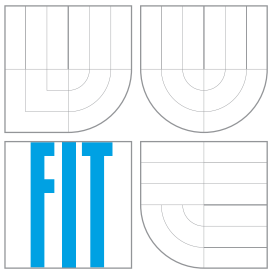
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



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

DETEKCE FORMULÁŘOVÝCH POLÍ VE SKENOVANÝCH DOKUMENTECH

FORM INPUT FILEDS DETECTION IN SCANNED DOCUMENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL MORAVEC

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÍTĚZSLAV BERAN, Ph.D.

BRNO 2016

Zadání bakalářské práce/18161/2015/xmorav28

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Moravec Michal**
Obor: Informační technologie
Téma: **Detekce formulářových polí ve skenovaných dokumentech**
Form Input Fileds Detection in Scanned Documents
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte metody pro segmentaci obrazů, zvýraznění hran a detekci čar, křivek, obdélníků apod. Seznamte se statistickými metodami vhodnými pro shlukování dat.
2. Vyberte vhodné metody a navrhněte vlastní algoritmus, který v šedotónovém obrazu skenovaného dokumentu nalezne pozice formulářových polí a provede korekci případně rotovaného dokumentu.
3. Připravte a anotujte datovou sadu pro vývoj a testování řešení.
4. Implementujte navržený algoritmus s využitím vhodných knihoven.
5. Proveďte testy na přesnost, rychlost a využitelnost řešení. Výsledky diskutujte.
6. Vytvořte plakát a krátké demonstrační video reprezentující Vaše řešení.

Literatura:

- M. Sonka, V. Hlaváč, R. Boyle: Image Processing, Analysis, and Machine Vision, CL-Engineering, ISBN-13: 978-0495082521, 2007.
- Gary R. Bradski, Adrian Kaehler. Learning OpenCV: Computer Vision with the OpenCV Library. ISBN 10: 0-596-51613-4, September 2008.
- Dále dle pokynu vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2, 3 a částečně bod 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Beran Vítězslav, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem práce bylo navrhnout algoritmus, který bude schopen z obrazu naskenované karty zdravotní pojišťovny vyseparovat pouze textová pole, která se budou dát dále použít v libovolném softwaru na převedení obrázku na text. Program by měl počítat se špatně naskenovanými a libovolně otočenými kartami. Celkový projekt je dělán jako zakázka pro firmu Medingo, která chce algoritmus zakomponovat do svého stávajícího systému. Co se týče výsledků, tak algoritmus dokáže detekovat a vyseparovat textová pole s velmi vysokou pravděpodobností.

Abstract

Main goal of the work was to implement an algorithm, which is able to separate input fields from card of the health insurance company, which are then used in some optical character recognition (OCR) software to convert it to text. The program should also work on incorrectly scanned and arbitrarily rotated cards. The whole project was made as a contract for the Medingo company, which will be able to implement the algorithm in their existing system. As for the results, the program is able to detect and separate input fields from cards with very high probability.

Klíčová slova

Počítačové vidění, Zpracování obrazu, Karta zdravotní pojišťovny, OpenCV, Detekce formulářových polí, Python

Keywords

Computer vision, Image processing, Health insurance company card, OpenCV, Form input fields detection, Python

Citace

MORAVEC, Michal. *Detekce formulářových polí ve skenovaných dokumentech*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Beran Vítězslav.

Detekce formulářových polí ve skenovaných dokumentech

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Moravec
17. května 2016

Poděkování

Chtěl bych tímto poděkovat svému vedoucímu, Ing. Vítězslavu Beranovi, Ph.D., díky kterému jsem práci úspěšně dokončil.

Obsah

Kapitola 1

Úvod

Hlavním důvodem pro vytvoření tohoto projektu je **zlevnění pracovní síly a automatizace zpracování skenovaných dokumentů**. Úkolem bylo navrhnout algoritmus, kde je vstupem **karta zdravotní pojišťovny**, která může být libovolně otočená, a výstupem jsou **vyfiltrovaná jednotlivá textová pole**, které tato karta obsahuje. Vstupní karta je načtena na skenovacím zařízení, ve kterém se může během skenování pohnout, což způsobí poškození, se kterým musí algoritmus také počítat. Dále musí počítat s případným otočením, když je karta vložena do skenovacího zařízení obráceně.

V kapitole 2 bude řešení popsáno z teoretického hlediska. Budou zde vysvětleny jednotlivé funkce zpracování obrazu, které jsou v projektu použity, ale také budou ukázána již stávající řešení, která v této oblasti existují. V kapitole 3 bude popsáno řešení opět teoreticky, tentokrát však už konkrétní funkcionalita. Nejprve bude provedena analýza, kde budou popsány jednotlivé problémy, které mohou nastat. Dále se bude pokračovat popisem postupu řešení, které vede k samotnému výsledku. Konkrétní ukázky z implementace budou ukázány v kapitole 4, kde se čtenář může dozvědět více o řešení, které je jinak teoreticky popsáno v předchozí části. Dále zde budou ukázány experimenty, které byly provedeny pro ověření funkčnosti, ale také testy, které byly prováděny během implementace.

Kapitola 2

Teorie

V této části budou čtenáři představena stávající řešení, která řeší podobný problém, a dále teorie, která je potřebná k pochopení řešení projektu. U teoretických informací je vše názorně předvedeno použitím operací na vzorové kartě ZP. Většinu informací jsem čerpal z konzultací s vedoucím, které jsem pak dále studoval na internetu. Na závěr této kapitoly budou uvedeny jednotlivé nástroje a knihovny pro zpracování obrazu.

2.1 Existující nástroje a systémy pro analýzu skenovaných dokumentů

Jelikož je zadání velmi specifické, tak k tomuto problému se nedalo najít přesné řešení, které by ho bylo schopné pokrýt. To ale neznamená, že neexistují algoritmy a programy, které řeší tento problém na více abstraktní vrstvě a ne tolik specificky. Jelikož do zadání již nepatří následné zpracování a konverze textu z obrazu pomocí OCR softwaru, nebudou zde takováto řešení popisována.

První prací, kterou jsem našel, je **Text Area Detection in Digital Documents Images Using Textural Features** (Detekce textových oblastí v digitálních dokumentech pomocí texturních rysů) [1]. Zde je popsána metoda, pomocí které jsou autoři schopni detekovat text např. na oskenovaném papíře, kde se ale dále nacházejí i netextové komponenty jako obrázky a čáry. Tohoto jsou schopni pomocí **Gaborova filtru**, pomocí kterého odhalí v obrázku vysokofrekvenční oblasti. Předpokládají, že v těchto oblastech se nachází text, na který se dále aplikuje operace prahování pomocí **Otsuovy binarizace**, která v dané oblasti detekuje ideální práh a podle něho provede prahování.

Jelikož v mé práci očekávám přibližně stejný formát všech karet, nemusím řešení dělat příliš obecně, narozdíl od výše zmíněné metody. Dále mohu počítat s tím, že v jednotlivých textových polích není text na více řádků, což metodu dále usnadňuje. Výše uvedené řešení je tedy použitelné, ale zbytečně složité.

Dalším problémem je poškození vlivem špatného naskenování karty. Zde bych zmínil článek **Text Alignment with Handwritten Documents** (Zarovnání textu v ručně psaných dokumentech) [6]. Zde autoři popisují metodu, pomocí které jsou schopni rukou napsaný text na naskenovaném papíře, kde však nejsou jednotlivá slova zarovnána přesně, zarovnat pro další použití.

Jelikož se na kartě ZP nevyskytuje rukou psaný text, který je třeba detekovat, nelze tuto metodu využít pro tento účel. Je tedy třeba navrhnout metodu, pomocí které se posunutý, špatně naskenovaný text zarovná. O této metodě se zmíním později.

Obrázek 2.1: Ukázka metody detekce textu pomocí metody MSER.

Co se týče detekce textových oblastí v obrázcích, existuje také řešení, ve kterém autoři detekují tyto oblasti přes metodu **maximálně stabilních extrémních oblastí (Maximally stable extremal regions = MSER)** [3]. Tato metoda se používá v algoritmech na rozpoznávání objektů, protože dokáže z obrázku vyseparovat oblasti, které jsou si podobné svými vlastnostmi. Původně byla navržena pro porovnávání dvou obrázků a nalezení podobností, i přes jejich možné posunutí nebo i jiný úhel pohledu v rámci jednoho obrázku k druhému.

V článku autoři uvádějí, že metodu MSER využívají k nalezení **kandidátů**, kteří odpovídají jednotlivým písmenům. Tito kandidáti jsou dále filtrováni na základě geometrických vlastností a šířky okraje, aby byli odstraněni ti, kteří neodpovídají písmenům. Jednotlivá písmena jsou dále propojena pomocí přímky do jednotlivých slov. Ukázka této metody je na Obrázku 2.1, kde na prvním obrázku je ukázka po aplikaci metody MSER, na druhém obrázku jsou je pokus o filtraci textu a jeho následné propojení do slov a na třetím obrázku je vidět obrázek s detekovanou textovou oblastí.

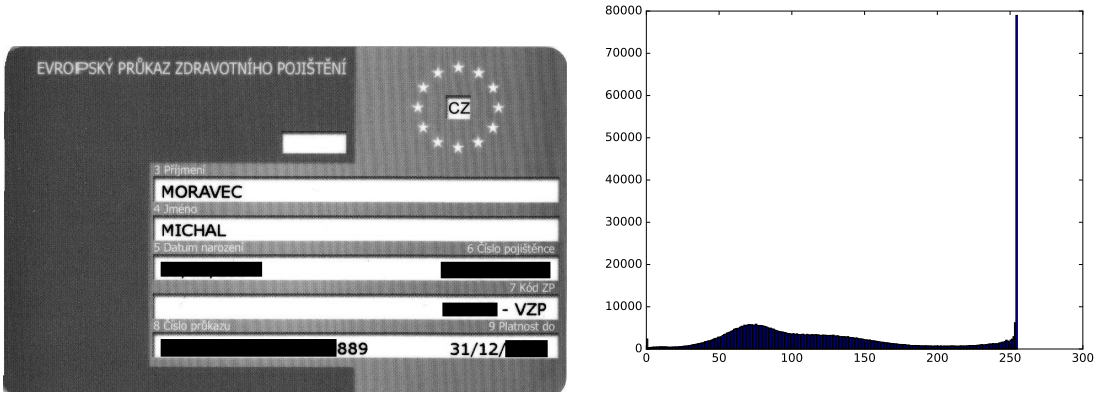
2.2 Histogram

Histogram je grafická reprezentace rozložení číselných dat (viz Obrázek 2.4). Konkrétně v oboru zpracování obrazu se používá pro **znázornění četnosti jednotlivých barevných odstínů v obrázku**. Když chceme sestavit histogram k libovolnému obrázku, tak ho nejprve rozdělíme na jednotlivé **třídy**. Tyto třídy představují stejně velké **intervaly**, do kterých musí spadat všechny jednotlivé body, které se na obrázku nacházejí. Z výsledné funkce můžeme vyčíst vlastnosti, které z obrázku nejdou na první pohled vidět, a to např. kontrast, poměr tmavých a světlých barev, nebo vyvážení jednotlivých barev.

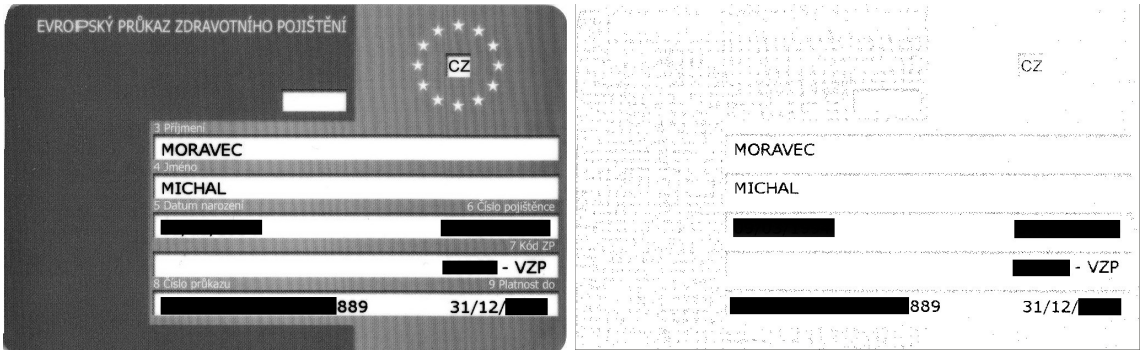
Přímo v projektu je histogram využit jako model pro detekci oblastí obrázku, na kterých se vyskytuje buď **pozadí** (bílé textové pole) a nebo **popředí** (černý text). Tyto oblasti jsou na histogramu obsaženy jako lokální maximum, ať už na levé straně (černý text) a nebo na pravé (bílé pozadí).

2.3 Prahování

Jedná se o jednoduchou operaci, která převede všechny body v obrázku jen na bílé a černé. Toto rozdělení se provede pomocí **prahu**, což je číselná hodnota odstínu barvy (zpravidla od 0 do 255, kde 0 je úplně černá barva a 255 úplně bílá), od které budou body v obrázku bílé, a do které budou jen černé 2.3. Toto prahování se nazývá jednoduché, existují však také složitější varianty, jako např. **adaptivní prahování** [2].



Obrázek 2.2: Ukázka histogramu karty vlevo.



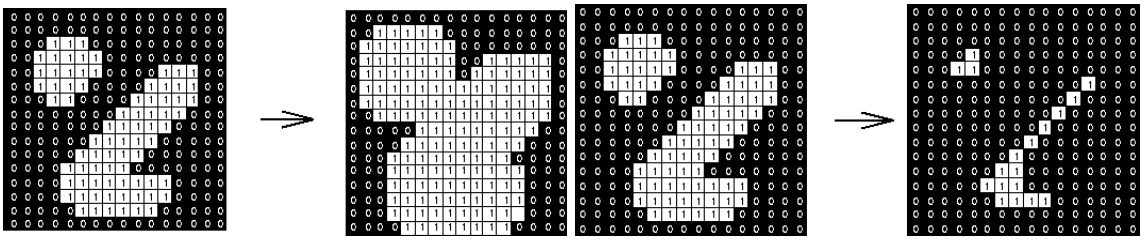
Obrázek 2.3: Ukázka prahování karty vlevo s hodnotou prahu 24.

Adaptivní prahování je velmi dobře využitelné v obrázku, který má velké rozdíly v kontrastu. Metoda totiž nepoužívá globální práh pro celý obrázek, ale vypočítá si ho pro každou malou oblast v obrázku zvlášť. Tímto se dá zajistit, že když je v obrázku oblast s úplně černým textem a úplně bílým pozadím, a vedle se nachází text tmavě šedý a pozadí světle šedé, v obou případech algoritmus zajistí správný práh, čímž se text stane černým a pozadí bílé. Další věcí, kterou je důležité zmínit, je jak se provede výpočet prahu z dané oblasti. Jednou z metod je výpočet průměrné hodnoty barvy všech bodů v daném okolí, druhou metodou je výpočet váženého součtu hodnot v okolí, kde váhy jsou vypočítány z úseku (okna) Gaussovy funkce.

Poslední metoda, o které se zmíním, je **Otsuova metoda** pro automatické prahování. Tato metoda funguje nejlépe v bimodálním obrázku, což je obrázek, který má v histogramu dvě lokální maxima. Práh se vypočítá tak, že se vezme polovina vzdálenosti mezi těmito maximy. Jedná se tedy čistě o jednoduché prahování, kde se ale hodnota prahu vypočítá dynamicky podle histogramu.

2.4 Kontury

Kontura, nebo také **obrys**, se dá popsat jako křivka, která je tvořena body, které ji spojují na jejím okraji [7]. Tato křivka má také vždy stejnou barvu. Využití má v detekci objektů, jejichž okraj je spojitá čára, jako je např. textové pole karet.



Obrázek 2.4: Ukázka morfologické operace dilatace (vlevo) a eroze (vpravo) se strukturním elementem 3×3 [4].

Před samotnou detekcí kontur je třeba obrázek převést na **binární** (obsahující jen černé a bílé body) pomocí prahování nebo libovolného filtru, který detekuje hrany, aby byl výsledek co nejpřesnější.

Výhoda tohoto přístupu spočívá v tom, že jednotlivé kontury mají určité vlastnosti jako obsah a poměr stran, které se pak dají dále využít pro jejich zpracování a filtraci. Detekované kontury můžeme rozdělit podle jejich tvaru na 2 druhy: **konvexní a konkávní**. Když je kontura **konvexní**, tak musí dle definice konvexnosti splňovat pravidlo, že když spojíme libovolné 2 body z celkové množiny bodů definující konturu, tak úsečka mezi těmito dvěma body musí ležet v daném útvaru. **Konkávní** útvar tuto vlastnost nesplňuje.

2.5 Morfologické operace

Morfologické operace jsou používány pro odstranění šumu, spojení, či zjednodušení tvarů objektů [5]. Fungují tak, že se na každý bod vstupního binárního obrázku aplikuje zpravidla menší množina bodů, tzv. **strukturní element**, který je také binárním obrázkem. Strukturní element se využije pro definici okolí, které se následně projde a tím se ovlivní původní procházený bod. Tato operace vytvoří nový obrázek, který se liší od původního vstupního. Strukturním elementem bývá zpravidla čtverec, kruh, nebo obdélník o délce hrany/průměru 3 body. Nejjednoduššími morfologickými operacemi jsou eroze a dilatace.

Operace **dilatace** funguje tak, že projde každý bod ve vstupním obrázku, a podívá se na všechny body v okolí, které je definováno strukturním elementem. Jestliže v tomto okolí najde bílý bod, nastaví původní bod také na bílou barvu. Jestliže se v okolí nenachází žádný bílý bod, původní bod není změněn. Kdybychom neoperovali nad binárním obrázkem, tak se v okolí hledá bod s maximální hodnotou odstínu barvy (u binárního obrázku je bílý). Provádí se tedy operace vektorového součtu množin vstupního obrázku A a strukturního elementu S:

$$A \oplus S = \{p \in \mathbb{E}^2 : p = a + s, \forall a \in A, \forall s \in S\}$$

Druhou základní operací je **eroze**, která funguje obdobně jako dilatace. V okolí pomocí strukturního elementu však hledá alespoň jeden černý bod, díky kterému nastaví původní procházený bod na černou. Obecně metoda hledá bod s minimální hodnotou odstínu barvy. Co se týče množinových operací, provádí se operace vektorového rozdílu:

$$A \ominus S = \{p \in \mathbb{E}^2 : p = a + s \in A \; pro \; \forall s \in S\}$$

Když tyto základní operace použijeme společně v kombinaci, vzniknou tím operace **otevření a uzavření**. Operace otevření vznikne nejprve aplikací eroze, a poté dilatace

takto:

$$A \circ B = (A \ominus B) \oplus B$$

Obdobně operace uzavření vznikne aplikací dilatace, a poté eroze:

$$A \bullet B = (A \oplus B) \ominus B$$

Aplikací těchto složitějších operací lze objekty zjednodušovat co se týče jejich okrajů nebo vnitřních elementů, každopádně celková velikost zůstává zpravidla zachována, narozdíl od jednoduchých operací, které výsledný útvar buď zmenší nebo zvětší.

2.6 Houghova transformace

Jedná se o metodu, která je schopna nalézt objekty v obrázku a **parametricky** je popsat [8]. Obecně je využitelná na různé geometrické útvary (kružnice, elipsy), hlavní využití v tomto projektu je ale pro vyhledání a popis **přímek**. V tomto řešení se hodí zvlášť proto, že je schopna detekovat přímky, které nejsou úplně rovné nebo jsou zakřivené vlivem špatného naskenování.

Houghova transformace používá k popisu přímky tuto rovnici:

$$x * cos(\theta) + y * sin(\theta) = r$$

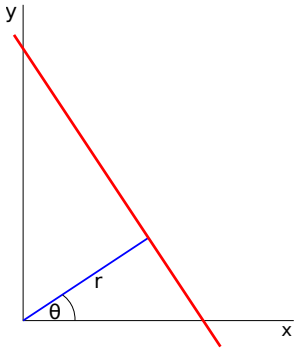
Přímka je tedy popsána v **polárních souřadnicích**. Pro lepší vysvětlení je tuto rovnici lepší popsat obrázkem 2.5. Samotná přímka je vyznačena červenou čarou. Modrou čarou je vyznačena normála, která prochází počátkem soustavy souřadnic. Délka této normály je dána proměnnou r, která určuje vzdálenost přímky od počátku. Úhel θ je sklon mezi osou x a normálou, který však také určuje i sklon samotné přímky.

Když detekujeme přímky z obrázku, tak parametry r a θ nejsou známé. Naopak známé jsou parametry x a y. Když je dosadíme do výše uvedené rovnice, dostaneme množinu přímek reprezentovaných v polárních souřadnicích, které procházejí tímto bodem. Když z této funkce vykreslíme graf, kde na jedné ose je parametr r a na druhé parametr θ, dostaneme funkci *sin* (viz Obrázek 2.6 vlevo). Kdybychom zvolili ještě jeden bod x a y, lišící se od původně zvoleného bodu a udělali s ním ty samé operace, vyjde nám jiná sinusovka, která však bude mít průsečík s první sinusovkou. Tento **průsečík** označuje přímku, která prochází právě námi definovanými body.

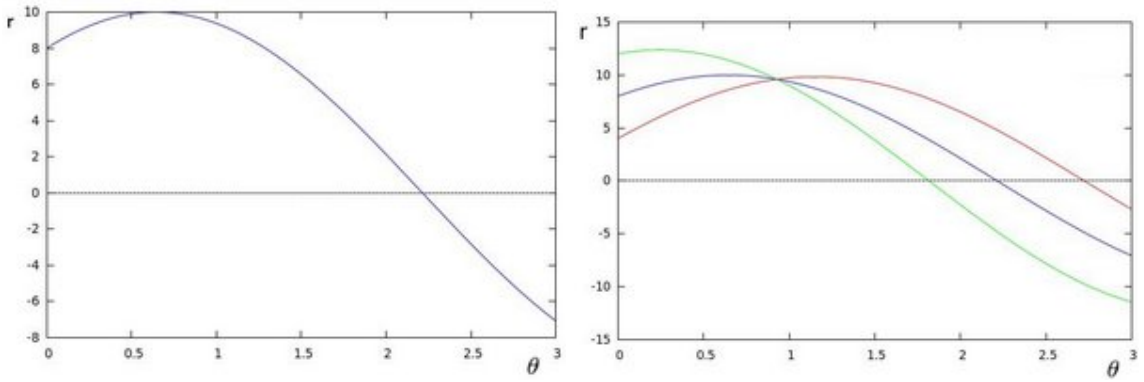
Kdybychom přidali další body, které budou procházet touto přímkou, jejich sinusovky by se protínaly v jednom bodě (viz Obrázek 2.6 vpravo). Houghova transformace pro detekci přímek tedy počítá tyto průsečíky a podle nich určuje, kolik přímek se v obrázku nachází. Jelikož by se ale každé dva body ležící vedle sebe daly brát jako přímka, je třeba zadat omezení počtu průsečíků, které je nutné pro uznání za přímku. Čím více průsečíků existuje, tím je pravděpodobnější, že jde o přímku.

2.7 Nástroje pro zpracování obrazu a počítačové vidění

V této části budou uvedeny a popsány existující nástroje a knihovny z oblasti zpracování obrazu. Knihovna `OpenCV`, která je v projektu použita, zde bude popsána více dopodrobna.



Obrázek 2.5: Ukázka reprezentace přímky v Houghově transformaci.



Obrázek 2.6: Ukázka sinusovky pro jeden bod (vlevo) a průniku sinusovek pro 3 body (vpravo). [8]

MATLAB (matrix laboratory) [1] je vysoce robustní multiplatformní nástroj nejen pro matematické výpočty, ale také pro zpracování obrazu. Program obsahuje vlastní skriptovací programovací jazyk (MATLAB - název jazyka je ekvivalentní názvu programu) a interaktivní programové prostředí. Co se týče matematických operací, podporuje např. jednoduchou manipulaci s maticemi, vykreslení grafů funkcí a dat. Kromě matematických funkcí se v něm dá programovat, vytvářet uživatelská rozhraní, ale také využívat knihovny napsané v jiných jazycích. Pro zpracování obrazu obsahuje tzv. **Image Processing Toolbox** [2], který poskytuje obsáhlou kolekci algoritmů, funkcí a aplikací. Lze ho tedy využít při analýze, segmentaci obrazu, odstranění šumu a dalších operacích. Mnoho funkcí podporuje vícejádrové procesory a grafické karty, takže je lze spouštět paralelizovaně.

Nevýhodou tohoto nástroje jsou jeho nároky na výkon, vzhledem ke všem funkcím, které obsahuje, ale také jeho cena a proprietární a uzavřené zdrojové kódy.

CImg [3] je sada nástrojů pro zpracování obrazu v jazyce C++. Je nejen zdarma, ale má také volně dostupné zdrojové kódy. Je také multiplatformní a měla by tedy fungovat se všemi nejrozšířenějšími C++ překladači. Pro její použití stačí do vlastního kódu přidat jeden hlavičkový soubor, nejsou tedy třeba žádné další závislosti. Knihovna poskytuje např. funkce pro načítání a ukládání obrázků, přístup k jednotlivým pixelům a jejich změnu, transformace, kreslení, atd. Výhodou je také její rychlost vzhledem k optimalizovanému

[1] http://www.mathworks.com/products/matlab/whatsnew.html
[2] http://www.mathworks.com/products/image/
[3] http://cimg.eu/

zdrojovému kódu v C++.

Nevýhodou je celý zdrojový kód napsaný do jednoho souboru, ale také ne příliš přehledná oficiální dokumentace.

PIL (Python Imaging Library) [4] je knihovna, která přidává do skriptovacího programovacího jazyka `Python` funkcionalitu pro zpracování obrazu. Kromě hromadného zpracování obrázků a jejich vykreslování v okně tato knihovna umí základní funkce zpracování obrazu, jako je rotace, translace a jiné transformace, převody barevných prostorů, nebo výpočet histogramu.

Poslední knihovnou, kterou zde uvedu, je **OpenCV** (Open Source Computer Vision) [5]. Tato multiplatformní knihovna s volně dostupnými zdrojovými kódy je použitelná s kterýmkoliv z nejpoužívanějších programovacích jazyků, jako jsou **C, C++, Python, nebo Java**. Dá se použít nejen pro vývoj aplikací na počítače, ale také na mobilní telefony s operačními systémy **Android a iOS**. Knihovna obsahuje sadu optimalizovaných funkcí pro zpracování obrazu, které využívají, když je to možné, vícejádrových procesorů.

[4] http://www.pythonware.com/products/pil/
[5] http://opencv.org/

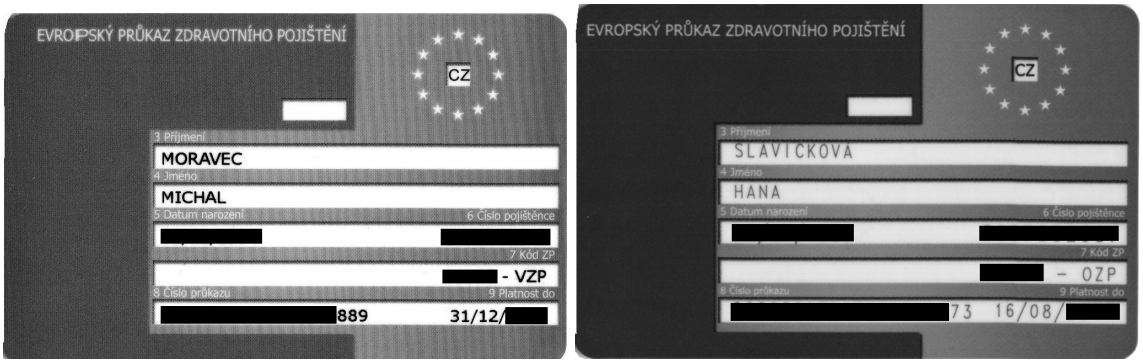
Kapitola 3

Metoda pro nalezení textových polí na naskenované kartě ZP

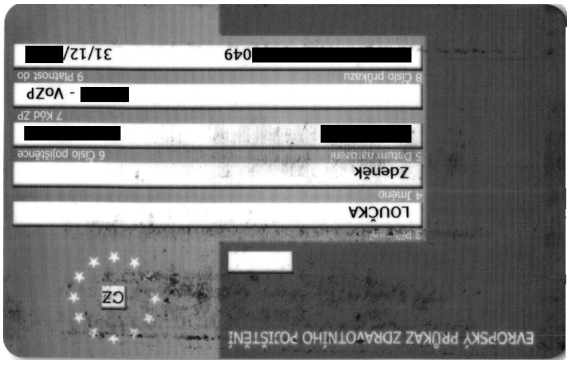
V této sekci bude nejprve popsána analýza, která byla provedena před samotným návrhem řešení. Dále bude čtenáři představen **návrh řešení** 3.3, jehož úspěšnost a efektivita bude porovnána v dalších částech práce. Finální řešení splňuje všechny požadavky dle zadání, a to **filtraci** textových polí ze všech druhů karet poskytnutých jako testovací sada (např. i karty uvedené na Obrázcích 3.2 a **korekci**, když je karta otočená. Jako **ukázka** tohoto řešení bude karta otočená o 180 stupňů 3.2, na které jsou navíc nečistoty. Na konci budou vidět výstupy správně otočené a ve správném pořadí. Řešení dále splňuje dodatečné požadavky od firmy **Medingo**, což je volba počtu výstupních obrázků. V základu program vrátí 5 textových polí, které karta obsahuje. Ve dvou textových polích se však v každém nacházejí dvě informace (celkem tedy čtyři), což znamená, že se dají rozdělit na celkem 7 textových polí. Dalším parametrem je přepínač postprocessingu, který bude zmíněn dále.

3.1 Analýza problému

Naskenovaná karta zdravotní pojišťovny obsahuje celkem **5 textových polí**, které je třeba vyseparovat, aby se z nich daly zjistit informace. Na prvním poli se nachází **příjmení**, na druhém **jméno**, na třetím **datum narození a číslo pojištěnce**, na čtvrtém **kód zdravotní pojišťovny** a na posledním **číslo průkazu a platnost**. Jelikož se na třetím a pátém textovém poli nacházejí 2 informace, je třeba dle zadání, aby je program dokázal od sebe odlišit a vrátit zvlášť, když je třeba. Naskenovaná karta je vždy **černobílá**, tudíž není třeba řešit barvy, které se na reálné kartě nacházejí. Co však řešit třeba je, jsou všechny různé druhy **poškození**, ať už na reálné kartě, tak v rámci skenování způsobené pohnutím. Když jsou textová pole posunuta vlivem špatného naskenování, je třeba provést **korekci**, která tyto pole zarovná a pokusí se je reprodukovat tak, aby byly tvarem co nejvíce podobné polím na skenované kartě. Karta může být do skeneru vložena z kterékoliv strany, tudíž musí program počítat i s libovolným **otočením**, přičemž výstup musí zůstat vždy nejen ve stejném pořadí, ale také i správně otočený tak, aby se text dal číst a nebyl vzhůru nohama. Co se týče výstupů, program by měl vracet buď 5 textových polí, nebo 7 částí, které obsahují každá jednu informaci. Všechny tyto výstupy by měly být vždy ve stejném **pořadí**, aby se daly použít jako vstup do dalšího softwaru.



Obrázek 3.1: Vlevo neotočená, nepoškozená karta nového formátu, vpravo karta s odlišným rozložením barev textu a pozadí textového pole.



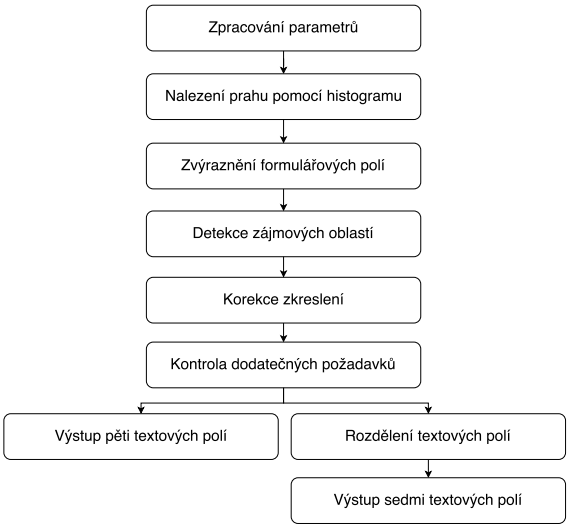
Obrázek 3.2: Ukázka karty otočené o 180 stupňů.

3.2 Ošetření vstupních parametrů

Jako první je třeba načíst **obrázek zadaný uživatelem**, ale také ošetřit jednotlivé další **parametry**. Jak bylo už zmíněno, je tu parametr určující počet výstupních polí, dále parametr určující, zda-li bude na výsledná pole aplikován postprocessing ve formě dynamického prahování. Parametr, který specifikuje cestu k obrázku, je jediný povinný. Když uživatel nezadá parametr určující počet výstupů, implicitně se vrátí 5 textových polí. Když se nezadá postprocessing parametr, implicitně se postprocessing nedělá.

3.3 Dynamický výpočet prahu podle histogramu

Ze vstupního obrázku se vypočítá funkce **histogramu**, která obsahuje četnosti jednotlivých barevných odstínů od nuly, což je černá barva, až do 255 představující bílou. Základní myšlenkou tohoto řešení je projít histogram jako **spojitou** funkci a najít v něm lokální **maxima**. Jelikož je to ale funkce **diskrétní**, a navíc jednotlivé hodnoty vedoucí do lokálního maxima nemusí být seřazené **vzestupně**, je třeba aplikovat jiný přístup. Nejprve se tedy projdou všechny hodnoty histogramu, a zjišťuje se, zda-li hodnota, která se nachází o dvě místa vlevo od procházené, je **menší** než hodnota, která se nachází o jedno místo vlevo od procházené. Analogicky se takto zkontroluje hodnota o jedno místo vlevo, a zda-li je **menší** než procházená hodnota. Dále se jde i od vrcholu **vpravo**, kde hodnoty vpravo od procházené musí být **menší** než tato procházená hodnota. Jestliže procházený bod **splňuje**



Obrázek 3.3: Blokové schéma řešení.

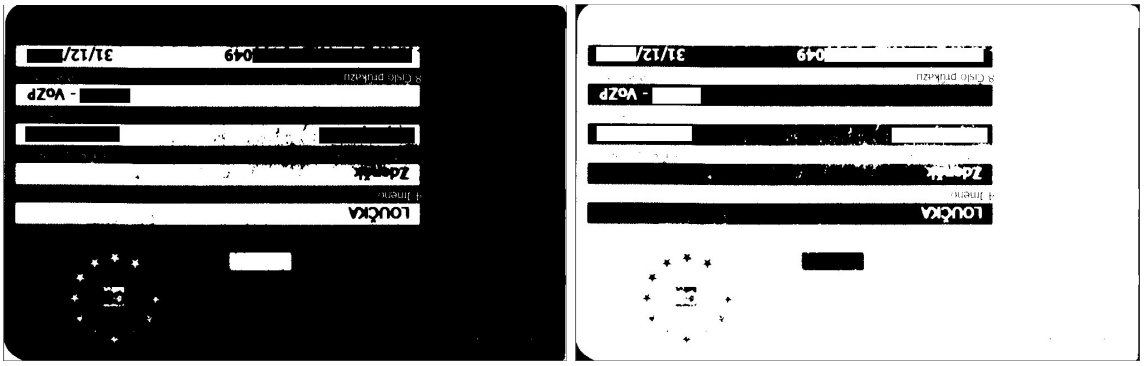
všechny tyto parametry, je přidán do kolekce vrcholů. Důležité ale je histogram procházet **od druhé hodnoty** jak zleva, tak zprava, jinak dojde ke čtení mimo rozsah histogramu.

Když se prochází histogram **zleva**, tak lokální maximum se hledá v **levé čtvrtině** kolekce s vrcholy, jinak se hledá v levé čtvrtině kolekce, která má **obrácené pořadí** prvků. Lokální maximum se najde jednoduchým vyhledáním největšího prvku v rámci prohledávaných, čímž se zajistí, že prvek je nejen **nejvyšším**, ale zároveň splňuje parametr **lokálního maxima** díky kontinuitě prvků, které ho předcházejí a následují.

Nalezené **lokální maximum** se ale nedá použít jako práh, protože by se následným prahováním obrázek rozdělil nežádoucím způsobem. Při filtraci textu by se např. vyfiltroval buď vnitřek jednotlivých písmen, nebo dokonce žádná písmena a jen černé prvky z pozadí. Je tedy třeba z lokálního maxima přejít do **nejbližšího lokálního minima**. Toto minimum se hledá již triviálněji, a to tak, že se od vrcholu lokálního maxima jde **vpravo**, když se jedná o procházení histogramu zleva, jinak se jde **vlevo**. Během procházení se kontroluje, zda-li je nová hodnota **menší než původní**. Jelikož je původní hodnota lokální maximum, tak by algoritmus skončil hned při prvním testu, a proto je zde hodnota **tolerance**. Ta zajistí, že když se nalezne hodnota **nižší, než je ta stávající**, tak se zjistí, zda-li je tolerance **větší než 0**, a pokračuje se dále. Stávající hodnota se tedy stále **aktualizuje**, dokud není hodnota tolerance nulová. Je také nutno zmínit, že když se narazí na hodnotu, která je **vyšší než ta aktuální procházená**, tolerance se **nemění**. Na závěr se tedy vrátí tato hodnota, která představuje výsledný práh využitelný v libovolné funkci prahování.

3.4 Zvýraznění formulářových polí

Po zpracování parametrů se tedy načte obrázek a zjistí se jeho rozměry. Dále se zjistí, zda-li jeho výška není větší než šířka. Když ano, tak to znamená, že je obrázek **otočen** buď o 90, nebo o 270 stupňů, a je třeba ho **otočit zpět**. V této fázi se ještě **nerozpoznává**, zda-li je otočen o 180 stupňů, nebo otočen není, což bude řešeno až dále. Když je tedy obrázek otočen v jeden z těchto dvou případů, tak se pomocí operací transponování matice a přehození všech bodů v horizontálním směru obrázek otočí o 90 stupňů, čímž se stane jeho šířka větší než jeho výška.



Obrázek 3.4: Ukázka karet po aplikování dynamického prahování (vlevo) a binární inverze (vpravo).

Nyní se provádí **dynamické prahování s procházením histogramu zprava** s hodnotou tolerance 30 (viz Obrázek 3.4 vlevo). Po **binární inverzi** je výsledkem opět obrázek 3.4 vpravo s bílým pozadím, bílým textem a černými textovými poli, na které se aplikují **morfologické operace uzavření a otevření**. Tyto operace postupně nejprve vymažou **černé body z bílých oblastí** a poté **bílé body z černých oblastí**. Bílý text se v obrázku slije do jednotlivých oblastí, kde některé může přesahovat za hranici černého textového pole, což se bude řešit později. Nyní je obrázek připraven pro následné vyhledání kontur.

Následuje nalezení **kontur** z binárně invertovaného obrázku z předchozího kroku. Pro jednotlivé nalezené kontury se spočítá **obsah konvexní obálky** a souřadnice společně s výškou a šířkou nejmenšího možného obdélníku, do kterého se tyto kontury vejdou. Kontury se dále musí opět vyfiltrovat do kolekce, která bude obsahovat jen ty, které představují textová pole. Tímto filtrem je kontrola na správný obsah (musí mít obsah menší než průměrný obsah textového pole, stejně jako v druhém řešení), a dále se nesmí dotýkat okraje obrázku.

3.5 Detekce textových polí pomocí přímek

V **prvotních návrzích** bylo použito řešení pro detekci textových polí, kde je třeba si vytvořit prázdnou matici o stejných rozměrech, jako jsou rozměry původního obrázku, na kterou se bude aplikovat **Houghova transformace** pro detekci přímek. Z původního obrázku se detekují **kontury**, které se ale musí dále **vyfiltrovat**. Podmínkou filtrace je, že plocha kontury musí být **větší než průměrná minimální plocha textového pole**, a obdobně menší než průměrná maximální plocha textového pole. Tyto průměrné hodnoty jsou zjištěny předem experimentálně a v programu jsou zadány jako konstanty. Vyfiltrované kontury jsou dále procházeny, aby se zjistila x souřadnice bodů, které leží nejvíce **vlevo a vpravo** v rámci všech kontur. Tyto body se budou používat jako meze pro vykreslení detekovaných a zarovnaných textových polí. Detekované kontury se také vykreslí do předem připravené prázdné matice.

Nad touto maticí se provede **Houghova transformace**, která by měla detekovat všechny přímky, které jsou delší než 180 pixelů, což je experimentálně zjištěná hodnota, při které se odhalí ideální počet přímek, které představují hranice textových polí. Jelikož se může stát, že hranice textových polí nejsou vždy úplně rovné, tak se detekuje více přímek v rámci jednoho okraje, které se překrývají. V tomto kroku je počítáno s tím, že všechny karty mají

pevný počet textových polí, a to 5. To znamená, že Houghova transformace by měla odhalit přesně **10 přímek**, které tyto pole shora a zdola ohraničují. Když dojde k detekci více přímek, znamená to, že se překrývají. Je tedy třeba kolekci s přímkami projít a zkontrolovat, zda-li nemají společný **průsečík**. Výběr přímky, která v kolekci zůstane, se provádí v každé iteraci cyklu kontrolou na existenci průsečíku. Když přímka, která je právě procházena, obsahuje průsečík s přímkou již ověřenou na neexistenci průsečíku, tak do výsledné kolekce není přidána. V této výsledné kolekci se nachází nejvýše 10 přímek. Když je v kolekci méně než 10 přímek, algoritmus nebyl schopen detekovat všechna textová pole a končí chybou.

Výsledná kolekce je dále seřazena podle y **souřadnic** vzestupně a procházena v cyklu. Jelikož jsou v kolekci uloženy jen horizontální přímky ohraničující textové pole shora a zdola, nemají žádný začátek, ani konec. V tomto kroku je třeba z těchto hranic vytvořit nová, zarovnaná textová pole, což znamená že hranice je třeba vyrobit. Na to se použijí body nacházející se nejvíce vlevo a vpravo detekované a zmíněné dříve, čímž se vytvoří buď **obdélníková** (v případě, že jsou přímky vodorovné s horním a dolním okrajem karty) nebo **kosodélníková** (přímky jsou skloněné o úhel maximálně 5 stupňů od horního/dolního okraje karty) textová pole. Těmito čtyřúhelníky se vyplní další prázdná, předem vytvořená matice o rozměrech odpovídajících původní kartě, čímž se vytvoří výsledná maska, která označuje pozice textových polí.

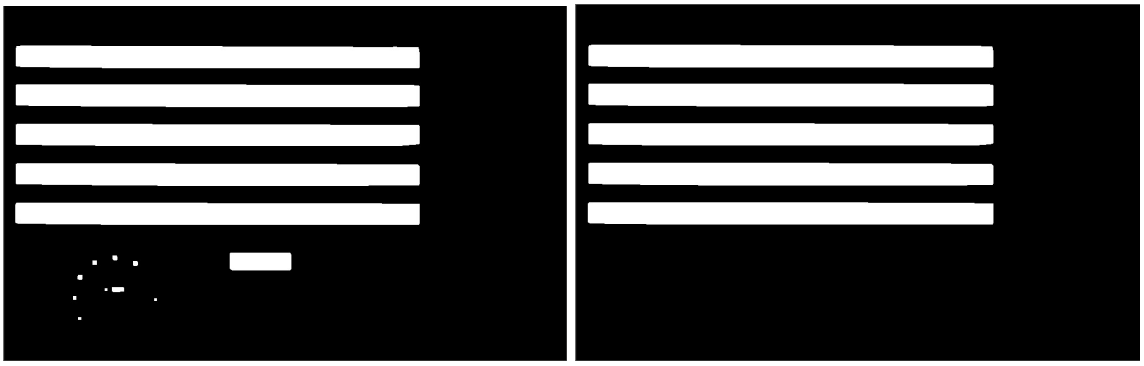
Jelikož se toto řešení ukázalo jako nevhodné pro detekci textových polí na kartách, které mají jakékoliv poškození vlivem skenování, bylo použito až následující, vylepšené řešení.

3.6 Detekce zájmových oblastí

Nyní zpět k výslednému řešení. Po filtraci ze sekce 3.4 je zapotřebí aplikovat na všechny kontury operaci **konvexní obálky**, která zahladí **konkávní poškození**, které může vzniknout při přetečení textu za hranice textového pole. Je třeba propojit části textových polí, která jsou rozdělena, a dále je vykreslit do **prázdné matice o rozměrech původního načteného obrázku** (viz Obrázek 3.5 vlevo). Rozdělená pole se spojí pomocí morfologické operace **uzavření**. Dále je třeba na obrázku se spojenými konturami znovu nalézt **nové, spojené kontury**. Jejich procházením v cyklu se dále zjistí, zda-li mají správný poměr stran. Tento poměr je zjištěn experimentálně, kdy šířka textového pole by měla být **alespoň desetkrát větší** než jeho výška. V rámci cyklu se také naplní kolekce, do které se ukládají souřadnice středů textových polí. Tato kolekce bude později využita pro korektní **otočení** karty, když je otočena o 180 stupňů. Jednotlivé kontury splňující podmínku správného poměru stran jsou také vykresleny do **prázdné matice**, opět o rozměrech původního načteného obrázku (viz Obrázek 3.5 vpravo). V tento moment by měla být detekována všechna textová pole.

3.7 Korekce zkreslení

Někdy se může stát, že se při skenování s kartou **pohne**, což způsobí **zkřivení obrazu**. Textová pole tedy již nejsou přesné obdélníky se zaoblenými rohy a vlivem posunutí pak zabírají po vložení do nejmenšího obdélníku, do kterého se vejdou, více místa na výšku, než je třeba. Je tedy nutné tato textová pole **zarovnat zpět** tak, jako by ke zkřivení nedošlo. K tomu je tu funkce, která jako první argument bere obrázek s vykreslenými obrysy výsledných textových polí vyrobený v předchozím kroku, a jako druhý argument je původní naskenovaný obrázek.

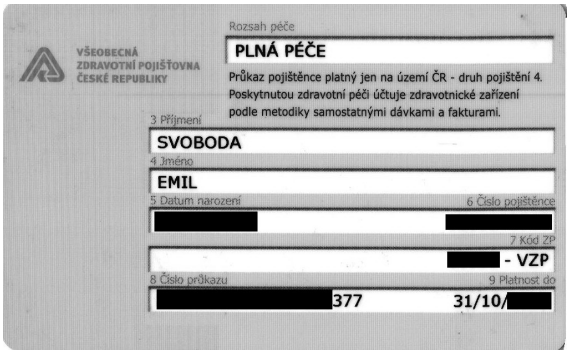


Obrázek 3.5: Ukázka vykreslení kontur na místech, kde jsou detekována textová pole.

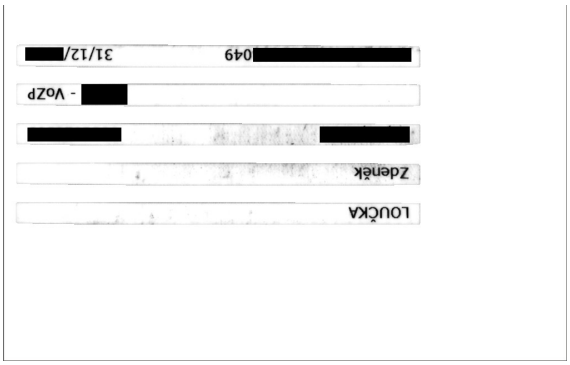
Nejprve se na obrázek s textovými poli aplikuje operace **vyhledání kontur**, které se následně prochází v cyklu. Nejprve jsou zjištěny **vlastnosti** procházené kontury, jako jsou souřadnice a rozměry. Cílem tohoto kroku je vytvořit **dvě kolekce**, do kterých se budou pro každou konturu ukládat body, které leží pouze v **horní a dolní** hraně kontury. Do jednotlivých kolekcí se tedy uloží podle jejich vertikální polohy. Přiřazení do jednotlivých polí probíhá tak, že se projde kolekce, které obsahuje všechny body, které procházená kontura obsahuje. Jedná se o body, které **definují okraj** procházené kontury. Jelikož je kontura spojitá a obsahuje vždy horní a dolní okraj, znamená to, že pro jednu x souřadnici může existovat více y souřadnic. Tohoto se tedy využije a do jednotlivých polí se nejprve přiřadí kterákoliv souřadnice, která je k dispozici, a když se objeví souřadnice, která je buď **výše** při ukládání do kolekce **horních souřadnic**, nebo **níže** při ukládání do kolekce **dolních souřadnic**, tak se pro dané x použije tato **nová**. Z tohoto vyplývá, že obě kolekce mají **stejný počet prvků**.

Když jsou vytvořeny tyto kolekce, tak se dále vypočítá **průměrná výška** textového pole tak, že se sečtou rozdíly horní a dolní y pozice pro danou x souřadnici a vydělí se jejich počtem. V tomto případě se může dělit délkou z **kterékoliv** z kolekcí (ať už horní, nebo dolní). Tato průměrná výška poslouží pro specifikaci rozměrů nové prázdné matice, která bude mít šířku stejnou jako již zmíněné kolekce. Do této matice se bude ukládat nové, už zarovnané, textové pole.

Původní textové pole se na nové zarovnané transformuje tak, že se postupně vezmou všechny **sloupce** původního pole o šířce **jednoho pixelu**, z nich se vypočítá **pozice středu** a nakonec se tento sloupec posune tak, aby tato pozice středu byla přesně na pozici **středu nového pole**. Jedná se tedy o vertikální posunutí tohoto sloupce. Při této operaci mohou nastat celkem **3 případy**. Prvním případem je, že původní sloupec je **menší než průměrná výška pole**. Toto se vyřeší nastavením chybějících pixelů v novém sloupci na bílou barvu. Druhým, ale velice ojedinělým případem, je **shodná výška původního i nového sloupce**. V tomto případě se neděje nic. A posledním, třetím případem je, že výška původního sloupce je **větší než průměrná výška textového pole**. Zde je zapotřebí provést **oříznutí** původního sloupce. Po aplikaci těchto operací je provedeno zarovnání všech jednopixelových sloupců tak, že se vejdou do nové matice, která má výšku nastavenou na průměrnou výšku textového pole, čímž je dosaženo požadovaného výsledku.



Obrázek 3.6: Ukázka karty s více než pěti textovými poli.



Obrázek 3.7: Ukázka karty s vyfiltrovanými textovými poli.

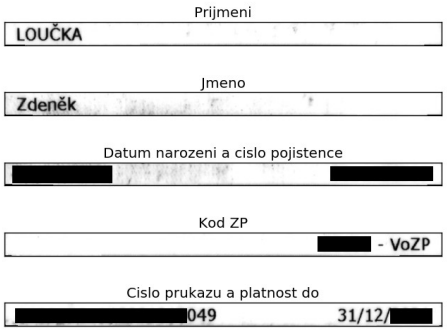
3.8 Kontrola dodatečných požadavků

Po zarovnání textových polí je třeba zkontrolovat, zda-li jich výsledná kolekce obsahuje právě **5**. Existují totiž karty, které obsahují textových polí 6 (viz Obrázek 3.6), kde se na tomto dalším poli nachází informace navíc. Jelikož je třeba zachovat **jednotný výstupní formát**, toto pole je třeba zahodit. Dále se může stát, že algoritmus není schopen odhalit ani 5 textových polí, což znamená, že na vstupu se nachází buď **zadní strana karty**, nebo došlo k **chybnému zpracování**. V obou těchto případech končí program **chybou**.

Když je ověřen správný počet textových polí, může se přejít ke kontrole, zda-li je možné textová pole **otočit**, když je vstupní karta otočena o 180 stupňů. Toto se provede pomocí výpočtu průměrné souřadnice středu všech textových polí. K tomu se využije kolekce, která obsahuje jednotlivé **souřadnice všech textových polí**, které se následně opět zprůměrují. Tato celková průměrná souřadnice středu by se měla vždy nacházet **napravo od středu karty**. Provede se tedy test na tuto podmínku, který když zjistí, že se střed nachází vlevo od středu karty, tak to znamená, že je karta **otočena o 180 stupňů**. V tento moment se ale textová pole neotáčí, jen je v **proměnné** uloženo, že se tak má stát. Samotné otočení se provede až **později**.

3.9 Výstup pěti textových polí

Po všech těchto operacích jsou textová pole připravena v **kolekci**, která se dá dále použít. Když uživatel specifikoval při startu programu argument, který zapíná **postprocessing**,



Obrázek 3.8: Ukázka pěti vyfiltrovaných výstupů.

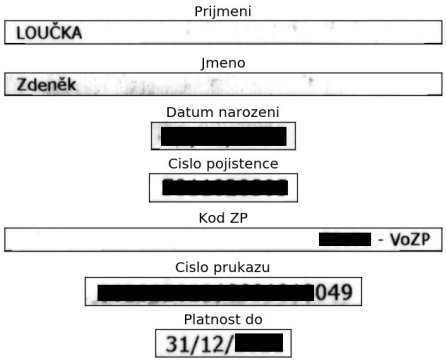
provede se ještě dodatečně prahování s dynamickým výpočtem prahu s procházením histogramu zleva s hodnotou tolerace 40. Když uživatel dále nespecifikoval počet výstupů, nebo specifikoval, že výstupů bude 5, je ještě třeba výsledná textová pole **otočit, když je to třeba**. Zkontroluje se tedy již dříve nastavená proměnná značící nutnost otočení, a potom se provede horizontální a vertikální obrácení, což způsobí rotaci o 180 stupňů. Následně jsou už jen textová pole vrácena jako výstup, např. zápisem do souboru (viz Obrázek 3.8).

3.10 Výstup sedmi textových polí

Uživatel ale může také specifikovat, že **počet výstupů bude 7**, což znamená, že se musí provést ještě dodatečné operace, které rozdělí pole, ve kterých se nachází **více než jedna informace**. Tato pole jsou vždy na všech kartách na stejných pozicích, což znamená, že se tyto pozice dají uložit do **konstant**. Algoritmus je také urychlen tak, že se rozdělí pouze ta pole, která obsahují více informací, a ta se následně prochází v cyklu.

Nejprve je na textové pole aplikováno **dynamické prahování s procházením histogramu zleva**, které jasně rozdělí text a pozadí a z obrázku udělá **binární**. Na tento binární obrázek je dále aplikována operace **eroze**, která text slije dohromady, čímž se z něj stane souvislá černá plocha. Tam, kde text není, by měla být souvislá bílá plocha. Takto upravené textové pole se projde po jednopixelových sloupcích, které obsahují určitý **poměr černých a bílých bodů**. Když je černých bodů více jak **polovina**, tento sloupec se nastaví **celý na černou barvu**, jinak se nastaví **celý na bílou**. Tímto vzniknou černé obdélníky stejně vysoké jako celé textové pole tam, kde se původně nacházel text, a zbytek vyplňuje bílá barva. Aby se dále daly najít kontury, které budou vyznačovat právě tyto oblasti s textem, je třeba obrázek **binárně invertovat**, aby bylo pozadí černé a textové oblasti bílé.

Po nalezení **kontur**, které by měly ohraničovat oblasti s textem, je třeba vyfiltrovat jen ty, které mají šířku **alespoň dvakrát větší než výšku**, jinak se může stát, že algoritmus vyfiltruje i chybné oblasti, kde se nenachází text, ale jen nějaká nečistoty či jiné poškození. Tyto vyfiltrované oblasti se uloží do kolekce, která se dále použije jako **maska** pro vrácení částí textových polí obsahujících text. U polí, která obsahují pouze jednu informaci a tedy není třeba je dělit, je postačí vrátit neupravené (viz Obrázek 3.9).



Obrázek 3.9: Ukázka sedmi vyfiltrovaných výstupů.

Kapitola 4

Nástroj pro detekci a korekci formulářových polí v obraze skenu karty ZP

V této části budou popsána konkrétní **řešení a algoritmy**, které jsou v projektu použity. Části z minulé kapitoly, které mohly být pro čtenáře po přečtení čistého textu **nejasné**, zde budou předvedeny konkrétně tak, jak jsou naimplementovány. Dále zde budou také tyto algoritmy vysvětleny přímo na příkladech.

Dále budou ukázány **experimenty a testy**, které byly provedeny pro ověření **správnosti** funkce algoritmu. Budou zde také porovnány jednotlivá řešení, ukázáno zlepšení úspěšnosti a dalších statistik. Dále bude definována a ukázána **testovací sada**, která byla pro tyto účely použita.

4.1 Použité knihovny a programovací jazyk

Celý projekt je implementován ve skriptovacím programovacím jazyce **Python 2.7**[1] . Tento jazyk je multiplatformní a ideální pro implementaci této úlohy. Jednotlivé funkce zpracování obrazu jsou použity z opět multiplatformní knihovny **OpenCV**[2], jejíž funkce se dají volat přímo z Pythonu. Jelikož knihovna provádí náročné výpočty, je třeba, aby byly vysoce optimalizované. Z tohoto důvodu používá pro tyto účely rozšíření Pythonu **Numpy**[3], které implementuje efektivní ukládání a operace nad maticemi, poli a dalšími strukturami. Toto rozšíření se dá také používat bez samotné knihovny OpenCV, když je třeba provádět např. iterace, které jsou jinak obecně ve skriptovacích jazycích neefektivní. Pro testování výstupů je použita knihovna **matplotlib**[4], která dokáže vykreslit obrázek do okna pomocí pár řádků kódu v Pythonu. Pro závěrečné a netestovací výstupy se obrázky již ukládají do souboru.

4.2 Parametry nástroje

Ve finálním řešení je ke zpracování argumentů použit modul **argparse**, který dokáže zastínit nízkoúrovňové testování pořadí zadaných parametrů uživatelem a abstrahovat jejich

[1] https://docs.python.org/2/
[2] http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html
[3] http://www.numpy.org/
[4] http://matplotlib.org/

význam. Celé zpracování argumentů je implementováno ve funkci `parse_args`. Jelikož je od uživatele požadován jako jediný povinný parametr **cesta k souboru s obrázkem**, je nastaven jako poziční, což znamená, že uživatel může zadat v příkazové řádce za název spouštěného programu rovnou cestu k obrázku, aniž by před ni uvedl jakýkoliv přepínač. U ostatních volitelných parametrů jsou však přepínače již třeba. Tyto přepínače jsou:

- `-p`, nebo `--postprocess` – zadáním tohoto přepínače se zapne postprocessing popsaný v části 3.9
- `-o`, nebo `--output` – nastaví počet výstupních polí na 7, jak je to popsáno v části 3.10
- `-d`, nebo `--debug` – zapne testovací mód, při kterém se po každém kroku zobrazí v matplotlib okně výsledek dané operace

Jednotlivé přepínače jsou v programu zastoupeny odpovídajícími boolean proměnnými `postprocess`, `outputs` a `debug`, které jsou nastaveny při jejich specifikaci. Jelikož jsou ale parametry nepovinné, musí se hodnoty proměnných nastavit i když uživatel žádné přepínače nezadá. V tomto případě se nastaví na implicitní hodnoty, které jsou zmíněny v části 3.2.

4.3 Dynamický výpočet prahu pomocí histogramu

Sada dvou funkcí popsaných v části 3.3 obsahuje část, ve které se hledají vrcholy, neboli lokální maxima. První je funkce `histFindBlack`, která prochází histogram zleva. Nejprve se vypočítá histogram pomocí funkce `hist`, která je obsažena v knihovně **matplotlib**, a je vrácen do proměnné `tmp`. Dále je vytvořen prázdný list `vrcholy`, do kterého se budou jednotlivé vrcholy vkládat. Následně se prochází histogramem pomocí tohoto kódu:

```
for i ← 0 to 254 do
    if i > 2 and i < 252 then
        if tmp[i] > 50 then
            if tmp[i − 2] < tmp[i − 1] and tmp[i − 1] < tmp[i] and
            tmp[i] > tmp[i + 1] and tmp[i + 1] > tmp[i + 2] then
                vrcholy.append(tmp[i])
            end
        end
    else
        vrcholy.append(tmp[i])
    end
end
```

Algoritmus 1: Hledání vrcholů

Do listu `vrcholy` se pro zjednodušení přidávají i body, které leží na krajích histogramu. Po tomto cyklu se pak provádí test, který vybere z první čtvrtiny vrcholů ten nejvyšší a ten se může nacházet úplně vlevo v bodě $tmp[0]$. Dále je tu podmínka zajišťující procházení jen těch vrcholů, jejichž hodnota je alespoň 50. Tímto se zamezí procházení vrcholů, které rozhodně nejsou lokálními maximy.

Druhou funkcí pro procházení histogramu, tentokrát zprava, je `histFindWhite`, která funguje velice obdobně, ale s tím rozdílem, že se cyklem prochází od hodnoty 255 do hodnoty 0.

4.4 Korekce špatně naskenovaných textových polí

Jak bylo zmíněno v části 3.7, všechna textová pole jsou procházena v cyklu, kde je vypočítána průměrná výška pole, do kterého se potom postupně vkládají jednotlivé jednopixelové sloupce. Tato funkcionalita je implementována ve funkci `correctShiftedAreas`. Samotná korekce jednopixelových sloupců se provádí v cyklu, kde je nejprve zjištěna aktuální výška sloupce takto:

$$rel\_height = bottom[i] - top[i]$$

Zde je proměnná `rel_height`, do které se uloží tato výška, která se vypočítá odečtením pozice horního pixelu od spodního. List `bottom` obsahuje pixely nacházející se ve spodní části kontury, `top` obsahuje pixely v horní části. Index `i` ukazuje na právě procházený sloupec.

Průměrná výška textového pole je již uložena v proměnné `mean`. Pro zjištění pozice horní a dolní části jednopixelového sloupce tak, aby byl zarovnán na prostředek průměrného výsledného textového pole, je třeba provést tento výpočet:

$$rel\_top = round(mean/2 - rel\_height/2)$$

$$rel\_bottom = round(mean/2 + rel\_height/2)$$

Do proměnné `rel_top` se tedy uloží horní bod, a analogicky do proměnné `rel_bottom` dolní bod. Tyto body se dále použijí jako hranice pro kopírování do výsledného textového pole. Aby nedošlo k přetečení přes hranici tohoto pole, je třeba ještě zařídit oříznutí těch sloupců, jejichž hranice přesahují hranice výsledného pole. Předpokládejme, že výsledné pole je umístěno na pozici $(0, 0)$, výšku má *mean* a šířku velikost pole *top* nebo *bottom*, které jsou co se týče délky ekvivalentní. Pro tento účel je proveden následující výpočet:

if $rel\_top < 0$ **then**
| top_crop = -rel_top
end
if $rel\_bottom > mean$ **then**
| bottom_crop = rel_bottom - mean
end

V proměnných *top_crop* a *bottom_crop* se nyní nachází hodnoty, které značí počet znaků, o kolik se musí původní jednopixelový sloupec oříznout shora a zdola. Když jsou tyto hodnoty nulové, ořezávaní se neprovádí. Do výsledného obrázku se kopíruje do oblasti $(i, rel\_top + top\_crop, i + 1, rel\_bottom - bottom\_crop)$, která vyznačuje obdélník (x_1, y_1, x_2, y_2), kde bod (x_1, y_1) je levý horní roh a (x_2, y_2) je pravý dolní roh.

4.5 Data a průběžné testování

K testování mi byla poskytnuta data ve formě naskenovaných karet zdravotní pojišťovny přímo v reálné čtečce **Evipa**[5]. Celkově poskytnutých dat bylo **317**, z toho 21 bylo zadních částí karet a 296 předních částí. V této sadě jsou obsaženy jak karty nejnovějšího formátu, tak staršího. Dále také obsahuje karty **špatně naskenované**, jak bylo uvedeno dříve, **posunutím** karty ve skenovacím přístroji, karty se **zašpiněním a reálným poškozením**, což dále znesnadňuje samotnou detekci. Některé karty jsou také **otočené**, s čímž musí

[5] http://www.medingo.cz/evipa.html

	Celkem	Počet úspěšných testů	Počet neúspěšných testů
Přední strana	296	285	11
Zadní strana	21	19	2

Tabulka 4.1: Výsledky finálních testů

výsledný detektor nejen počítat, ale vrátit výsledná textová pole ve **stejném pořadí**, jako kdyby karta otočená nebyla.

Aby měl program v reálném provozu co nejvyšší úspěšnost, měl by detekovat **co nejvíce polí v rámci testovací sady**. Pro začátek jsem z celkové sady dat vybral pouze **neotočené, nijak nepoškozené karty nejnovějšího formátu**. Těchto karet jsem pro testovací účely vybral 10 a snažil jsem se, aby na nich detektor našel všechna textová pole korektně. Tímto vzniklo počáteční řešení. Na předem vybraných deseti kartách bylo řešení schopno detekovat všech 50 textových polí, čímž je dosaženo 100% úspěšnosti v rámci testovací sady.

Aby řešení fungovalo i na ostatní karty, bylo ho třeba změnit tak, aby bylo využito **dynamické určení prahu pomocí histogramu**. Zde jsem vybral jako testovací sadu již 20 karet, které obsahují jak **normální nepoškozené, tak i ty poškozené a špatně naskenované**. Je zde zahrnuta i jedna karta starého formátu. Řešení bylo experimentálně nastaveno tak, aby na těchto kartách detekovalo také 100% textových polí.

Jelikož i toto řešení však nesplňovalo všechny požadavky zadání, jako je otáčení, a navíc se objevily i požadavky nové přímo od firmy Medingo, bylo třeba řešení vylepšit. Vybral jsem opět 20 karet, kde ale některé z nich jsou i otočené tak, aby zde byla alespoň jedna otočená o 90, 180 a 270 stupňů. Algoritmus se mi opět podařilo naladit tak, aby na těchto datech měl 100% úspěšnost detekce textových polí.

4.6 Experimenty

Po těchto ladících testech bylo ale třeba řešení ověřit na **celé testovací sadě**, která čítá celkem **317 karet**. Z těchto karet je celkem **296 předních částí** a zbytek, **21 karet, je zadních**. Pro úspěšné splnění testu na detekci zadní strany karty musí algoritmus skončit **chybou**. Z 21 karet, které byly testovány, skončil algoritmus chybou u **19** z nich. U dvou detekoval textová pole, jelikož se na zadní straně nacházel černý text na bílém pozadí. Úspěšnost v tomto testu byla tedy 90.4%.

Dalším testem byla detekce textových polí na **předních stranách karet**. Aby se test dal brát jako úspěšný, algoritmus nesměl skončit chybou a zároveň musel být schopen vyfiltrovat správně všech **5** textových polí. Když byla karta otočena, musel ji také korektně otočit zpět. V tomto testu program prošel testem u 285 karet, kde splnil všechny uvedené podmínky. Úspěšnost v tomto testu byla 96.3%. Výsledky testů jsou uvedeny v tabulce 4.1

Jelikož bylo také třeba ověřit rychlost algoritmu, bylo provedeno několik testů, ve kterých byl změřen čas, po který program běží. Toto měření bylo provedeno pomocí linuxové utility **time**. V rámci testu byl program spuštěn celkem na osmi různých kartách, a v rámci každé karty byl spuštěn buď bez parametrů, s jedním, s druhým, nebo s oběma parametry. Výsledky jsou vidět v tabulce 4.2, kde na posledním řádku jsou vidět průměrné časy běhu s jednotlivými parametry. Testy byly provedeny na počítači s procesorem **Intel Core i5 3210M** s frekvencí 2.5 GHz.

Č. měření	Bez parametru (s)	`-o` (s)	`-p` (s)	`-p` a zároveň `-o` (s)
1	1.280	2.034	2.524	3.315
2	1.244	2.027	2.585	3.274
3	1.321	2.034	2.552	3.202
4	1.291	2.168	2.530	3.341
5	1.244	2.105	2.547	3.231
6	1.224	2.158	2.491	3.279
7	1.224	2.055	2.488	3.271
8	1.263	2.166	2.554	3.318
Průměr	1.261375	2.093375	2.533875	3.278875

Tabulka 4.2: Výsledky měření trvání běhu algoritmu s danými parametry

Kapitola 5

Závěr

Tato práce představila problém automatického zpracování a extrakce informací z obrazu naskenované karty zdravotní pojišťovny, kde mohou nastat problémy se špatně naskenovanými, případně poškozenými kartami, ale také s otočením a detekcí zadní strany. Nejprve byly představeny stávající řešení, které by mohly být aplikovatelné na tento typ problému a dále byla vysvětlena teorie potřebná k pochopení jednotlivých funkcí ze zpracování obrazu použité dále.

Po analýze problému byl navržen postup, který nejen řeší tyto problémy, ale je také schopen co se detekce týče pokrýt co největší sadu karet. Byly zde také popsány jednotlivé kroky a metody potřebné k dosažení výsledku, jako jsou např. korekce špatného naskenování karty a korekce otočení. Jednotlivé metody byly do podrobna popsány, aby byla jasná jejich funkce a význam.

Navržený postup byl implementován v jazyce `Python` s pomocí knihovny `OpenCV` a dalších funkcí, přičemž při implementaci bylo důležité řešení průběžně testovat pro ověření správnosti. Díky využití optimalizovaných funkcí z této knihovny byla dosažena nejen vysoká úspěšnost v detekci, ale také celková rychlost. Po dokončení implementace byl program testován na poskytnutých reálných datech, karet ZP, s úspěšností 96.3% v detekci textových polí na předních stranách karet. V detekci zadních stran karet byl algoritmus méně úspěšný, a to 90.4%, vzhledem k existenci textových polí na kartách, na kterých detekce selhala.

Výsledek je využit firmou Medingo, která vlastní systém Evipa. Tento systém obsahuje čtečku karet zdravotní pojišťovny, která po naskenování spustí moje řešení, které dále detekuje jednotlivá textová pole. Tato pole jsou dále vstupem do OCR softwaru, který se znalostí významu jednotlivých polí je schopen vyseparovat jednotlivé informace o pacientech. Tyto informace jsou dále použity pro vyhledání pacienta v elektronické kartotéce u lékaře.

Jelikož bude firma chtít algoritmus nejen používat, ale i dále vylepšovat, byl implementován tak, aby byl dále rozšiřitelný.

Literatura

[1] Ar, I.; Karsligil, M. E.: Text Area Detection in Digital Documents Images Using Textural Features. In *Proceedings of the 12th International Conference on Computer Analysis of Images and Patterns*, CAIP'07, Berlin, Heidelberg: Springer-Verlag, 2007, ISBN 978-3-540-74271-5, s. 555–562.
URL http://dl.acm.org/citation.cfm?id=1770904.1770980

[2] Bradski, G. R.; Kaehler, A.: *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008, ISBN 0-596-51613-4.

[3] Chen, H.; aj.: Robust Text Detection in Natural Images with Edge-Enhanced Maximally Stable Extremal Regions. In *Image Processing (ICIP)*, 18th IEEE International Conference on. IEEE, 2011.

[4] Fisher, R.; Perkins, S.; Walker, A.; aj.: Morphology. [online], 2003, [vid. 2016-5-4].
URL http://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm

[5] Horák, K.: Multimediální interaktivní didaktický systém. [online], 2010, [vid. 2016-5-5].
URL http://midas.uamt.feec.vutbr.cz/ZVS/Exercise10/content_cz.php

[6] Kornfield, E. M.; Manmatha, R.; Allan, J.: Text alignment with handwritten documents. In *Document Image Analysis for Libraries, 2004. Proceedings. First International Workshop on*, 2004, s. 195–209, doi:10.1109/DIAL.2004.1263249.

[7] Sonka, M.; Hlaváč, V.; Boyle, R.: *Image Processing, Analysis, and Machine Vision*. CL-Engineering, 2007, ISBN 978-0495082521.

[8] opencv dev team: Hough Line Transform. [online], 2011-2014, [vid. 2016-5-7].
URL http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html

Přílohy

Seznam příloh

Příloha A

Obsah CD

A.1 Textová verze bakalářské práce ve formátu PDF

A.2 Zdrojové kódy textu bakalářské práce v LaTeXu

A.3 Zdrojové kódy programu v Pythonu

A.4 Datová sada karet ZP k vyzkoušení (osobní údaje jsou záměrně vymazané)

A.5 Soubor README s pokyny k běhu programu

A.6 Soubor LICENCE s uvedenou licencí k programu

A.7 Plakát

A.8 Video

Příloha B

Plakát