

[11] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[12] Elena Šikudová, Zuzana Černeková, Vanda Benešová, Zuzana Haladová, and Júlia Kučerová. *Počítačové videnie. Detekcia a rozpoznávanie objektov*. Wikina, Praha, 2013.

[13] Hossein Khosravi and Ehsanollah Kabir. Farsi font recognition based on sobel–roberts features. *Pattern Recognition Letters*, 31(1):75–82, 2010.

[14] Lukáš Neumann and Jiří Matas. Real-time scene text localization and recognition. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3538–3545. IEEE, 2012.

[15] Ray Smith. An overview of the tesseract ocr engine. 2007. `http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf`.

Príloha

Súčasťou práce je CD s elektronickou prílohou obsahujúcou zdrojové súbory programu a tiež skompilovanú, spustiteľnú verziu určenú na testovanie. Súčasťou spustiteľnej verzie je aj databáza fontov spoločne s testovacou databázou. Zdrojové súbory sa dajú nájsť aj v repozitári na GitHube na adrese `https://github.com/jav0/bakalarska_praca`.