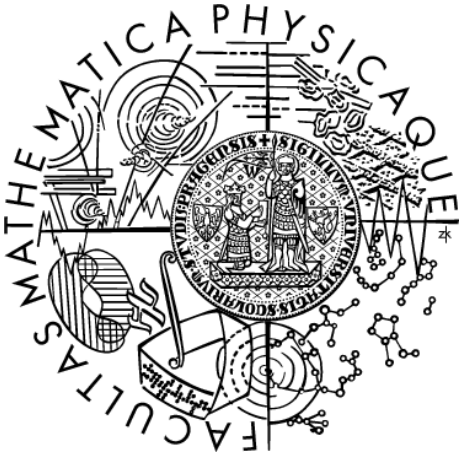
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Bohuslav Macháč

Načítání dat z tištěných dokladů

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Jan Kolomazník
Studijní program: Informatika
Studijní obor: Programování

Praha 2011

Rád bych poděkoval vedoucímu práce Mgr. Janu Kolomazníkovi za věnovaný čas, cenné rady a náměty.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle č60 odst. 1 autorského zákona.

V Praze dne 27.5.2011 Bohuslav Macháč

Název práce: Načítání dat z tištěných dokladů

Autor: Bohuslav Macháč

Katedra: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Jan Kolomazník

e-mail vedoucího: Jan.Kolomaznik@mff.cuni.cz

Abstrakt: V této práci jsem vyvinul aplikaci schopnou extrahovat data z naskenovaných dokumentů. Pro optické rozpoznávání znaků jsem použil externí OCR engine Tesseract, který lze snadno vyměnit. Pro jednotlivé doklady používám šablony s informacemi o datových oblastech a jejich datových typech. Pokusil jsem se automatizovat většinu kroků nutných pro extrakci dat a vytvoření nové datové šablony. Uživatel má možnost opravit nebo změnit výsledky těchto kroků. Pro výstup z aplikace jsem implementoval komponenty, které exportují data do formátů XML, HTML a do obyčejného textu. Další komponenty mohou být snadno přidány, aby přizpůsobily aplikaci různým použitím.

Klíčová slova: OCR, digitální zpracování obrazu, rozpoznávání textu

Title: Data extraction from document scans

Author: Bohuslav Macháč

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Jan Kolomazník

Supervisor's e-mail address: Jan.Kolomaznik@mff.cuni.cz

Abstract: In this work I developed an application capable of extracting data from scanned documents. For optical character recognition, I used external OCR engine Tesseract, but it can be easily changed. I use document templates, which have informations about data areas and its data types. I tried to automatize most of the steps which are required to extract data or create new data template. User can improve or change results of these steps. For export from application I implemented components, which export data to XML, HTML or plain text. Another components can be easily added, to adapt application for various uses.

Keywords: OCR, digital image processing, text recognition

Obsah

Úvod

Přečíst údaje z dokladu pro nás není žádný problém, stačí, že umíme číst. Pro počítač je taková úloha ovšem o mnoho složitější. Samozřejmě, pokud bychom doklad do počítače přepsali a dali ho počítači v textové podobě, zvládl by tuto úlohu stejně snadno jako my. Pokud mu ale poskytneme doklad naskenovaný jako obrázek, pak má k dispozici pouze spoustu jednotlivých pixelů, o kterých ví, jakou má který z nich barvu a kde je umístěn. Ovšem písmena a slova tak, jak je vnímáme my, pro něj v tu chvíli neexistují, vnímá je maximálně jako shluky pixelů s podobnou barvou. Zde přichází ke slovu digitální zpracování obrazu a počítačové vidění, aby i počítač mohl snadno z dokladu přečíst potřebné údaje.

Cílem následující práce je představit aplikaci, která umožní snadné rozpoznání dat z naskenovaného dokladu v počítači. Od pole pixelů ke smysluplnému a správnému textu, který odpovídá datům na dokladu, vede dlouhá cesta. Nejprve je potřeba naskenovaný obrázek upravit pro další zpracování, poté detekovat oblasti s daty, přiřadit jim logický význam a následně extrahovat samotný text. V této práci zmapuji jednotlivé kroky tohoto procesu a dám jej, v podobě aplikace, k dispozici uživateli, aby i on mohl rozpoznávat data z dokladů a rychle je uložit v digitální podobě v počítači. V průběhu procesu se může objevit mnoho překážek, zabraňujících úspěšnému dokončení úlohy. Vždy je popíšu a pokud to bude možné, dám k dispozici nástroj, jak se konkrétní překážky zbavit. Prvním předpokladem pro úspěšnou extrakci dat je dostatečná kvalita skenu, kterou vyžaduje i OCR engine a bez které nelze očekávat uspokojivý výsledek.

Aplikace je postavená nad externím OCR systémem Tesseract, ve verzi 3.0, která podporuje češtinu. OCR systém je ale připojen takovým způsobem, aby bylo snadné jej v případě potřeby vyměnit. Aby extrahovaná data měla nějakou strukturu, aplikace používá datové šablony, které pojmenovávají jednotlivé oblasti a přiřazují jim příslušnost k některému ze základních datových typů. Datové typy jsou využívány pro základní kontrolu korekce a mohou být využity při exportu, pro ovlivnění jeho podoby. Datové šablony definuje uživatel za pomoci aplikace, která mu automatizací navrhne podobu obrysu dokladu i jednotlivých datových oblastí. Tuto podobu může uživatel v případě potřeby manuálně upravit. Požadavek na výstup z aplikace se může lišit podle potřeb uživatele, v aplikaci jsou implementované vzorové výstupní moduly do formátu XML, HTML a pro textový editor. Další moduly je snadné přidat.

Program jsem pojmenoval *Optical Document Recognizer*. Inspiroval mě anglický název rozpoznávání textu - *Optical Character Recognition*. Změna na *Document* vyplývá ze zaměření aplikace a výraz *Recognizer* má určovat, že jde o konkrétní počítačovou aplikaci a ne o obecnou techniku jako v případě OCR.

Struktura práce

Toto je úvod práce, zde jsou názvy a obsah jednotlivých kapitol:

1. Teorie - obsahuje teoretické poznatky a algoritmy použité při tvorbě aplikace

2. Návrh aplikace - popisuje rozčlenění do hlavních problémů souvisejících s extrakcí dat z dokladů a popisuje, jak byly tyto problémy v aplikaci řešeny

3. Architektura aplikace - zabývá se impementací programu, rozdělením na jednotlivé části i strukturou a třídami obsaženými v těchto částech

4. Příručka programátora - představuje aplikaci z pohledu programátora, umožňuje základní orientaci v kódu a poskytuje rady, jak aplikaci dále rozšiřovat

5. Uživatelská příručka - manuál představující aplikaci uživatelům, popisuje všechny části aplikace a správný průběh práce v nich

V závěru jsou shrnuty dosažené cíle i náměty k další práci.

1. Teorie

Cílem aplikace je z naskenovaného dokladu extrahovat data. K uskutečnění tohoto procesu jsou potřeba znalosti o tom, kde v obrázku se dokument nachází, jaké je rozložení dat v dokladu a samozřejmě také nástroj na samotné rozpoznání textu.

Pro rozpoznání polohy a layoutu dokumentu, ale také textu, je potřeba využít nástrojů počítačového vidění. Úpravu obrázku do podoby vhodné pro rozpoznávání lze provést pomocí algoritmů z digitálního zpracování obrazu. Tato kapitola obsahuje přehled použitých metod z daných oblastí. Hodně postupů se používá v obou oblastech, proto rozdělení jednotlivých metod je spíše podle mého osobního dojmu.

1.1 Digitální zpracování obrazu

Poznatky o digitálním zpracování obrazu jsem čerpal převážně z této knihy [1]. Ostatní reference jsou uvedeny přímo v daných sekcích.

1.1.1 Prahování

Prahování je jednoduchý proces ze segmentace obrazu, kdy je obrázek převeden z šedotónových barev na binární. Základ algoritmu tvoří práh a všechny pixely, které mají hodnotu vyšší než tento práh, jsou označeny jako objektové pixely a je jim přiřazena jedna hodnota, ostatní jsou pixely pozadí a dostanou hodnotu druhou.

Důležitým krokem je určení vhodného prahu. K tomuto účelu existuje řada metod, které k určení prahu používají různé přístupy. Některé vycházejí z tvaru histogramu, a hledají vrcholy a údolí mezi nimi. V ideálním případě jsou vrcholy 2 a mezi nimi jednoznačné údolí, v takovém případě dávají tyto metody dobrý výsledek. Pokud je ale vrcholů větší množství, nebo jsou mezi nimi jen mělká údolí, nastává problém s výběrem správného oddělujícího údolí.

Jiné metody využívají shlukování, kdy jsou hodnoty pixelů rozděleny do 2 skupin(popředí a pozadí) a na kvalitu tohoto rozdělení jsou kladeny různé nároky a uplatňována specifická kritéria. Mezi tyto metody patří i mnou používaná Otsuova metoda.

Otsuova metoda

Předpokladem pro úspěšné použití této metody je vstup, který obsahuje dvě třídy pixelů. Algoritmus pracuje tak, že za pomoci histogramu spočítá optimální práh oddělující tyto dvě třídy na základě diskriminačního kritéria. Pro definici kritéria je nejdříve potřeba představit pár pojmů.

Rozdělme pixely v obrázku na 2 třídy T_0 a T_1 takové, že do třídy T_0 patří všechny pixely s hodnotou $0, \ldots, k$, do třídy T_1 pak všechny ostatní.

Vezměme ω_i jako pravděpodobnost třídy T_i, tedy součet pravděpodobností výskytu pro všechny hodnoty pixelů patřících do dané třídy.

Uvažujme μ_i jako střední hodnotu třídy T_i, tedy součet násobku hodnoty a její podmíněné pravděpodobnosti pro každou hodnotu patřící do třídy T_i.

Definujme H_i jako množinu hodnot pixelů ve třídě T_i.

Rozptyl třídy T_i označme jako σ_i^2 a definujme jako $\sum_{k \in H_i}(i - \mu_i)^2 p_i/\omega_i$.

Definujme μ_T jako střední hodnotu celého obrázku a σ_T^2 jako jeho rozptyl.

Nyní definujeme rozptyl v rámci třídy $\lambda = \sigma_B^2/\sigma_W^2$ a rozptyl mezi třídami $\kappa = \sigma_T^2/\sigma_W^2$, kde $\sigma_W^2 = \omega_0\sigma_0^2 + \omega_1\sigma_1^2$ a $\sigma_B^2 = \omega_0\omega 1(\mu_1 - \mu 0)^2$.

Diskriminační kritérium je formulováno jako *maximalizace separability výsledných tříd*, které je dosaženo pomocí minimalizace rozptylu v rámci třídy, avšak rychlejší je použití maximalizace rozptylu mezi třídami, která také maximalizuje separabilitu. Více o tomto algoritmu v článku [2].

1.1.2 Histogram

Histogram je statistika obrázku, udávající četnost výskytu pixelů s danou hodnotou. Sestavení histogramu je jednoduché, připravíme si akumulační pole, indexované hodnotou pixelu, poté projdeme celý obraz a pro každý pixel zvýšíme čítač odpovídající jeho hodnotě o 1. Výsledkem je tedy tabulka, kde pro každou hodnotu je uveden počet pixelů, které danou hodnotu v obrázku nabývají.

1.1.3 Morfologické operace

Morfologické operace jsou nelineární operace v obrazu, jejichž nejdůležitější součástí je *strukturovací element*. Ten se postupně aplikuje na okolí každého pixelu v obrázku a podle typu operace se určí výsledná hodnota pixelu. Zmíním jen morfologické operace fungují na binárních obrázcích, protože na jiných formátech obrázků je v aplikaci nepoužívám. V aplikaci často používám strukturovacím element ve tvaru horizontální úsečky, který pro velikost 3x3 vypadá následovně:

0	0	0
1	1	1
0	0	0

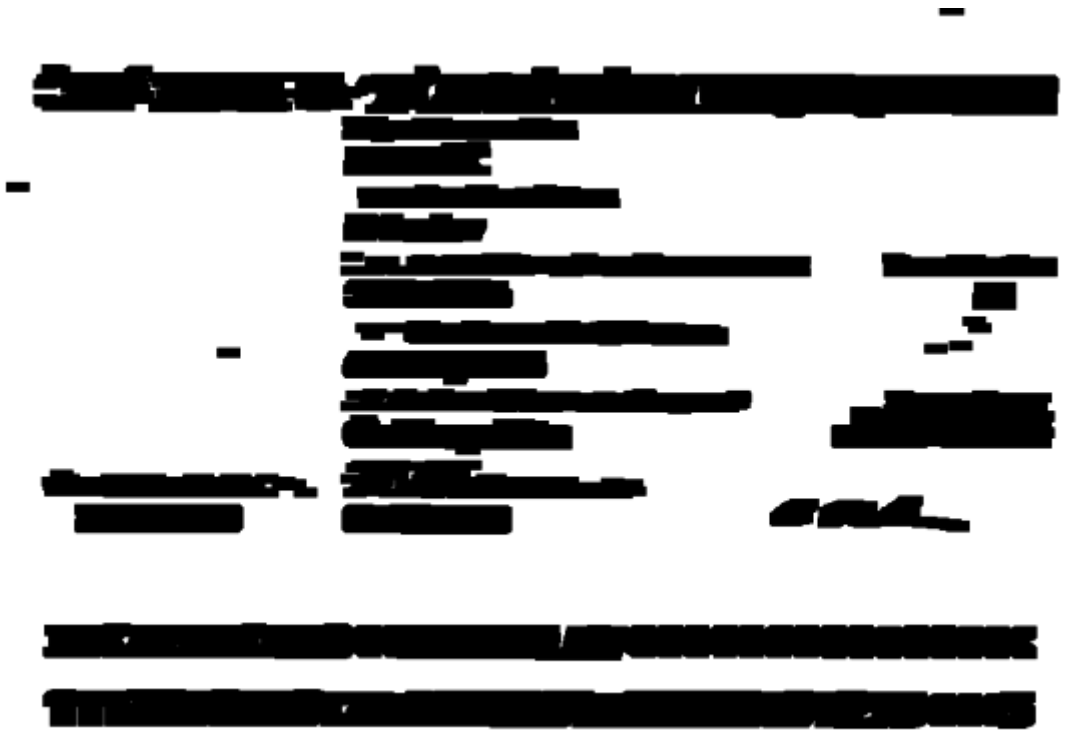
Výsledek dilatace s takovým strukturním elementem je znázorněn na obrázku 1.1

Eroze

Na každý pixel a jeho okolí postupně aplikujeme strukturovací element. Pokud na všech místech, kde je v elementu 1, je i v obrázku 1(popředí), pixelu přiřadíme hodnotu 1, jinak mu nastavíme 0. Výsledným efektem je úbytek pixelů na okrajích oblastí popředí a zvětšení mezer mezi těmito oblastmi.

Dilatace

Postup podobný jako u eroze, ale stačí výskyt 1 na jakémkoliv místě, kde se element a obrázek překrývají, tj. kde je u obou 1. Zde je výsledkem naopak rozšíření hranic oblastí popředí a zmenšení mezer mezi nimi.



Obrázek 1.1: Aplikace dilatace s horizontální úsečkou

Otevření a uzavření

Obě tyto metody jsou odvozeny od základních morfologických operací, dilatace a eroze. Otevření a uzavření jsou navzájem duální, to znamená, že pokud provedeme otevření objektových pixelů daným strukturním elementem, dostaneme stejný výsledek, jako kdybychom použili uzavření pixelů pozadí se stejným strukturním elementem.

Efekt otevření je podobný jako u eroze, také dochází k úbytku objektových pixelů na hranicích objektů, ale pouze v menší míře, přesný výsledek závisí na použitém strukturním elementu. Často se používá čtvercový element o délce hrany 3 pixely. Otevření je složení operací eroze a dilatace tak, že nejprve se na obrázek použije eroze a na výsledek poté dilatace, přičemž obě použijí stejný strukturní element.

Naopak uzavření je podobné dilataci, rozšiřuje objekty na okrajích, ale opět v menší míře než dilatace. Uzavření je také kombinací eroze a dilatace, ale tentokrát se jako první použije dilatace a až po ní eroze.

Hit-a-miss transformace

Další morfologickou operací je hit-a-miss transformace, kde hit znamená trefit a miss naopak minout, doslovný překlad tedy trefit-a-minout. Zde dochází k rozšíření strukturního elementu o další hodnotu. Dala by se nazvat například „nezájem“. Strukturní element se zde mění v jakousi masku. Pokud okolí vyšetřovaného pixelu má 1 a 0 na stejných místech jako element, nastaví se pixel na hodnotu 1,

v opačném případě na hodnotu 0. Pixely, jimž odpovídají na elementu hodnoty „nezájem“, nás nezajímají, jejich hodnota nemá vliv na výslednou hodnotu daného pixelu.

Ztenčení a ztluštění

Tyto operace využívají hit-a-miss transformaci. Podobně jako otevření a uzavření, jsou i tyto operace navzájem duální.

Ztenčení funguje tak, že pro daný obrázek a rozšířený strukturní element(má navíc hodnotu „nezájem“) spočítá hit-a-miss transformaci a tu pak od původního obrázku odečte. V této operaci tedy dochází k úbytku pixelů, typickým použitím je nalezení kostry objektů.

Ztluštění využívá hit-a-miss transformaci tak, že ji s původním obrázkem sjednotí. Pixely zde tedy přibývají. Typickým použitím je nalezení konvexního obalu, což je i případ využití v mé aplikaci.

1.2 Počítačové vidění

1.2.1 OCR

OCR(z anglického Optical Character Recognition) neboli optické rozpoznávání znaků je proces transformace tištěného textu do počítačového textu. Tištěný text se pomocí scanneru nahraje do počítače jako obrázek a pomocí OCR je pak převeden na normální počítačový text. Kvalita výsledku závisí zejména na kvalitě vstupního obrázku a samozřejmě také na kvalitě OCR enginu. Vytvořit kvalitní OCR není jednoduché, ani nejlepší existující enginy nejsou v rozpoznávání 100% úspěšné, a to ani na „ideálním“ vstupu co se do kvality obrázku a výskytu šumu týče.

1.2.2 Detekce hran

Jedná se o jeden ze základních nástrojů ve zpracování obrazu a počítačovém vidění, který v obrazu detekuje body, kde dochází k výrazné změně hodnoty. Takto nalezené hrany se pak dále používají třeba pro detekci objektů. Detektory hran jsou typicky založeny na derivacích, a to buď na první nebo druhé derivaci.

Mezi nejznámější metody založené na první derivaci patří:

- Prewittův operátor - K detekci hran používá výpočet odhadu gradientu, má rychlý výpočet, ale dává hrubé výsledky.
- Robertsův kříž - Výhodou je rychlost, ale velkou nevýhodou je vyšší citlivost na šum.
- Sobelův operátor - Je podobný Prewittovu operátoru, ale není tolik citlivý na směr hrany.
- Cannyho detektor hran - Jedná se o „state-of-the-art“ detektor, používá na obrázek konvoluci s Gaussovou funkcí a poté použije některý z výše

uvedených operátorů. Díky konvoluci s Gaussovou funkcí dojde k mírnému rozmazání a tím i omezení vlivu šumu.

Sobelův operátor používá filtry S_x a S_y:

$$\mathbf{S_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \mathbf{S_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Tyto filtry použije každý zvlášť na původní obrázek I a výslednou velikost gradientu $|\nabla I|$ spočítá jako $|\nabla I| = \sqrt{(\mathbf{S_x} * I)^2 + (\mathbf{S_y} * I)^2}$.

Zástupcem metod využívajících druhou derivaci je diferenciální detekce hran.

Konkrétní požadavky na detektory pro použití v aplikaci jsou popsány v sekci 2.1.1, kde ji i představení použité metody.

1.3 Houghova transformace

V základní verzi je cílem Houghovy transformace detekovat v obrazu přímky. Přímky jsou vyjádřeny v pomocném parametrickém prostoru podle následující rovnice.

$$xcos\alpha + ysin\alpha = \rho$$

Kde α je velikost úhlu normály přímky a ρ je vzdálenost přímky od počátku.

Článek představující použití takovýchto parametrů a popisující Houghovu transformaci. [3]

Algoritmus pak prochází obraz po pixelech a pro každý pixel prostředí spočítá všechny přímky, jimž by tento bod mohl náležet. Následně v akumulátoru přičte výskyt pro dané přímky. Parametry pro přímky v obrazu dostaneme nalezením lokálních maxim v akumulátoru a dalším filtrováním závislým na konkrétní aplikaci. Tento postup může být zobecněn na další objekty popsatelné analyticky pomocí parametrů. S narůstajícím počtem parametrů ovšem rychle narůstá náročnost výpočtu.

1.4 Geometrická analýza layoutu

Je součástí analýzy layoutu a soustředí se na fyzickou segmentaci jednotlivých komponent v obrazu. V závislosti na úrovni segmentace může v textu oddělovat jednotlivá písmena, slova, řádky nebo bloky textu. Nezabývá se logickým významem těchto oblastí. Cílem této analýzy je popis geometrické struktury dokumentu.

Existují dva přístupy ke geometrické analýze a to postup zeshora dolů, nebo opačný zespoda nahoru. Přístup zespoda nahoru začíná s velkým množstvím komponent a postupně je spojuje do větších celků, zatímco analýza zeshora dolů rozděluje větší části na menší. Důležitou součástí pro mé použití je *odhad šikmosti*.

1.4.1 Odhad šikmosti

Jde o proces, jehož cílem je detekovat úhel odklonu orientace dokumentu od horizontálního a vertikálního směru. K tomuto odklonu často dochází při skenování dokumentu. Metody pro odhad šikmosti se děli do následujících hlavních tříd:

- analýza projekčních profilů
- Houghova transformace
- shlukování spojitých komponent
- korelace mezi přímkami

Tento proces je detailně popsán v technické zprávě [4]. Já používám k odhadu šikmosti Houghovu transformaci, i když díky povaze dokladů trochu jinak, než jak je to popsáno ve zmíněné zprávě, a úhel zjišťuji z obrysu dokladu a nikoliv přimo z textu.

2. Návrh aplikace

Tato kapitola obsahuje informace o použitých technikách na rozpoznání polohy dokumentu v obrázku a některých předpokladech, které jsem o dokladech učinil. Další postup, který je zde rozebrán je detekce důležitých míst v dokumentu, jedná se o oblasti obsahující text, jehož extrakce je cílem aplikace. Zároveň se zmiňuji o úspěšnosti a omezeních plynoucích z použitých metod, jakož i o jiných vyzkoušených postupech a jejich problémech, které vedly k jejich nevyužití. Nechybí ani průběh popisu zarovnávání buněk při aplikaci šablony.

2.1 Detekce obrysu dokladu

Detekovat polohu dokumentu v obrázku je nutné k úspěšnému pokračování a dokončení celého procesu. Při skenování mohlo dojít třeba k pootočení dokumentu, což by značně ztížilo detekci textových oblastí. V této oblasti jsem si dovolil učinit několik předpokladů, které vycházejí z práce s různými typy dokladů a jistých přirozených očekávání. Mé předpoklady jsou následující:

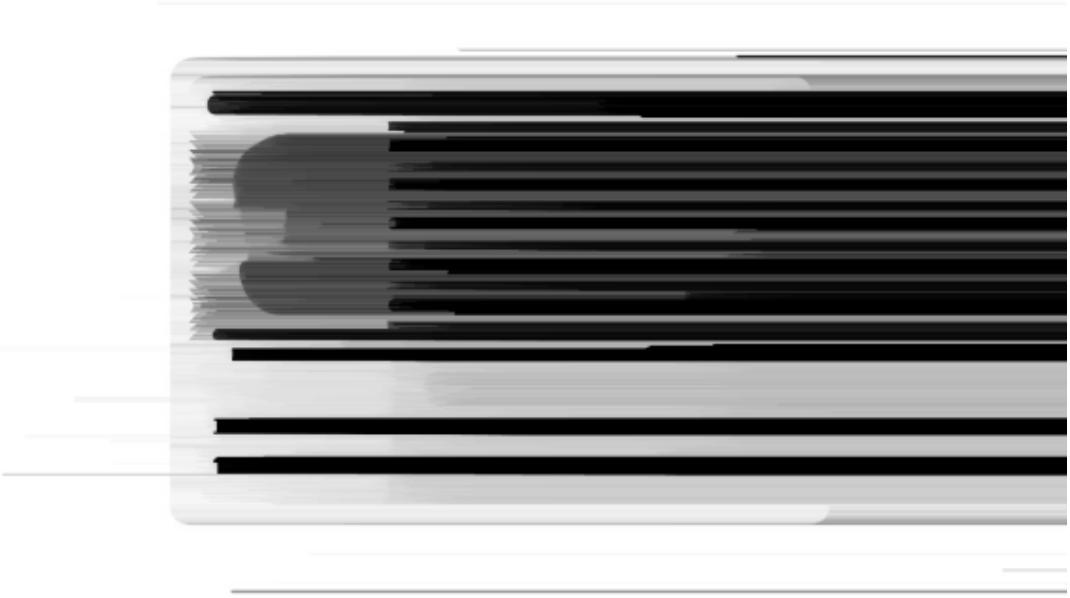
- Obrys dokumentu je tvořen obdélníkem.
- Směr textu v dokladu je pouze jediný, text má jednotnou orientaci.
- Tento směr je rovnoběžný s obrysem dokladu, případné otočení textu tedy odpovídá otočení celého dokladu.
- Doklad je naskenován na bílém pozadí, je tedy možné rozeznat jeho obrys.

Při splnění těchto předpokladů se mohu zaměřit na správnou detekci obdélníkového obrysu a automaticky tak získám i informaci o otočení textu. Odhaduji tedy šikmost dokladu tak, jak je popsaná v 1.4.1.

2.1.1 Předzpracování obrázku

Před samotnou detekcí obrysového obdélníku je potřeba vstupní obraz upravit tak, aby samotná detekce byla co nejpřesnější a nejúspěšnější. Ideální by bylo, aby v obrazu zůstal pouze hledaný obdélník a veškerý ostatní obsah by zmizel. Díky předpokladu, že je dokument naskenován na bílém pozadí, by mělo být možné oddělit od sebe pixely uvnitř a vně obdélníku. Snažil jsem se toho dosáhnout prahováním a nalezením takové hodnoty, aby většina pixelů dokladu byla označena jako objektové pixely a naopak vnější pixely aby byly označeny jako pozadí.

Následně jsem na takto upravený obraz chtěl pustit detektor hran, viz. 1.2.2, který by z vyplněného obdélníku ponechal pouze obvodové hrany. Zde jsem vyzkoušel Sobelův operátor i diferenciální detekci hran. Zejména díky nedostatečnému oddělení vnějších a vnitřních pixelů během prahování, kdy zůstalo buď velké množství vnitřních pixelů bílých, nebo naopak kdy jako objektové byly označeny i pixely vně dokladu, však výsledky nebyly uspokojivé. Proto jsem přišel s poněkud odlišným řešením, které má ale stejný základní cíl, a tím je oddělit pixely vně a uvnitř dokladu.



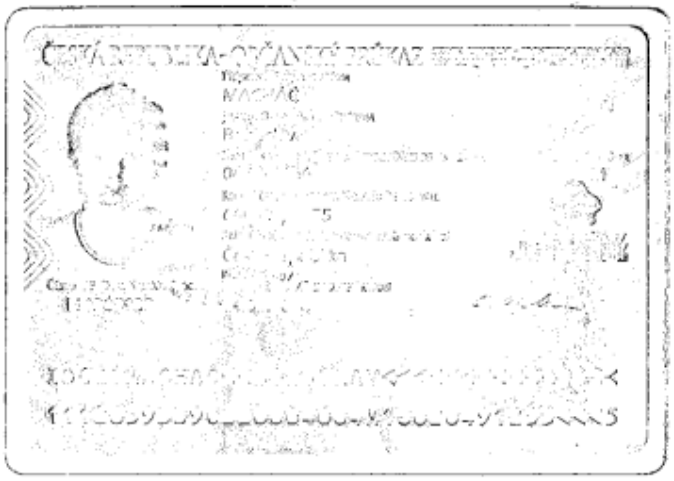
Obrázek 2.1: Zaplavování - horizontální zleva doprava

Zaplavování

Předpoklad o bílém okolí jsem využil tak, že postupuji od okrajů(zvlášť pro vertikální a zvlášť pro horizontální směr) a pokud pixel má nižší hodnotu, než jeho sousední pixel blíže k okraji, nastavím mu tuto hodnotu. Hodnota pixelů v daném směru tedy postupně klesá a nikdy neroste. Původně jsem postupoval z obou stran, kdy se skončilo uprostřed obrázku. Toto řešení vedlo k problémům v případech, kdy obě hrany dokladu v daném směru byly na jedné polovině obrázku, došlo totiž k tomu, že se druhá strana v pořadí podle směru postupu stala neviditelnou a nebyla detekována. Cílem je mít obrázek, kde bude mít obdélník výplň s hodnotami aspoň takovými, jaké jsou na okraji.

Detekce hran

Poté na obrázek pustím detektor hran, ale vždy jen v tom směru, ve kterém jsem postupoval předtím. Chci totiž detekovat pouze hrany kolmé na směr ve kterém postupuji. Pokud bych detekoval i rovnoběžné hrany, došlo by díky předchozí úpravě k výskytu velkého množství falešných hran, vzniklých vytvářením tmavé výplně vnitřku dokladu. Konkrétně je to vidět na obrázku 2.1, kde je výsledek daného postupu v horizontálním směru. Pokud bychom postupovali v tomto obrázku vertikálně, je vidět, že sousední pixely nemusí mít stejné, ba ani podobné hodnoty, a to i přesto, že je původně měly a žádná hrana se na takovém místě nenalézala. Pokud bychom na takovýto obrázek pustili detektor, který detekuje i vertikální hrany, nalezl by veliké množství těchto falešných hran a výsledek by byl nepoužitelný. Ze stejného důvodu je důležité, aby použitý detektor přihlížel pouze k okolním pixely ve směru postupu. To bohužel nesplňuje žádný z běžných detektorů hran a proto jejich nasazení nepřicházelo v úvahu.



Obrázek 2.2: Zaplavování - detekce hran

Jednoduchý detektor

Řešení je naštěstí vcelku jednoduché, stačí pro daný směr porovnávat hodnotu pixelu pouze s předcházejícím pixelem z tohoto směru a případný rozdíl označit jako hranu. Tento postup je rychlý a jednoduchý, a pro toto využití naprosto dostačující. Návíc se dá spojit se zaplavováním a tím celý proces ještě urychlit.

Nakonec jsem se rozhodl kroky zaplavování a detekce hran spojit. V okamžiku, kdy měním hodnoty pixelů směrem od kraje, mám již dostatek informací pro detekování hran. Na začátku nastavím referenční hodnotu jako hodnotu okrajového pixelu. Postupuji od kraje a jako váhu hrany nastavím rozdíl mezi hodnotou pixelu a referenční hodnotou, ale jen pokud má pixel hodnotu nižší, než je hodnota referenční, v takovém případě navíc nastavím referenční hodnotu na původní hodnotu pixelu. Jinak nastavím hodnotu pixelu na 0 a referenční hodnota se nemění.

Výsledky hranové detekce ze všech 4 směrů pak složím do jednoho obrazu. Výsledný obraz bude pravděpodobně obsahovat šumové hrany, ale jediné delší souvislé hrany by měly být právě hrany obrysové.

Výsledek celého procesu pro občanský průkaz je vidět na obrázku 2.2.

2.1.2 Možnosti detekce obdélníku

Po předzpracování obrázku už by nemělo být těžké najít obdélník obrysu dokladu. K nalezení obdélníku jsem se rozhodl využít Houghovu transformaci, popsanou v kapitole Teorie v subsekci číslo 1.3. V úvahu přicházely následující možnosti:

1. najít jednotlivé přímky, jejichž části tvoří obdélník a spočítat jejich průsečíky a poskládat tak dohromady parametry obdélníku

2. vyjádřit parametricky celý obdélník a pomocí Houghovy transformace hledat rovnou celý obdélník

3. využít zobecněné Houghovy transformace na principu porovnávání vzorů, za vzor vzít obdélník a hledat jej tímto způsobem

Možnost číslo 3 jsem vyloučil rovnou, jelikož neznám dopředu rozměry obdélníka, nebylo by ani možné vytvořit správný vzor, nehledě na to, že pokud vezmu v úvahu i otočení, celý proces by nebyl moc efektivní.

Možnost číslo 2 jsem zkusil implementovat, obdélník vyjádřil parametricky pomocí referenčního bodu, úhlu natočení a rozměrů, ovšem tento počet parametrů se ukázal pro rychlý běh aplikace jako moc vysoký a toto řešení bylo nepoužitelné. Později jsem jej ještě upravil pro případ, že znám alespoň rozměry obdélníku a mohu tak výšku a šířku nahradit jedním parametrem a to stupněm zvětšení, ale algoritmus i nadále běžel v řádu minut.

Varianta číslo 1 se tedy ukázala jako jediná použitelná a časová náročnost spojená s výběrem správných kandidátů a dopočítání potřebných údajů jako zanedbatelná v porovnání s ostatními metodami.

2.1.3 Proces detekce

Z principu Houghovy transformace a předpokladu, že obrázek obsahuje šum, plyne, že po aplikaci Houghovy transformace a vybrání takových přímek, které nabývají v akumulátoru lokálního maxima, dostanu více přímek, než žádoucí 4, které jsou potřebné k definici obdélníka. Nalezené přímky proto musím přefiltrovat a vybrat pouze ty 4 správné přímky, které tvoří hledaný obdélník.

Prvním filtrovacím krokem je minimální počet hlasů pro danou přímku, zjevně nemá cenu vybírat přímku s několika málo hlasy, i když nabývá lokálního maxima. Jako kritérium minimálního počtu hlasů beru polovinu součtu výšky a šířky obrázku. Je to dobrá výchozí hodnota, prověřená testováním, která odfiltruje většinu přímek. V případě, že se ukáže hodnota jako příliš vysoká, je snížena a proces se opakuje. Dalším filtrovacím krokem je vybrat z každého dostatečně malého okolí pouze přímku s nejvyšším počtem bodů. Dostatečně malé okolí znamená úhel mezi přímkami menší než 10 stupňů a současně rozdíl vzdáleností od referenčního bodu menší, než 20% menšího z rozměrů obrázku. Tyto hodnoty byly otestovány, a i když by zřejmě šlo dosáhnout většího vyfiltrování přímek už v tomto kroku, z důvodu opatrnosti byly ponechány tyto hodnoty. Pokud jako dostatečně malé okolí beru oblast řádově menší než je doklad, nemohu jednou obrysovou přímkou vyřadit jinou obrysovou přímku. Chci tedy, aby kratší strana dokladu měřila více, než je pětina kratšího rozměru obrázku, což by měly doklady běžně splňovat, případně je možné obrázek oříznout před použitím v aplikaci. být bez problémů. Případným dalším filtrem je požadavek na přímku, aby k ní ve vybraných přímkách existovala ještě jedna přímka rovnoběžná a 2 přímky kolmé. Zde určitě nevyřadím správné přímky.

Pokud je stále přímek více než potřebný počet(4), roztřídím přímky podle úhlů a vyberu ten úhel, který má přímka s největším počtem bodů a úhel o 90 stupňů menší/větší tak, aby přímky s těmito úhly byly na sebe kolmé. V případě, že stále mám moc kandidátů, vyberu 2 přímky s největší vahou pro každý z 2 úhlů.

Nyní jen najdeme průsečíky, sestavíme úsečky, a zjistíme případný úhel otočení. Pokud v jakékoliv fázi filtrování máme méně než 4 kandidáty, nastavíme nižší

minimální počet hlasů pro první filtr a začneme od začátku. Pokud již nelze hranici snížit a stále jsme neuspěli, ohlásíme uživateli neúspěch a necháme ho vybrat obrys manuálně.

2.1.4 Využití existující šablony

Každá šablona vzniká na určitém exempláři naskenovaného dokumentu. V případě, že jde o typ dokumentu, pro který již máme vytvořenou šablonu a chceme ji na něj použít, je navíc nutné detekovat posunutí dokladu v obrázku vůči poloze exempláře, ze kterého byla šablona vytvořena. Naopak máme k dispozici informaci o šířce a výšce obrysu, doklad ale může být naskenován v jiném měřítku, a proto tyto údaje nemůžeme použít přímo pro usnadnění detekce obrysu. Pokud má obrázek opravdu jiné měřítko, lze pomocí těchto údajů alespoň upravit šablonu tak, aby pasovala i na tento obrázek.

2.2 Detekce význačných oblastí

Po zjištení obrysu dokumentu přichází na řadu detekce textových oblastí. V případě, že nalezený obrys dokumentu není rovnoběžný s horizontální a vertikální osou, je nejprve potřeba dokument otočit zpět do vodorovné pozice, aby následně použité postupy byly účinné, zejména se to týká morfologických operací. Díky povaze dokumentů by mělo být možné jednotlivé pixely dokladu snadno rozdělit na pozadí a popředí, kde text je popředím a tedy výraznější než okolí.

Prvním úkolem bude tyto dvě skupiny pixelů oddělit. K tomu se hodí již zmíněné prahování 1.1.1. Díky možnosti rozdělení do 2 tříd je ideální použít shlukovací metodu na určení prahu, já konkrétně používám Otsuovu metodu. Nyní se již zabývám pouze oblastí samotného dokladu, což zlepší výsledek, protože do oddělování popředí a pozadí nezasahuje samotné pozadí kolem dokumentu. V ideálním případě mám po nalezení správného prahu a aplikaci prahování k dispozici obrázek, kde jsou jednotlivá písmena černá, oddělená bílým pozadím. Ve skutečnosti se ukázalo, že písmena většinou nejsou úplně souvislá a jsou rozdělena na více částí. Do třídy s textem jsou většinou přiděleny ještě šumové pixely, které, pokud se neslejí s textem, jsou eliminovány kritériem pro minimální velikost komponenty.

Program se zajímá o detekci textových oblastí z pohledu fyzické struktury, nikoliv významového hlediska. Jako význačná oblast může být chybně detekován třeba popisek opravdové význačné oblasti, pokud je přiřazen do stejné třídy pixelů. Stejně tak může být označen například oddělující pruh v barvě, nebo spíše intenzitě, podobné textu. Odstranění takovýchto oblastí zůstává na uživateli. Možné rozšíření aplikace o automatické rozlišení správných oblastí je nastíněno v závěru práce.

2.2.1 Slévání oblastí

Po prahování přichází na řadu morfologické operace, popsané v subsekci 1.1.3, konkrétně dilatace. Je potřeba spojit jednotlivé části písmen a poté jednotlivá písmena do souvislých oblastí odpovídajících slovům, datovým polím, popřípadě

řádkům. K tomu je zapotřebí znát jisté vlastnosti dokumentu, zejména mezeru mezi písmeny, slovy, řádky. Zde, narozdíl od stránky plné souvislého textu, lze pouze těžko použít běžně aplikované statistické metody pro určení těchto mezer. Detekujeme tedy komponenty s nějakým konkrétním nastavením parametrů a případně jej necháme uživatele upravit. Zde je těžké nějak určit správné parametry pro obecné využití, proto se očekává vstup od uživatele, který hodnoty parametrů snadno upraví podle výsledku prvního pokusu.

Z parametrů textu už snadno zjistíme potřebné nastavení parametrů dilatace a jejím aplikováním získáme požadované oblasti. Pro dilaraci se používají strukturovací elementy ve tvaru horizontální a vertikální úsečky, s rozměry danými parametry obrázku. To jsou ty těžko zjistitelné, které nám pomůže nastavit uživatel. Výsledné oblasti jsou nalezeny pomocí hledání souvislých komponent. Procházím výsledný obrázek po jednotlivých pixelech a eviduji, které jsem již navštívil. Pokud narazím na do té doby nenavštívený objektový pixel, najdu vyhledáváním do šířky po objektových pixelech celý souvislý tvar. Z něj snadno získám obdélník, který obsahuje celý tvar.

Ještě zkontroluji, jestli nalezený tvar splňuje požadované vlastnosti, konkrétně jde o minimální velikost. Komponenta o velikosti několika málo pixelů, jen těžko může nést nějaké informace a mnohem pravděpodobněji se jedná o šum. Pokud by se nejednalo o šum, přichází v úvahu třeba tečka nad písmenem, ale v takovém případě je potřeba upravit parametry detekce a tečku detekovat jako součást oblasti s písmenem.

2.2.2 Alternativní postup

Nejprve jsem vyzkoušel poněkud jiný přístup detekce komponent. Místo dilatace jsem nalezl jednotlivé komponenty hned po provedení prahování a podle daných parametrů(maximální vzdálenost v horizontálním a vertikálním směru) je postupně spojoval dohromady. Výsledek je u obou přístupů podobný, ale využití dilatace vedlo ke zvýšení rychlosti výpočtu a je pro danou úlohu přirozenější.

2.3 Zarovnání textových oblastí

Při extrakci dat se může stát, že jednotlivé buňky z datové šablony nebudou přesně sedět a text na obrázku se do nich nevejde celý. Důvodem může být například to, že při vytváření šablony byl v dané buňce kratší text, než na dokladu, ze kterého chceme data extrahovat. Na toto je možno myslet i při tvorbě šablony a nechat vytvářeným buňkám určitou rezervu. Případně může nastat jisté posunutí i ve vertikálním směru, způsobené nepřesným napasováním šablony. Aby nebylo nutné při každé takové situaci upravovat šablonu, je k dispozici jednoduchá metoda na automatické zarovnání oblastí. Ta má ale svá omezení a nebude vždy fungovat. Jako mnohem jednodušší řešení se jeví nadefinovat šablonu pořádně, zvláště pokud bude používaná pro větší množství dokladů, a předejít tak podobným situacím, či je alespoň minimalizovat.

2.3.1 Průběh zarovnání

Pro každou buňku si vezmu ke zpracování její okolí, které tvoří asi polovina rozměrů buňky na každou stranu. Pokud je posunutí mezi buňkou a šablonou větší, nebude tento postup fungovat. V případě, že nastane taková situace, nemusí už ani být možné rozhodnout, pro kterou oblast je šablona určena, protože může například zabírat větší část jiného řádku. Taková situace by ale nastat vůbec neměla, pokud ano, je nejpravděpodobnější příčinnou špatný návrh šablony a k vyřešení bude zřejmě potřeba šablonu upravit.

Poté pomocí parametrů uložených v šabloně aplikuji na vybranou oblasti dilataci. Následně začínám od prostředka hledat objektový pixel prohledáváním do šířky. Po jeho nalezení provedu detekci souvislé komponenty, shodně jako při hledání význačných oblastí. Rozměry a polohu buňky upravím podle nalezené komponenty.

2.3.2 Alternativní metoda zarovnání

Dřive jsem zkoušel pro každou buňku spočítat statistiku průměrné vzdálenosti mezi objektovými pixely a z ní získat parametry pro dilataci, ale přístup s využitím parametrů použitých při vytvoření šablony je efektivnější a měl by dávat lepší výsledky. Tato metoda ale zůstává používána pro případy, kdy byla šablona vytvořena ručně, bez použití automatické detekce a tudíž i bez nastavení potřebných parametrů.

3. Architektura aplikace

Tato kapitola se zaměřuje na samotnou aplikaci, popisuje její strukturu, jednotlivé části a implementaci postupů z minulé kapitoly. Důležitou součástí aplikace je grafické rozhraní, které umožňuje intuitivní použití i nezkušeným uživatelům, kteří nemají žádné znalosti o digitálním zpracování obrazu či počítačovém vidění.

3.1 Implementační prostředí

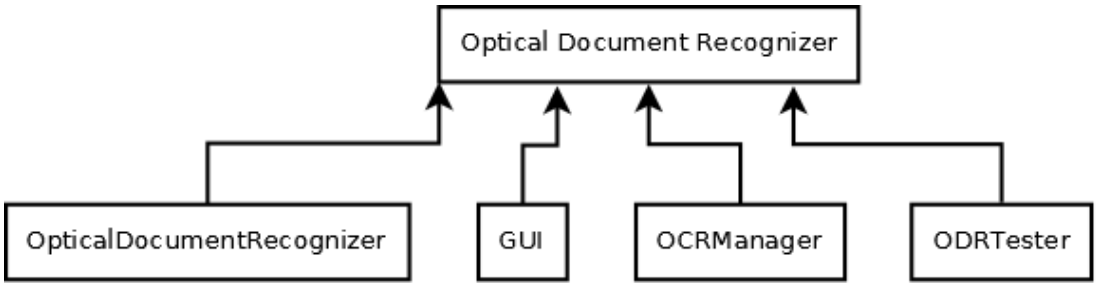
Jako prostředí pro implementaci aplikce jsem zvolil jazyk C# a prostředí .NET. S tímto moderním jazykem mám největší zkušenosti a chtěl jsem napsat co nejkvalitnější aplikaci a soustředit se na funkce aplikace a ne vývoj zpomalovat učením nového jazyka. Svou roli sehrál i fakt, že C# není v této oblasti příliš používaný, a tak jsem chtěl i já alespoň malou měrou přispět k rozšíření C# aplikací v této oblasti. Dalším velkým plusem je technologie Windows Presentation Foundation, umožňující pomocí jazyka XAML snadno vytvořit uživatelsky přívětivé prostředí, která má z programátorského hlediska tu výhodu, že odděluje funkčnost aplikace a její vzhled. Nevýhodou plynoucí z tohoto výběru je použitelnost pouze v prostředí operačního systému Windows.

3.2 Struktura programu

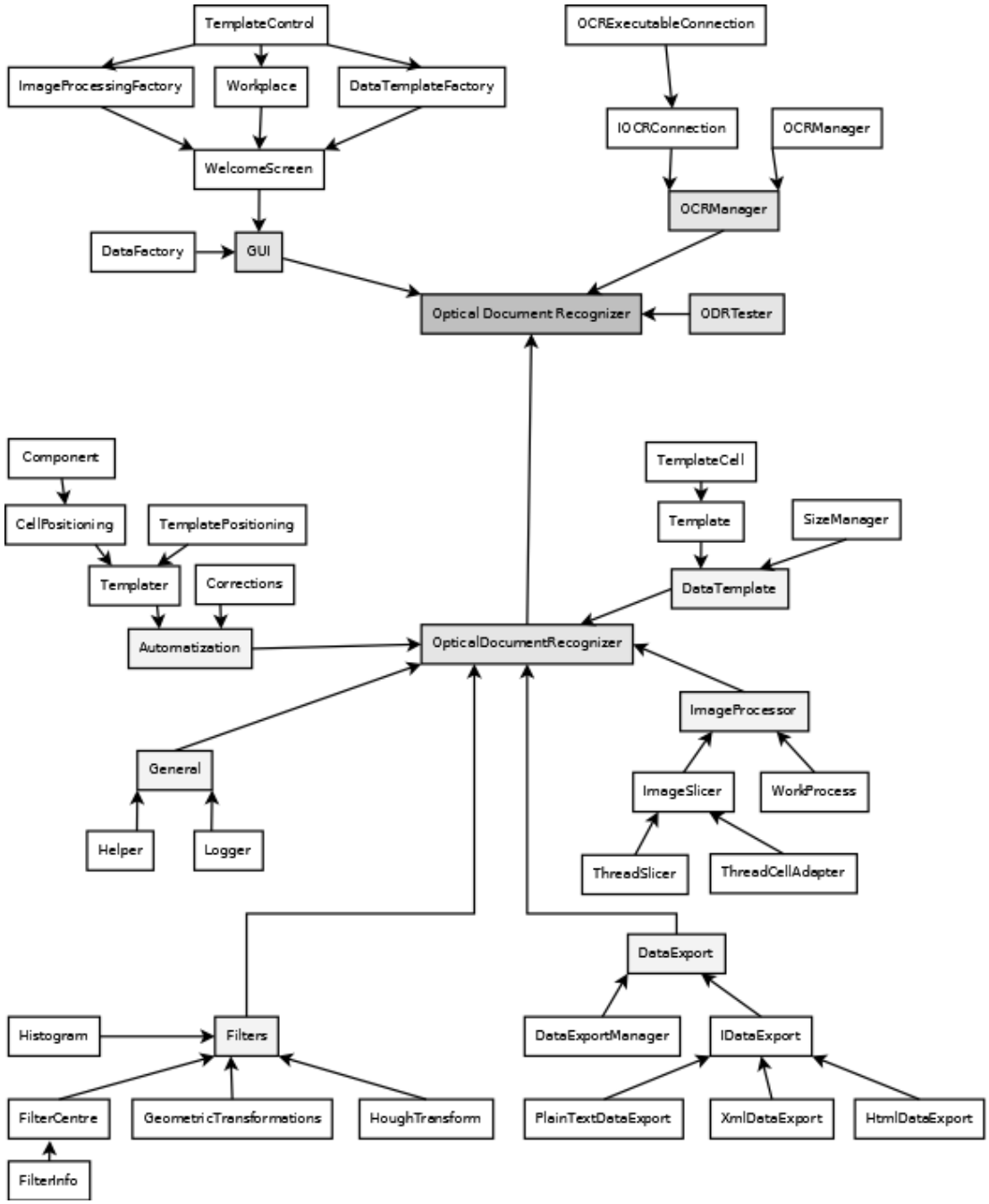
Program je rozdělen do několika hlavních částí, jak je znázorněno na UML diagramu 3.1.

Podrobnou strukturu, obsahující všechny důležité třídy aplikace pak zachycuje diagram 3.2.

Toto rozdělení aplikace vyplynulo z její logické struktury. GUI opravdu obsahuje pouze grafický kabátek aplikace, veškeré algoritmy jsou v jiných částech projektu. Vyjímku tvoří implementace vlastní funkcionality GUI, zejména se jedná o část s definicí šablony, kde je potřeba pracovat s jednotlivými buňkami, zobrazit je v aplikaci a snadno s nimi manipulovat.



Obrázek 3.1: Optical Document Recognizer - hlavní části



Obrázek 3.2: Optical Document Recognizer - podrobná struktura

Obrázek 3.3: OpticalDocumentRecognizer - tvorba datové šablony

3.3 GUI

Grafické rozhraní je navrženo tak, aby umožňovalo co nejsnadnější práci. Na začátku se uživateli spustí uvítací rozcestník, kde si vybere z nabízených možností a aplikace se poté přepne do vybraného módu. Z celé aplikace je přístupné jednoduché menu, které umožňuje návrat na rozcestník, ukončení aplikace, dále pak přepínání pracovního nastavení aplikace a zobrazení stručných informací o aplikaci. Aplikace je pro přehlednost rozdělena do 3 hlavních módů, v nichž je možné vykonávat rozdílné činnosti.

Prostředí umožňuje přepínání mezi 3 nastaveními podle úrovně uživatele. Základní nastavení obsahuje pouze hlavní části - tvorbu šablon a extrakci dat, v pokročilém nastavení je navíc zpřístupněn editor a ve vývojářském nastavení jsou navíc některé pomocné informace v jednotlivých částech, ponechané zejména pro usnadnění případného budoucího rozšiřování aplikace.

3.3.1 GUI Tvorba datové šablony

Po vybrání konkrétního obrázku se zobrazí obrazovka s obrázkem a levým a pravým panelem, tak jak ukazuje obrázek 3.3. V levém panelu je menu pro tvorbu šablony. V horní části je možné přepínat mezi jednotlivými režimy tvorby buněk a ve spodní části potom otevřít již existující šablonu nebo naopak uložit právě vytvořenou. Také se zde dá nechat automaticky vygenerovat obrys, či případně obrys vyznačit manuálně. V pravém panelu je zobrazen název šablony a v dolní části potom tabulka s vlastnostmi vybrané buňky.

Průběh práce v tomto módu ukazuje workflow diagram 3.4.

Režimy tvorby buňek

Rozhraní pro tvorbu šablony lze přepínat mezi následujícími režimy:

- Normal - v tomto režimu lze pouze vybírat jednotlivé buňky
- New - při zmáčknutí levého tlačítka myši lze tažením a následným uvolněním tlačítka tvořit nové buňky
- Move - lze buňky vybírat a tažením přesunovat
- Delete - kliknutím na buňku dojde k jejímu vymazání
- Pinpoint - speciální režim, aktivován tlačítkem Select Border slouží k manuálnímu vybrání obrysu dokladu stejným způsobem jako je tvorba buňky

Vlastnosti buňky

Tabulka vlastností buňky obsahuje údaje o vybrané buňce, jsou to jméno buňky,datový typ, poloha levého horního rohu(X a Y souřadnice) a rozměry(šířka a výška). Tyto údaje jsou editovatelné a změna se okamžitě projeví v okně se zobrazením šablony.

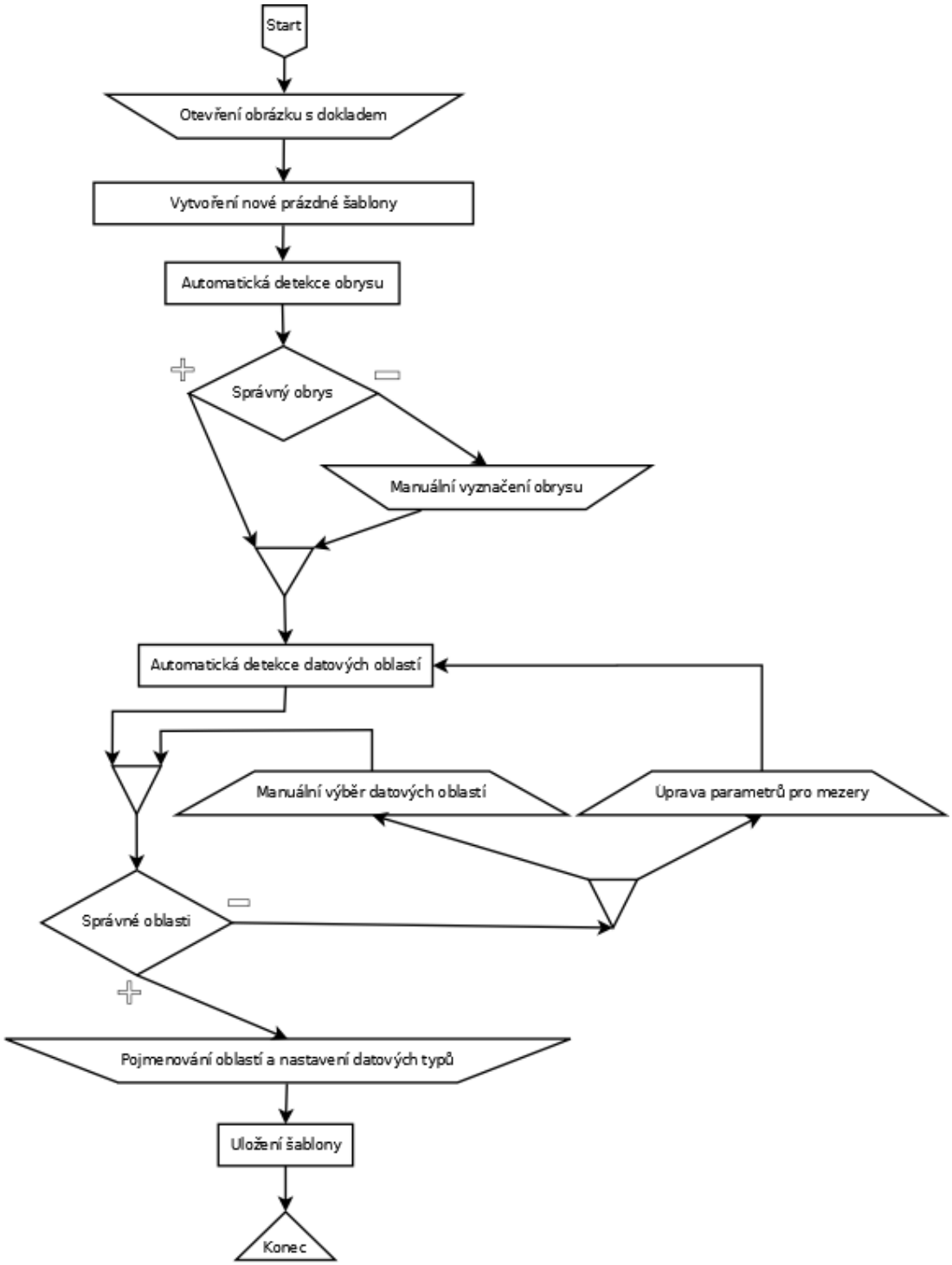
3.3.2 GUI Extakce dat

V okně se zobrazí vybraný obrázek ze kterého chceme extrahovat data. Ovládací panel je v horní části. Obsahuje seznam dostupných šablon a také seznam procesů pro předzpracování obrázku. Dále je zde tlačítko pro automatické zarovnání buněk šablony. A nakonec tlačítka pro extrakci textu z obrázku, jedno které provede extrakci a zobrazí okno s výběrem možností pro export a druhé, testovací, které pouze zobrazí extrahovaná data. Obrázek 3.5 zobrazuje aplikaci v režimu extrakce dat.

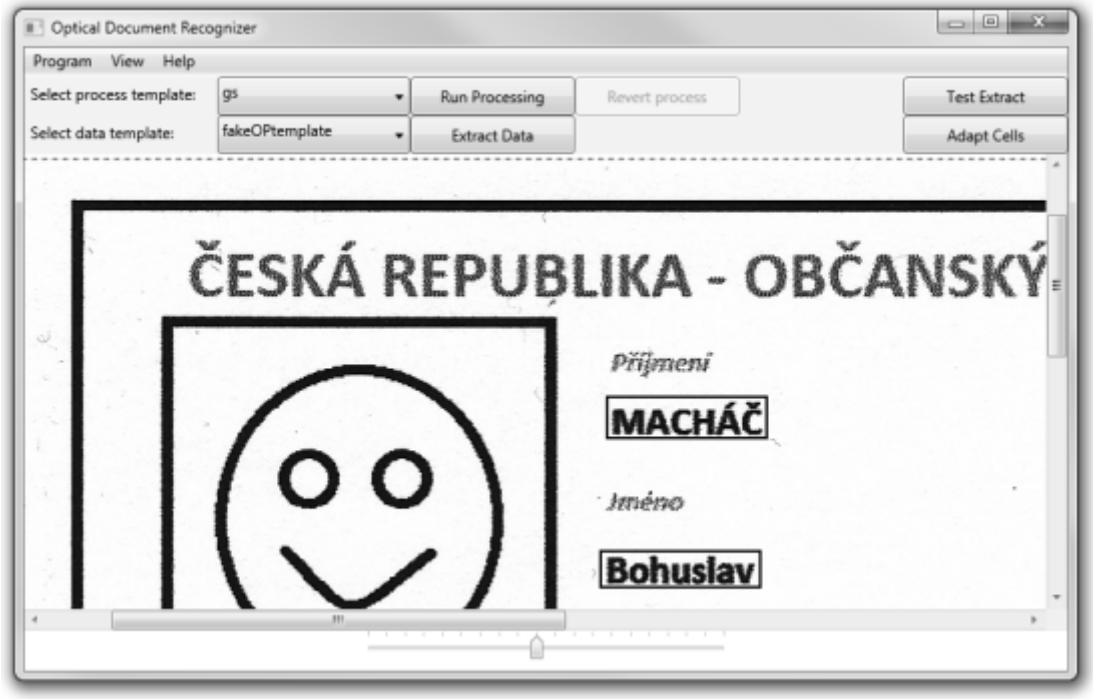
Průběh práce v tomto módu ukazuje workflow diagram 3.6.

3.3.3 GUI Editor

Editor slouží zejména pro tvorbu a správu procesů pro předzpracování obrazu. Jednotlivé filtry jsou zobrazeny v levé horní části. Po vybrání se pod filtrem zobrazí dostupné parametry a tlačítko pro aplikaci filtru. Pod ním postupně vzniká seznam použitých filtrů v pořadí, jak šli za sebou, i s hodnotami parametrů. V dolním menu lze načíst již existující proces, uložit stávající, nebo otevřít jiný obrázek, či upravený obrázek uložit. Během vývoje byl hodně používán na testování účinků jednotlivých filtrů a lze jej i nadále využít jako pracovní nástroj na úpravu obrázků pomocí filtrů. Na obrázku 5.2 je zobrazena aplikace v módu editoru.



Obrázek 3.4: Workflow diagram - tvorba šablony



Obrázek 3.5: OpticalDocumentRecognizer - extrakce dat

3.4 OpticalDocumentRecognizer

Jádro aplikace, obsahující veškerou vnitřní funkcionalitu. V rámci aplikace se jedná o samostatný projekt, který se kompiluje do dynamické knihovny. Tento projekt je nadále rozčleněn na několik podčástí, jak je znázorněno na části diagramu 3.2.

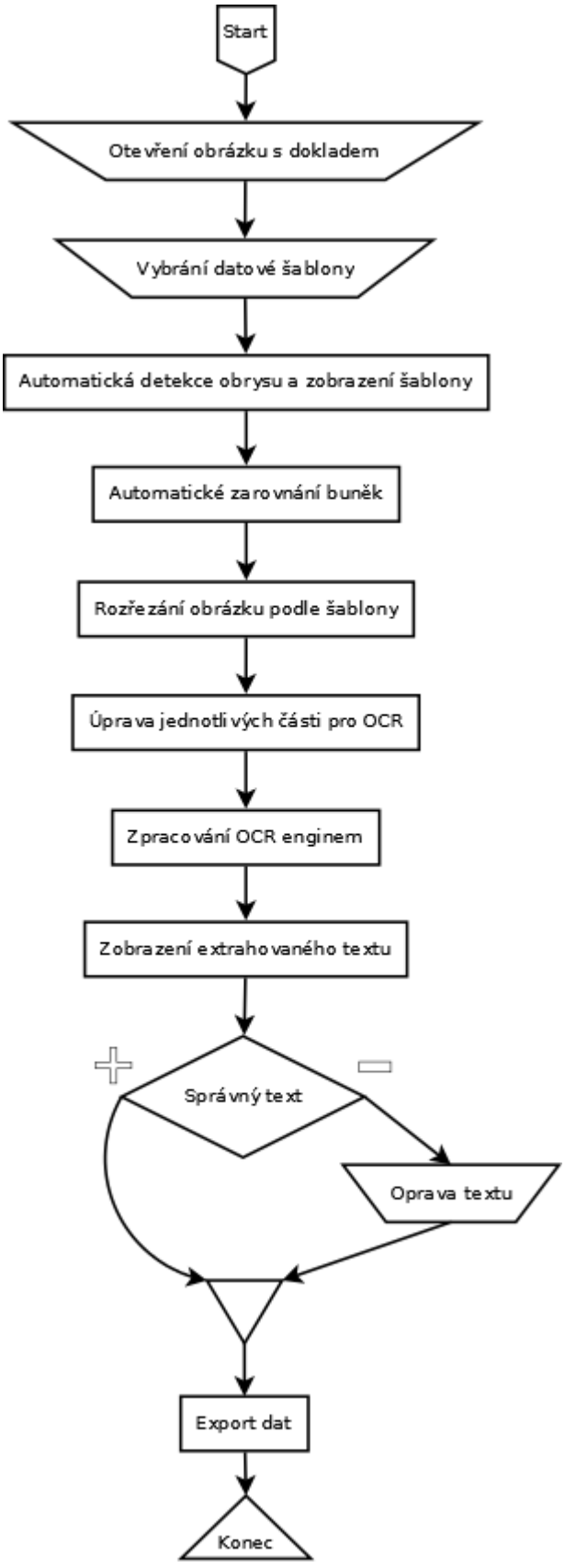
3.4.1 Automatization

Tato část aplikace obsahuje vše, co souvisí s automatizací v aplikaci, zejména detekci obrysu a datových polí. Tyto procesy zastřešuje třída `Templater`. Metoda *CreateBorder* hledá obrys v dokumentu a případně jej pootočí, pokud je to potřeba. Metoda *FindComponents* slouží k nalezení datových oblastí na základě zadaných parametrů.

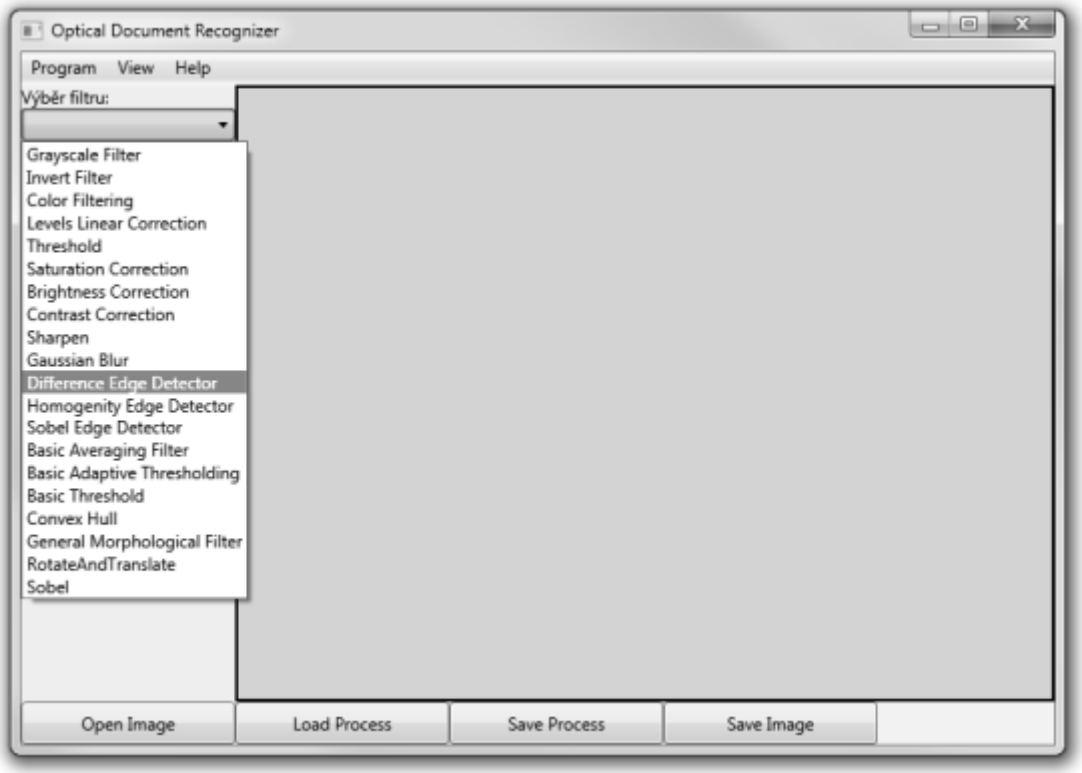
Třída `CellPositioning` řeší zarovnání buněk šablony pro konkrétní obrázek tak, jak je to popsané v sekci 2.2. Třída `Component` slouží pro reprezentaci obalu souvislé komponenty při hledání datových oblastí. Má implementován komparátor podle x souřadnice a další metody pro snadnou práci s komponentami. Třída `Corrections` má na starosti automatické opravy výstupu z OCR. Jde pouze o několik odpozorovaných chyb a situací, které jsou definované ve zvláštním XML souboru.

Třída `TemplatePositioning`

Zabývá se již konkrétně hledáním obrysu, a to již od fáze předzpracování obrázku, který nejprve v případě potřeby převede na šedotónový, poté detekuje hrany



Obrázek 3.6: Workflow diagram - extrakce dat



Obrázek 3.7: OpticalDocumentRecognizer - editor se seznamem filtrů

pomocí postupu popsaném v subsekci 2.1.1 a nakonec obrázek invertuje a připraví tak na další fázi, Houghovu transformaci.

V metodě *FindBorder* se provede Houghova transformace a následně se filtrují nalezené přímky tak, aby se dosáhlo nalezení obrysového obdélníka, který také tato metoda vrací.

Metoda *AnalyzeInput* se využívá v případě, že hledám obrys pro dokument na který už mám nadefinovanou šablonu a cílem je tedy zjistit případné posunutí a natočení dokladu vůči šabloně.

3.4.2 DataExport

Požadavky na formu výstupu dat z aplikace mohou být velice rozličné, a proto byl oddíl pro export navrhnut tak, aby bylo snadné v případě potřeby doprogramovat další možnosti výstupu.

Interface `IDataExport`

Toto rozhraní definuje všechny metody potřebné pro přidání další možnosti exportu dat. Důležitá je vlastnost *Type*, pokud má hodnotu *File*, aplikace zavolá okno s vybráním souboru pro uložení a použije metodu s předáním streamu. Jinak volá *SaveOutput* bez streamu, pouze s definicí šablony a daty. *GenerateOutput* slouží k vygenerování výstupního textu bez uložení do souboru.

Třída `DataExportManager`

Manažer, který poskytuje přístup ke všem dostupným možnostem exportu a volá jejich metody přes interface `IDataExport`.

Třída `PlainTextDataExport`

Jednoduchá implementace exportu do textového souboru. Pro každou buňku vypíše na řádek její jméno a pomlčkou oddělenou hodnotu.

Třída `XmlDataExport`

Implementace exportu v XML formátu. Výstup je obalen v elementu s názvem šablony. Jednotlivé buňky jsou reprezentovány elementy s jejich názvem a uvnitř elemetů jsou extrahované hodnoty.

3.4.3 DataTemplate

Tento oddíl se zabývá reprezentací datové šablony v paměti při běhu aplikace. Na disku je šablona uložená ve formátu XML, podle XML schématu `OpticalDocument-Recognizer:DataTemplate:DataTemplate.xsd`, ke kterému je pomocí nástroje *xsd.exe* vygenerovaná třída `DataTemplate`.

Třída `Template`

Pro snadnější práci s šablonou v aplikaci je dále navržena ještě třída `Template`. Mezi její funkce patří možnost načtení a uložení ze třídy `DataTemplate`, která se sama načítá/ukládá pomocí objektu *XmlSerializer*. Její struktura je více přizpůsobena použití v aplikaci, narozdíl od generované třídy `DataTemplate`, která všechny kolekce reprezentuje jako pole.

Třída `TemplateCell`

Třída pro reprezentaci jedné buňky datové šablony. Údaje o rozměrech jsou uložené v objektu `Rectangle`, který je zároveň grafickým znázorněním buňky v aplikaci.

Třída `SizeManager`

Tato třída má za úkol vypořádat se s různými velikostmi naskenovaných obrázku a má přizpůsobit šablonu tak, aby na ně pasovala.

3.4.4 Filters

Oddíl Filters je ze spodní vrstvy programu, jsou v něm implementovány používané filtry a také je k němu připojena externí knihovna *AForge.NET*, která obsahuje další používané filtry. Z filtrů, které jsou z knihovny *AForge.NET* se používají zejména následující:

- Grayscale BT709 - převádí barevný obrázek na šedotónnový

- Invert - invertuje barvy v obrázku
- Threshold - provádí prahování šedotónového obrázku, ale zanechává formát 8bpp

Další filtry této knihovny jsou přístupné v editoru. Nyní se zmíním detailněji o jednotlivých mnou implementovaných filtrech a třídě `FilterCentre`.

Třída `FilterCentre`

`FilterCentre` je hlavním objektem pro práci s filtry. Obsahuje wrappery pro vybrané metody z knihovny *AForge.NET* a většinu mnou implementovaných filtrů. Poskytuje GUI editoru informace o dostupných filtrech a informace o vybraném filtru pomocí předání objektu třídy `FilterInfo`. Ta obsahuje jméno filtru a jeho parametry, tj. jejich typy a omezení na hodnoty.

Morfologické operace

Z morfologických operací jsou implementovány eroze a dilatace, pomocí společné metody `GeneralMorphological`. Dále pak zeštíhlení a pomocí něj metoda na hledání konvexního obalu objektů. Více o morfologických operacích v subsekci 1.1.3.

Třída `HoughTransform`

Tato třída obsahuje implementaci Houghovy transformace pro hledání přímek v obrázku, která je popsaná v sekci 1.3, a také všechny metody, které přímky filtrují a skládají z nich obrys dokladu. Nechybí ani pomocná metoda pro vykreslení nalezených přímek, která sloužila pro testování.

Třída `Histogram`

Základem je metoda pro výpočet histogramu obrázku. Informace o histogramu jsou v subsekci 1.1.2. Dále jsou k dispozici metody k určení vhodné hranice pro prahování z histogramu. Mezi ně patří metoda *GetThreshold*, která spočítá hranici tak, aby oddělila parametrem udaný počet pixelů se zadanou odchylkou od zbytku.

Součástí je i implementace Otsuovy metody, popsané v subsekci 1.1.1. V implementaci se využívá rychlejší výpočet pomocí *maximalizace rozptylu mezi třídami.*

Ostatní třídy

Třída `GeometricTransformations` má metodu pro posun a otočení obrázku. Ta se využívá v případech, kdy je doklad na obrázku pootočen nebo když je potřeba nasadit již existující šablonu na doklad.

Třída `Line` je pomocná třída pro reprezentaci přímky, definované pomocí obecného tvaru rovnice, a používá se při hledání obdélníkového obrysu.

Již zmíněná třída `FilterInfo` obsahuje informace o filtru pro GUI, které z nich generuje formulář pro zadání parametrů pro daný filtr.

3.4.5 ImageProcessor

Hlavní součástí tohoto oddílu je třída `ImageSlicer`, jejímž hlavním úkolem je vyřezat z obrázku oblasti odpovídající jednotlivým datovým buňkám a ty pak předat OCR enginu k extrakci textu. Další metoda implementuje zarovnání buněk šablony na daný obrázek.

Oba tyto procesy jsou pouštěné ve vlastním vlákně, což zajišťují třídy `ThreadSlicer` a `ThreadCellAdapter`.

Ještě sem patří třída `WorkProcess`, která slouží pro reprezentaci procesu pro zpracování obrazu v paměti. Na disk se proces ukládá ve formátu XML, jehož strukturu určuje XML schéma *Process.xsd* a k uložení se používá generovaná třída `Process` a třída *System.Xml.XmlSerializer*.

3.4.6 Ostatní části

Oddíl *General* obsahuje třídy a funkce používané na více místech aplikace. Třída `Logger` slouží k udržování logu aplikace a jeho případnému ukládání. Ve složce *XML Schema* jsou jednotlivá schémata definující strukturu XML souborů a k nim vygenerované C# třídy reprezentující datové šablony, seznam filtrů pro editor, automatické opravy a procesy zpracování obrazu. XML soubor *Corrections.xml* obsahuje seznam automatických oprav a soubor *FilterList.xml* seznam filtrů dostupných pro použití v editoru.

3.5 OCRManager

Tato část aplikace se stará o integraci externího OCR enginu do aplikace. Umožňuje snadné připojení enginu pomocí implementace jednoduchého rozhraní `IOCRConnection`. V aktuální verzi aplikace je funkční připojení k enginu Tesseract, ve verzi 3.0, konkrétně k jeho command line verzi. Připojení je zajištěno třídou `OCRExecutableConnection`, která umožnuje snadnou výměnu za jiný command line OCR engine.

Původně bylo dostupné připojení k Tesseractu verze 2.0, které fungovalo pomocí C# wrapperu Tessnet 2.0. Tato verze ale nepodporovala češtinu, novější verze Tessnetu není dostupná a Tesseract 3.0, který češtinu podporuje, nemá stejné rozhraní a nelze jej využít v Tessnetu 2. Tyto důvody a hlavně umožnění snadnější výměny OCR enginu vedly ke stávající podobě připojení.

3.6 ODRTester

Testovací projekt, obsahující několik jednoduchých testů, kterými jsem zkoušel a ladil funkčnost použitých postupů při práci s histogramem, při aplikaci Houghovy transformace a také při automatické detekci obrysu a textových oblastí. Ponechán pro případný další vývoj.

3.7 Datová šablona

Každý typ dokladu má svoji datovou šablonu. Dokonce jeden dokument může mít i více šablon, pokud například v jednom případě použití chceme získat jen omezenou sadu informací a v jiném stojíme o všechny informace. Datová šablona se ukládá na disk ve formátu XML. V aplikaci je pro ni vytvořené XML schéma.

3.7.1 Obsah šablony

Šablona obsahuje zejména informace o jednotlivých datových oblastech, ale také řídící informace pro usnadnění použití šablony a zvýšení kvality automatických výstupů.

Mezi řídící informace patří:

- rozměry a poloha obrysu - usnadňuje nalezení správného obrysu, umožňuje sesazení šablony na doklad
- PPI(pixel per inch) obrázku, na kterém byla šablona vytvořena - umožňuje správně měnit velikost šablony v závislosti na kvalitě obrázku s dokladem
- horizontální a vertikální mezera mezi komponentami nastavená při tvorbě šablony - usnadňuje vyhledání správných datových oblastí a jejich sesazení s šablonou

Dále šablona udržuje informace o jednotlivých buňkách. O každé buňce konkrétně: jméno, datový typ, polohu vůči levému hornímu rohu obrysu a rozměry.

3.7.2 Ukázka uložené šablony

Pro lepší představu následuje krátká ukázka konkrétní datové šablony, význam jednotlivých elementů by měl být zřejmý z předchozího popisu.

```
<?xml version="1.0"?>
<DataTemplate Name="OPTemplate" xmlns="
    OpticalDocumentRecognizer:DataTemplate:DataTemplate.xsd
    ">
  <Pinpoint X="249" Y="74" Width="829" Height="586" />
  <Cell>
    <Name>Prijmeni</Name>
    <DataType>Text</DataType>
    <X>273</X>
    <Y>302</Y>
    <Height>22</Height>
    <Width>151</Width>
  </Cell>
\ldots
<Properties>
    <PPI>
      <Horizontal>96</Horizontal>
      <Vertical>96</Vertical>
```

```
    </PPI>
    <Space>
        <Horizontal>12</Horizontal>
        <Vertical>6</Vertical>
    </Space>
  </Properties>
</DataTemplate>
```

3.8 Proces pro zpracování obrazu

Tyto procesy je možné vytvářet v editoru. Původní záměr byl takový, že ke každému dokladu bude existovat několik procesů, které obrázek s dokladem upraví tak, aby vyhovoval aplikaci a ta si s ním byla schopná poradit. Během vývoje a testování se ukázalo, že je možné použít jeden obecnější postup pro většinu dokumentů a procesy tedy při běžné práci nejsou potřeba. V různých specifických situacích ale mohou být stále přínosné, a tak zůstaly v aplikaci přístupné pokročilým uživatelům.

Pokud by bylo potřeba zpracovat větší množtví dokladů naskenovaných na stejném přístroji se stejnými parametry a tím pádem s velice podobnou kvalitou obrazu, je možné využít speciální proces ke zvýšení kvality všech těchto dokumentů. Nebo pokud by jiná skupina dokumentů trpěla nějakou stejnou chybou obrazu, kterou by bylo možné odstranit pomocí dostupných filtrů, je opět možné vytvořit speciální proces a použít jej při zpracování takové skupiny.

3.8.1 Reprezentace procesu na disku

Procesy se podobně jako datové šablony uchovávají na disku ve formátu XML. I zde je vytvořeno XML Schéma. V souboru jsou postupně jednotlivé kroky procesu, reprezentované identifikátorem použitého filtru a hodnotami jeho parametrů. Krátký proces může vypadat třeba následovně:

```
<Process Name="ColorToBin" xmlns="
    OpticalDocumentRecognizer:ProcessTemplate:Process.xsd">
  <Step Filter="Grayscale">
    <Parameter Name="Common" Value="BT709" />
  </Step>
  <Step Filter="ThresholdX">
    <Parameter Name="Threshold" Value="155" />
  </Step>
</Process>
```

Tento proces převede barevný obrázek na binární. Je vidět, že je potřeba specifikovat přesné hodnoty, např. pro prahování. Proto je proces vhodný pro použití na velmi podobnou skupinu dokladů.

4. Příručka programátora

V této kapitole představím aplikaci z pohledu programátora, usnadním orientaci v kódu a vysvětlím, jak lze snadno změnit některé důležité věci.

4.1 Struktura zdrojových souborů

Celá aplikace tvoří jedno *solution* Microsoft Visual Studia 2010.

GUI je samostatný Windows Presentation Foundation projekt a má namespace `OpticalDocumentRecognizer.GUI`.

OCRManager je samostatný projekt, který se kompiluje do dynamické knihovny, a používá namespace `OpticaDocumentRecognizer.OCRManager`.

Dalším samostatným projektem je OpticalDocumentRecognizer, obsahující veškerou funkcionalitu, kromě napojení na OCR, o které se stará právě OCRManager. Tento projekt je dále rozčleněn do složek podle konkrétních funkcí a tomu odpovídá i rozdělení namespaců.

Složka Automatization obsahuje třídy související s automatizací, tj. detekcí obrysu, detekcí datových polí, zarovnáním buňek a automatikcými opravami výstupu z OCR. Používá namespace `OpticalDocumentRecognizer.Automatization`.

V oddílu DataExport je řešení pro export dat z aplikace, o možnostech přidání dalšího řešení píšu níže, použitý namespace je `OpticalDocumentRecognizer.DataExport`.

DataTemplate je oddíl pro práci s datovými šablonami, využívá namespace `OpticalDocumentRecognizer.DataTemplate`. O struktuře XML s datovou šablonou se zmiňuji v subsekci 3.7.

Namespace `OpticalDocumentRecognizer.Filters` zahrnuje všechny použité práci, jakož i práci s externí knihovnou *AForge.NET*.

Namespace `OpticalDocumentRecognizer.General` obsahuje společná data a objekty používané z celé aplikace.

Namespace `OpticalDocumentRecognizer.ImageProcessor` zahrnuje především prostředky na rozřezání obrázku na části odpovídající jednotlivým buňkám datové šablony a dále třídu pro proces zpracování obrazu, vytvářený v editoru.

Složka XML Schema obsahuje definice struktury používaných xml souborů.

4.2 Úprava automatických oprav

Automatické opravy jsou definované ve zvláštním XML souboru nazvaném `Corrections.xml`. Zde je krátká ukázka struktury xml souboru.

```
<Corrections xmlns="
    OpticalDocumentRecognizer:Corrections:Corrections.xsd">
        <CorrectionGroup type="date">
                <Correction o="," r="."/>
```

```
        </CorrectionGroup>
</Corrections>
```

Ukázkový XML soubor definuje nahrazení čárky „,“ v datumu tečkou „.“.

CorrectionGroup označuje skupinu oprav pro daný datový typ, definovaný atributem *type*. *Correction* pak definuje konkrétní opravu, atribut *o* označuje text k nahrazení, atribut *r* pak požadované nahrazení.

Editací tohoto xml souboru je možné snadno přidávat, upravovat nebo odebírat automatické opravy. Důležité je ovšem vytvářet jen takové opravy, které budou v každém případě správné. Pokud například v datumu nahradíme „1“ za „2“, zřejmě nepůjde o správné řešení. Program také volá opravu pouze v případě, že selže kontrola správnosti extrahovaného textu.

4.3 Přidání filtrů do editoru

Při dlouhodobém používání, nebo při speciálních typech dokumentů se mohou hodit i jiné filtry, než které jsou přístupné v editoru nyní. Snažil jsem se proto přidání nových filtrů co nejvíce usnadnit. V editoru funguje systém na zobrazené parametrů k danému filtru, rozlišuje 3 typy parametrů a to: číselnou hodnotu, výběr ze seznamu, odpověď typu ano/ne.

Přidání nového filtru sestává ze dvou kroků. Nejprve je nutné ve třídě `FilterCentre` v metodě *ApplyFilter* přidat novou větev do switch příkazu, obdobně jako jsou zde záznamy pro jiné filtry, tj. vymyslet stringový identifikátor filtru a do tělo příkazu switch pak pro tuto větev doplnit volání metody, která implementuje přídáváný filtr. Metoda musí vracet výsledek typu *Bitmap*. Druhým krokem je editace souboru *FilterList.xml*, kde je potřeba přidat záznam o novém filtru.

```
<Filter id="Morphological" name="General Morphological
    Filter">
     <Question name="Erosion" default="true" />
     <Enum name="Structuring Element" type="System.String"
          default="Horizontal line">
          <Value>Horizontal line</Value>
          <Value>Vertical line</Value>
          <Value>Cross</Value>
     </Enum>
     <Range name="Size" min="3" max="11" default="3"
          type="System.Int32" />
 </Filter>
```

Ukázka záznamu o filtru pro erozi/dilataci v souboru *FilterList.xml*.

Atribut *id* v elementu *Filter* je id filtru, zadané již ve třídě FilterCentre.

Atribut *name* je potom jméno filtru zobrazované v editoru.

Question je typ parametru ano/ne, zde se jedná o přepínač mezi erozí a dilatací.

Enum je parametr pro výběr hodnoty ze seznamu, elementy *Value* jsou pak jednotlivé možnosti.

Range je parametr pro výběr čísla z intervalu, omezeného hodnotami atributů *min* a *max*.

Každý parametr může mít výchozí hodnotu, tu udává atribut *default*.

Parametry typu Enum a Range pak mají i datový typ, definovaný atributem *type*.

4.4 Výměna OCR

Aplikace je navržená tak, aby fungovala s externím OCR enginem, který je snadno vyměnitelný. Dostupné jsou 2 metody připojení enginu k aplikaci.

Tou první je připojení přes implementaci interface `IOCRConnection`, které vyžaduje implementaci 2 metod, obě příjmají *Bitmap* jako vstup a jedna vrací text rozdělený po slovech, tedy pole typu *string* a druhá vrací celý text jako jeden řetězec.

Druhou možností je připojení externího OCR ve formě spustitelného souboru, přes konzolového klienta. Tento postup je v aplikaci použitý pro engine Tesseract 3.0 a snadnou úpravou ve třídě `OCRExecutableConnection` lze tento engine vyměnit za jiný. Stačí změnit cestu ke spustitelnému souboru a nastavení parametrů se kterými se spouští.

4.5 Vlastní exportní modul

Aplikace je snadno rozšiřitelná o další exportní moduly, praktické by mohlo být například ukládání do sql databáze nebo do xml souboru obsahujícího všechny data dokladu jednoho typu. Exportní modul musí splňovat rozhraní IExportData, které vyžaduje implementaci následujících vlastností a metod:

- Name - název exportu, zobrazovaný v aplikaci
- Type - určuje typ exportu, používané typy jsou 2, pokud je typ File, aplikace zavolá okno na uložení do souboru a stream k tomuto souboru pak předá exportu, při typu Other předá pouze data a zbytek nechá na exportu
- Init - slouží k inicializaci exportu/ověření že je možné daný export použít, pokud nelze, vrátí false a export se nezobrazí v možnostech aplikace
- GenerateOutput - pokud je to možné vygeneruje data k uložení, pro možnost zobrazení uživateli a kontrolu
- SaveOutput s parametrem Stream - používá se u typu File, uloží data do souboru ve formátu podle typu exportu, pro exporty typu Other nemusí být implementováno
- SaveOutput bez parametru Stream - používá se u typu Other, kam uloží data nechá na exportu, pro exporty typu File nemusí být implementováno

Po vytvoření exportního modulu je ještě potřeba jej přidat do `DataExportManageru`, kde stačí vytvořit novou instanci exportu, zavolat Init a pokud se povede, tak export přidat do seznamu dostupných exportů.

5. Uživatelská příručka

Aplikace *Optical Document Recognizer* je určena pro extrakci dat z naskenovaných obrázků. Je napsaná v jazyce C# v prostředí .NET. Na následujících stánkách je popsáno jak aplikaci používat, od instalace, přes základní orientaci v aplikaci, až po pokročilou práci s procesy pro zpracování.

5.1 Instalace

Proces instalace se nijak neliší od běžné instalace jiných aplikací. Po spuštení instalačního souboru *Setup.exe* je potřeba zvolit umístění aplikace a o nic jiného není třeba se starat. Všechny potřebné externí části jsou součástí instalačního balíčku, včetně knihovny *AForge.NET* a spustitelného souboru OCR enginu *Tesseract*. Po instalaci je možné aplikaci spustit souborem *OpticalDocumentRecognizer.exe* umístěném ve složce vybrané během instalace.

5.2 První spuštění

Po spuštění se zobrazí obrazovka jako na obrázku 5.1.

5.3 Režimy práce

Pro maximální jednoduchost pochopení, a také proto, aby práce s aplikací byla co nejjednodušší, dává aplikace uživateli na výběr ze tří módů, odstupňovaných podle úrovně uživatele.

Základní mód zpřístupní užívateli režim pro tvorbu šablony a režim pro extrakci dat. Protože nemá přístup do editoru, a tudíž ani nemůže tvořit procesy pro zpracování, nemá ani možnost je používat v režimu extrakce dat.

V módu pro pokročilé uživatele se zpřístupní editor a veškerá funkcionalita spojená s procesy pro zpracování obrazu. Jedná se tedy o plný mód, kdy z funkčnosti aplikace nechybí vůbec nic.

Poslední mód je pak určený vývojářům, obsahuje stejnou funkcionalitu jako mód pokročilého uživatele a navíc zobrazuje několik kontrolních a testovacích údajů v jednotlivých sekcích.

5.4 Tvorba šablony

Do tohoto režimu se dostanete kliknutím na tlačítko s nápisem „Vytvořit datovou šablonu“ na uvítací obrazovce. Poté po vás aplikace bude požadovat výběr obrázku s dokladem, na který chcete šablonu vytvořit. Následně se otevře obrazovka jako na obrázku 5.3. Po vybrání obrázku se zobrazí obrazovka tvorby šablony. Nyní lze kliknutím na tlačítko *Generate Border* najít obrys dokumentu. V případě, že aplikace neuspěje s hledáním obrysu, bude vám to oznámeno a mělo by přijít na řadu manuální vybrání obrysu. To lze provést po kliknutí na tlačítko *Select*

Obrázek 5.1: OpticalDocumentRecognizer - uvítací obrazovka

Border. Poté přichází na řadu detekce datových oblastí. K automatické detekci slouží tlačítko *Generate Cells*. Po jeho stisknutí proběhne první detekce a na obrazovce se zobrazí navrhované datové buňky. K úpravě tvaru nalezených buněk slouží 2 posuvníky - *Vertical space*, ten určuje vertikální mezeru mezi buňkami, a *Horizontal space*, který určuje mezeru horizontální. Jejich posunem lze upravit podobu buněk tak, aby odpovídala realitě.

Pokud například po prvotním vygenerování jsou jako buňky označena jednotlivá písmena, je potřeba zvětšit hodnotu *horizontal space*. Naopak, pokud splývají datové oblasti umístěné na dokladu vedle sebe, je potřeba tuto hodnotu zmenšit. Podobně pokud například jako 2 buňky jsou označeny písmeno a jeho čárka nebo háček, je potřeba zvětšit hodnotu *vertical space*. A opět, pokud jsou do jedné buňky spojeny například 2 řádky, lze je rozdělit snížením hodnoty. V případě, že se v dokladu vyskytuje text různých velikostí, je doporučeno toto nastavení provést s ohledem na velikost textu, který nás zajímá. Pokud nás zajímá text různých velikostí, doporučuji nastavit takové parametry, aby byl dobře zpracován text velikosti, která se v datových oblastech vyskytuje nejvíce, zbytek se pak upraví manuálně.

Pro manuální úpravu slouží horní panel nástrojů, který přepíná mezi jednotlivými režimy práce s buňkami. Následuje stručný popis jednotlivých tlačítek:

- Normal - výchozí mód, žádné speciální funkce
- New - slouží pro vytvoření nové buňky, stisknutím a držením levého tlačítka myši a následným pohybem vznikne obdélník vymezující oblast buňky, uvolněním tlačítka je buňka vytvořena
- Join - slouží pro spojení buněk, po označení buňky a kliknutí na jinou buňku dojde k jejich spojení, buňce zůstane jméno první označené buňky

- Move - slouží k přemisťování existujících buňek, po stisku a držení levého tlačítka myši nad buňkou je možné s ní libovolně manipulovat
- Delete - mód pro mazání buněk, po kliknutí na buňku dojde k jejímu smazání

Jednotlivé módy zůstávájí aktivní, dokud nedojde k přepnutí do jiného módu, lze tedy snadno rychle vytvořit/spojit/smazat i větší množství buněk.

Po označení buňky se v pravé části dole zobrazí seznam vlastností buňky, které lze měnit. Každá buňka má jméno, které ovlivňuje zejména podobu výstupu. Dále se zde dají nastavit šířka a výška a poloha buňky v šabloně.

Každá buňka má také svůj datový typ, který určuje, jaká data jsou v dané oblasti. Program podle tohoto nastavení zkontroluje formální správnost extrahovaných dat a případně sám provede některé opravy. Dostupné datové typy jsou:

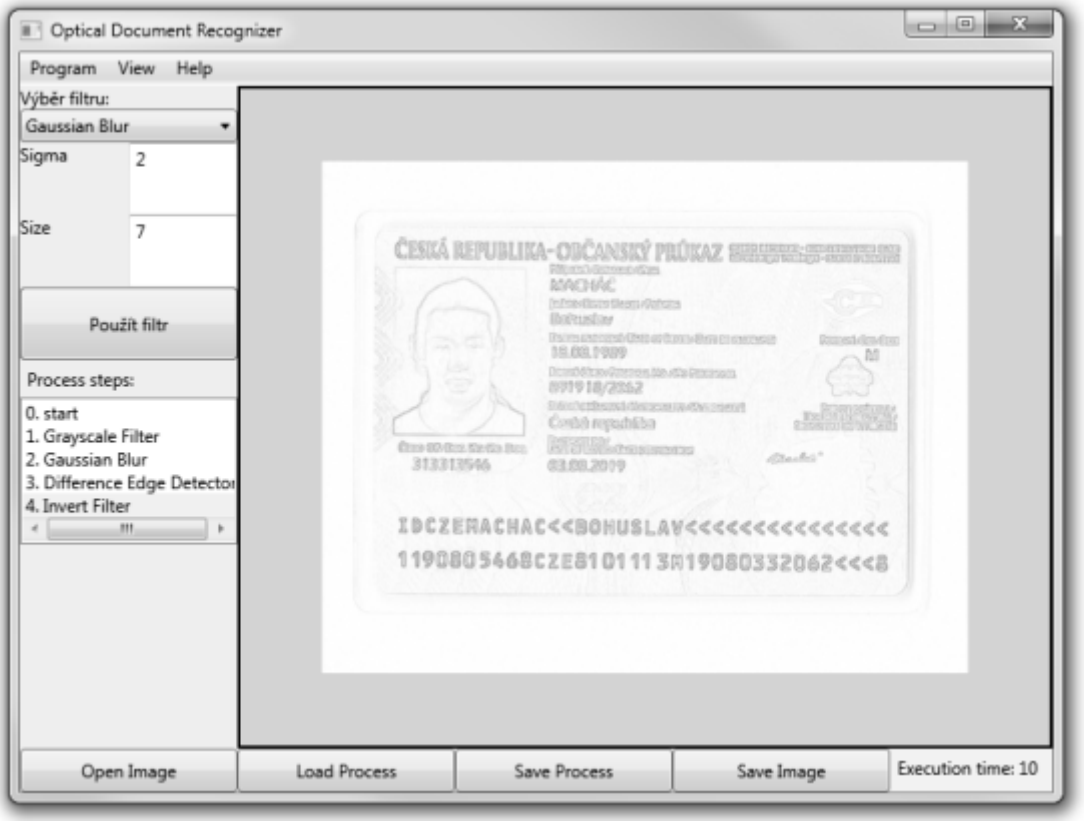
- IntegerNumber - celé číslo
- DecimalNumber - číslo s desetinnou čárkou
- Text - obecný text
- Date - datum
- Unspecified - nespecifikováno, neproběhne žádná kontrola
- Multiline - víceřádkový text
- Image - provede výřez obrázku, jako hodnotu vrátí cestu k souboru s výřezem na disku

Po vytvoření šablony je ještě potřeba ji pojmenovat, políčko pro její jméno je v pravé části. Poté je již možno šablonu uložit kliknutím na tlačítko *Save Template*. Tlačítko *Open template* pak slouží k zobrazení existující šablony, kterou je takto možno upravit a uložit.

5.5 Extrakce dat

Pro vstup do tohoto režimu je potřeba na uvítací obrazovce klknout na tlačítko *Zpracovat dokument*. Následuje výběr obrázku s dokladem, ze kterého se budou data extrahovat. Po jeho načtení se zobrazí obrazovka jako na obrázku 5.4. Nyní je potřeba v seznamu *Select data template* vybrat správnou datovou šablonu pro doklad na obrázku. Po jejím vybrání dojde k automatickému sesazení šablony na doklad. V případě, že se proces sesazení nepovede, je možné šablonu přetažením sesadit manuálně. Kliknutím na tlačítko *Adapt Cells* je možné zarovnat jednotlivé buňky v případě, že přesně nesedí.

Po stisknutí tlačítka *Extract data* dojde k extrakci dat, jejich kontrole podle předdefinovaných datových typů a případným automatickým opravám. Poté jsou zobrazena v přehledné tabulce jako na obrázku 5.5, kde je možno opravit případné chyby výstupu z OCR enginu. Po konrole dat je ve spodní části seznam s výběrem formátu exportu a kliknutím na tlačítko *Export* dojde k uložení dat.



Obrázek 5.2: OpticalDocumentRecognizer - editor

Pro pokročilé uživatele je navíc dostupná možnost vybrat proces pro zpracování obrazu přes seznam *Select process template* a tlačítkem *Run processing* jej aplikovat na obrázek s dokladem. Je zpřístupněna i možnost testovací extrakce dat, tlačítkem *Test Extract*, která zobrazí nové okno s výstupem z OCR enginu a jeho zavřením je možno se vrátit zpět do režimu extrakce dat a třeba ještě změnit proces pro zpracování. V okně s exportem je pak pokročilým uživatelům zpřístupněna možnost zobrazit podobu exportovaných dat, tlačítkem *Show*, která, pokud se dá takto zobrazit, se ukáže v novém okně v textovém poli.

5.6 Editor a proces zpracování

Tento režim je přístupný jen pokročilým uživatelům a předpokládá, že pro jeho používání mají dostatečné znalosti. Cesta do něj vede přes tlačítko *Editor* na uvítací obrazovce. Zobrazí se obrazovka jako na obrázku 5.2. Pro používání filtrů a tvorbu procesů pro zpracování je potřeba referenční obrázek. Ten je možné načíst po kliknutí na tlačítko *Open Image* v dolní části obrazovky. V levé horní části je možné vybírat mezi dostupnými filtry. Po vybrání konkrétního filtru se zobrazí parametry dostupné pro daný filtr. Filtr se aplikuje tlačítkem *Použít filtr*.

Použití filtru se eviduje v tabulce *Process steps*, kde je vidět posloupnost použitých filtrů. Kliknutím na libovolný krok v tabulce se obrázek vrátí do podoby po aplikaci vybraného kroku. Změnou parametrů nebo výběrem jiného filtru je možné změnit proces a pokračovat novým filtrem. Celý proces se pak dá uložit tlačítkem *Save Process*, kde je potřeba vybrat jméno, pod kterým bude dostupný

pro editaci a hlavně v režimu extrakce dat. Upravit existující proces lze po jeho nahrání tlačítkem *Open Template*.

Editor také umožňuje kdykoliv uložit právě zobrazený obrázek, k tomu slouží tlačítko *Save Image*. V pravé spodní části se také zobrazuje údaj o trvání aplikace filtru, aby měl uživatel představu jak je daný filtr náročný. Údaj je zobrazen u popisku *Execution time* a je v milisekundách.

Závěr

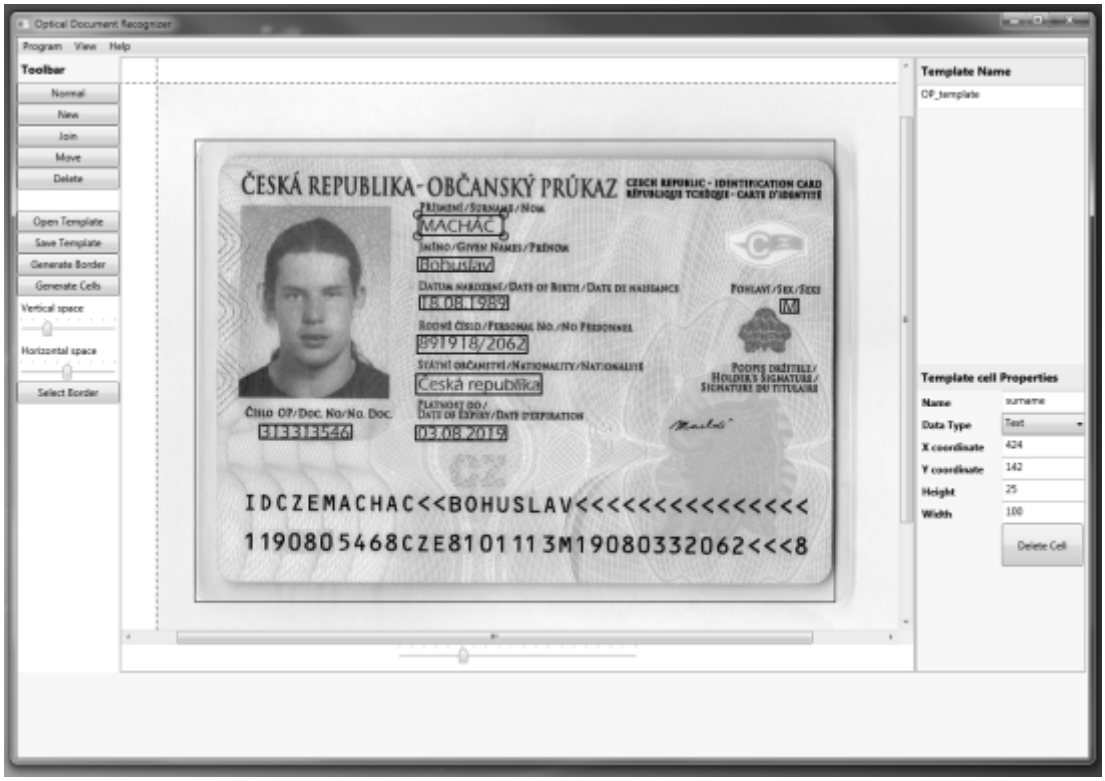
Navrhnul jsem a následně realizoval aplikaci umožňující extrahovat data z naskenovaných tištěných dokladů. Aplikace používá k extrakci datových šablon, které popisují daný typ dokladu, zejména tedy oblasti, kde se nachází data, která je potřeba extrahovat. K optickému rozpoznávání znaků používá aplikace externí OCR engine Tesseract. V případě potřeby je možné tento engine snadno vyměnit za jiný. Po získání dat z dokumentu nabízí aplikace export dat do formátu XML, HTML nebo obyčejného textu. Tyto exportní moduly jsou napsány tak, aby bylo jednoduché přidat další možnosti exportu, vyplývající z daného využití aplikace.

Funkčnost aplikace byla vyzkoušena na několika typech dokumentů, jakožto i více kusech dokladů stejného typu. Vzhledem k tomu, že na většině takových dokladů jsou osobní údaje, není možné zde uvádět konkrétní ukázky. Pro demonstraci jsem tedy použil svůj upravený občanský průkaz, kde jsou pozměněny všechny důležité textové údaje. Také jsem vytvořil fiktivní doklad, pro další možnosti prezentace. Zde je ukázka extrakce dat z mého občanského průkazu, ve fázi detekce obrysu a buněk při tvorbě šablony na obrázku 5.3, přes výběr hotové šablony v extrakci dat na obrázku 5.4, až po zobrazení extrahovaných dat na obrázku 5.5.

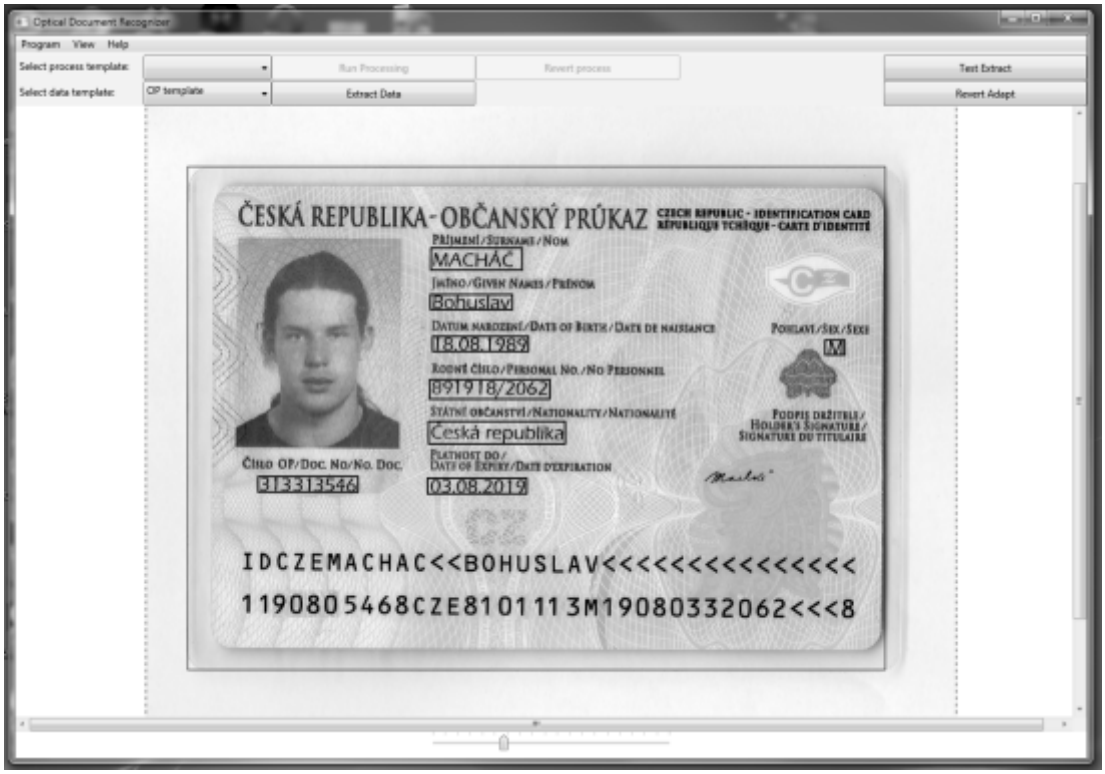
Vyzkoušené doklady:

- občanský průkaz
- kartička pojištěnce
- řidičský průkaz
- osvědčení o registraci motorového vozidla
- opencard
- kartička Student Agency
- ISIC
- Plzeňská karta

Pokud byly vstupní obrázky dostatečně kvalitní a vyhovovaly požadavkům aplikace, byl proces detekce obrysu i jednotlivých oblastí úspěšný. Problémy se objevily s doklady s průhlednými a bílými okraji, kde podle osvětlení během skenování, buď okraj byl viditelný a rozpoznatelný, nebo nebyl a obrys byl zdeformován. U průhledných dokladů většinou posunut k neprůhlednému okraji, u bílých občas vůbec nenalezen. Občas se objevil i vzorek s jinými problémy, ale pokud se nejednalo o nedostatečně kvalitní obrázek, vylučující úspěch OCR systému, dalo se za použití manuálních úprav dostat ke správnému výsledku. Osvědčil se i použitý OCR systém, některé jeho drobné nedostatky řeší automatické opravy, jiné zůstávají na uživateli.



Obrázek 5.3: občanský průkaz - tvorba šablony



Obrázek 5.4: občanský průkaz - extrakce dat

Obrázek 5.5: občanský průkaz - extrahovaná data

Plány do budoucna

Projekt rozhodně nabízí mnoho příležitostí k rozšíření a zdokonalení. Implementované metody jsou sice pro extrakci dat většinou dostačující, ale existují krajní případy, kdy použité postupy selžou. Průběh extrakce i v případě, že nenarazí na žádný problém, potřebuje vstup a interakci od uživatele, s případnými problémy potom rostou i nároky na uživatelský vstup. Cílem by tedy mohlo být aplikaci dále automatizovat a vylepšovat použité metody tak, aby si samy dokázaly poradit s větším množstvím nestandardních situací. Ideálem v tomto směru by byl stav, kdy by uživatel pouze vybral dokument, případně složku s více dokumenty, a formu exportu a zbytek by zvládla aplikace samostatně.

Z hlediska funkcionality aplikace bych se při dalším vývoji zaměřil na implementaci možnosti zpracování velkých dávek stejných dokladů na základě postupu při zpracování prvního z nich způsobem, který se používá v aplikaci nyní. Také by stálo za to dopracovat některé části, jejichž funkcionalita je pouze naznačena, nebo omezená. Například log v současné fázi eviduje pouze výjimky a není přístupný mimo běh aplikace. V grafické části aplikace je také prostor pro vylepšení. Nabízí se rozšíření možností exportu, nebo také vymezení OCR enginu z aplikace tak, aby se dal vyměnit bez zásahu do kódu. Dal by se zdokonalit systém datových typů, například rozšířit o regulární výrazy.

Z pohledu automatizace by se mohl vývoj ubírat směrem vylepšování stávajících funkcí, ale i přidáním funkcí úplně nových. V detekci obrysu by mohl být navrhnut důmyslnější způsob výběru přímek a propracovanější testování jejich smysluplnosti. Systém detekce je sice obecný a měl by pojmout většinu případů, ale určitě i zde je prostor pro vylepšení. V detekci význačných oblastí by šla přidat logická analýza layoutu, například pro odhad jména buněk podle popisků, nebo testování obsahu buňky a odhad datového typu. Dále by se dalo přijít s nějakou metodou, jak lépe odhadovat parametry textu. Dále by šla v automatické fázi přidat podpora rozdílné velikosti textu tak, aby to mělo vliv i na velikost mezer. Případně implementovat některou pokročilejší metodu pro analýzu layoutu s podporou více směrů textu, otázkou je, jestli by se taková funkce uplatnila.

Seznam použité literatury

[1] Gonzalez, Rafael C., Woods, Richard E.. *Digital Image Processing*. 3rd edition Upper Saddle River: Prentice-Hall, 2008. ISBN 978-0-13-168728-8.

[2] Otsu, Nobuyuki. *„A Threshold Selection Method from Gray-Level Histograms“*. Systems, Man and Cybernetics, IEEE Transactions on , vol.9, no.1, pp.62-66, Jan. 1979. doi: 10.1109/TSMC.1979.4310076 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4310076-&isnumber=4310064

[3] Duda, Richard O. ,Hart, Peter E.. *Use of the Hough transformation to detect lines and curves in pictures*. Commun. ACM 15, 1 (January 1972), 11-15, Jan. 1972. DOI=10.1145/361237.361242. URL: http://doi.acm.org/10.1145/361237.361242

[4] Cattoni, Roldano, Coianiz, Tarcisio, Messelodi, Stefano, Modena, Carla Maria. *Geometric Layout Analysis Techniques for Document Image Understanding: a Review*. ITC-irst Technical Report TR#9703-09. January 1998.

Obsah přiloženého CD

`doc/`	dokumentace
`prace.pdf`	text bakalářské práce
`setup.exe`	instalační balíček aplikace
`src/`	zdrojový kód aplikace
`test/`	testovací obrázky dokladů