VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

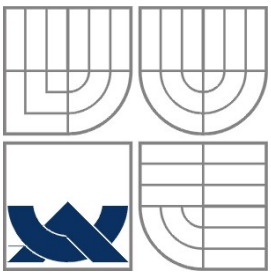
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

OPTICKÉ ROZPOZNÁNÍ TEXTU V OBRÁZCÍCH

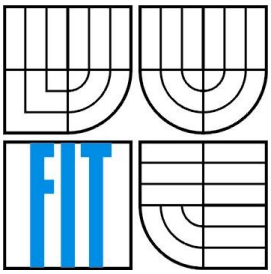
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PAVEL KADLIC
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

OPTICKÉ ROZPOZNÁNÍ TEXTU V OBRÁZCÍCH

OPTICAL CHARACTER RECOGNITION IN IMAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVEL KADLIC

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR CHMELAŘ

BRNO 2011

Abstrakt

Bakalářská práce se zaměřuje na detekci, lokalizaci, sledování a extrakci textu v obrázcích a videu. Jsou v ní popsány algoritmy vedoucí k dosažení optického rozpoznání textu. Práce se zabývá také implementací algoritmů, výběrem vhodného vzorku testovacích dat a jeho vyhodnocením.

Abstract

The bachelor's thesis focuses on detection, localization, tracking and extraction of text from images and videos. There are described algorithms achieving optical chacarcter recognition. The thesis deals with implementation of algorithms, selecting a suitable sample of test data and its evaluation.

Klíčová slova

Optické rozpoznání textu v obraze, OCR, detekce, lokalizace, sledování, spojité oblasti, segmentace, shlukování, nalezení ostrých hran, OpenCV.

Keywords

Optical character recognition in images, OCR, detection, localization, tracking, connected components, segmentation, clustering, edge detection, OpenCV.

Citace

Pavel Kadlic: Optické rozpoznání textu v obrázcích, bakalářská práce, Brno, FIT VUT v Brně, 2011

Optické rozpoznání textu v obrázcích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Chmelaře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Pavel Kadlic
27. července 2011

Poděkování

Děkuji vedoucímu práce Ing. Petru Chmelaři za poskytnuté konzultace, odborné články a cenné rady.

Obsah

1 Úvod

S problémem optického rozpoznání textu, některým bude bližší termín OCR, se setkáváme denně. OCR (z anglického *optical character recognition*) je označení procesu, při kterém z původního obrázku s výskytem textu vznikne výstup v podobě prostého textu (ideálně do textového dokumentu). My se avšak touto metodou budeme zabývat pouze okrajově.

Hlavní náplní práce je především správně lokalizovat text z obrázků a videí. Je třeba určit na základě dále zmíněných kritérií, zda-li se v obrázku nachází na určité pozici text, případně výskyt netextových částí klasifikovat jako oblast, kde se text nenachází. K tomuto účelu nám pomohou všeobecné vlastnosti textu vyskytujícího se v digitální podobě, o kterých se zmiňuje kapitola 2. Budeme s nimi počítat v algoritmech pro lokalizaci. Druhá kapitola nám také představí ucelenější pohled na celý proces hledání textu v obraze. Proces se rozčlení do několika fází. U každé uvádím kritická místa, na které je třeba si dát pozor.

Kapitola 3 popisuje operace nad obrázkem vedoucím k závěru, které oblasti budou pro další hledání textu podstatné a které budeme moct vynechat. Představím dva základní principy hledání zajímavých bodů a to metodu založenou na hledání ostrých hran v obraze a metodu hledající spojité oblasti. Kromě těchto principů uvedu i některé další, které mohou být užitečné v některých dalších případech.

V kapitole 4 je popsán vlastní návrh řešení. Způsoby vedoucí k nalezení a extrakci textu z obrázku v ideálních případech, možné problémy a nastínění jejich řešení.

Kapitola 5 hodnotí předem nasbírané zkušenosti a návrhy v podobě implementace několika algoritmů. Zde se také dočtete o časové náročnosti jednotlivých úseků kódu, co bylo přínosem a co naopak ztrátou času. Jelikož pracujeme s obrázky, můžeme se v každém kroku implementace přesvědčit o účelnosti použitých algoritmů.

Finální zhodnocení, testování a případné využití softwaru naleznete v kapitolách 6 a 7.

2 Vlastnosti textu a jeho nalezení

V práci se zabýváme operacemi s obrázky i videi. Abych nemusel pokaždé uvádět fakt, že se daná operace vztahuje k obrázkům i videím, budu uvádět pouze operace nad obrázkem (vždyť samotné video je sekvence rychle po sobě jdoucích obrázků), pokud tomu vysloveně nebude jinak.

Prvním krokem vedoucím k úspěchu je zjištění samotných vlastností textu, které se v digitálním obrázku vyskytují. Jedná se především o ty vlastnosti, které nám pomohou oddělit text od ostatních objektů v obraze. Textem máme na mysli buď celé věty, slovní spojení nebo jen slova, či samostatná písmena. V práci se zaměřujeme na nalezení větších souvislých celků textu než na pouhá písmena nebo oblasti s textem, která jsou příliš malá nebo příliš velká.

Jaký dopad mají vlastnosti textu na jejich nalezení v obraze pojednává podkapitola 2.1. O to jak s informacemi o textu nakládat zase popisuje podkapitola 2.2. Hlavním přínosem této podkapitoly je předvést možnosti zpracování obrazu v takové míře, kdy je nalezený text v obraze oddělen od ostatních částí dostatečně na to, aby byl rozpoznán.

2.1 Podoba textu v obrázcích

Mezi základní vlastnosti textu, se kterými budeme pracovat, patří vlastnosti související se sazbou textu (barva, tvar, velikost, font, obtékání, pozice) a ostrostí hran. Dále zmíním význam komprese, coby vlastnosti obrazu, a v případě videa i stálosti textu.

Prvním charakteristickým znakem textu v obrázcích je jeho barva. Člověku nedělá problém rozlišit a přečíst text, který se skládá z více barev. Počítač to má ale složitější. V případě, že má text, který v obrázku hledáme, všechny písmena ve stejné barvě, poté je jeho identifikace úspěšnější než v případě textu, který je různobarevný. Nyní trošku předběhneme za cílem znázornění tohoto faktu.

Ve fázi předzpracování obrazu ve většině případů používáme konverzi barevného obrázku do stupňů šedi (více v kapitole 3). V případě barevného obrázku a jednobarevného textu bude mít text po převodu na obrázek do škál šedi opět stejnou intenzitu. Obrázky 2.1 a 2.2 poukazují na převod barevného obrázku se stejnobarevným a různobarevným textem do stupňů šedi.

Obrázek 2.1a: Vstupní barevný obrázek.

Obrázek 2.1b: Obrázek po převodu do stupně šedi.

Obrázek 2.2a: Vstupní barevný obrázek.

Obrázek 2.2b: Obrázek po převodu do stupně šedi.

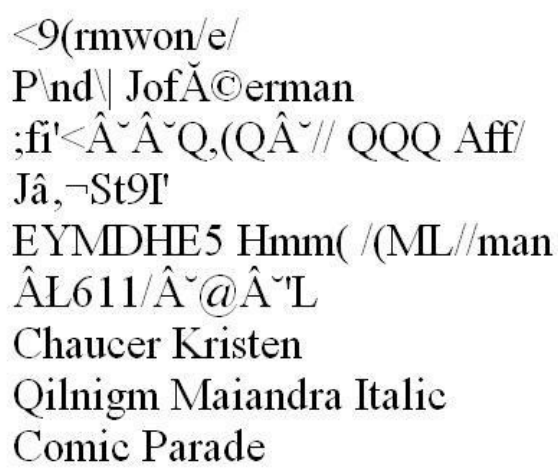
Ukázka převodu stejnobarevného textu do stupně šedi nezpůsobí takové ztráty na poznávacích znacích textu v obraze jako převod různobarevného textu. Na obrázku 2.2a je rozdíl intenzity mezi barevnými písmeny a pozadím vysoká, kdežto na obrázku 2.2b je v některých případech minimální, což může v některých případech vést k ne úplně správné lokalizaci textu oproti pozadí. Hlavní roli hrají ohraničující barvy kolem jednotlivých písmen. V případě jednobarevného textu předpokládáme i jednotnou barvu, která text ohraničuje. To je přínosné pro další rozpoznávací charakteristiky textu. Naopak u vícebarevného textu nedokážeme určit barvu obtékání textu. Nevíme, jestli se jedná o jednotnou barvu nebo více barev. Při převodu s binárním prahováním na černobílý obrázek může dojít ke sloučení některých písmen s pozadím a to právě v závislosti minimálního rozdílu intenzity mezi písmenem a pozadím.

Barevnost textu má vliv ještě na algoritmy, které jsou probrány v kapitole 3. Při detekci hran v obraze, spojitých oblastech i při shlukování se k ní vrátíme.

Dalším charakteristickým znakem textu v obraze je jeho tvar. Pravidelně se setkáváme se základními fonty používanými například na obálky knih, noviny, reklamní předměty nebo titulky ve videích. Tyto základní fonty a tvary písmen mají společné vlastnosti a to, že jsou snadno čitelné. Navíc většina programů již má databázi těchto fontů a umí s nimi náležitě zacházet.

V programu, který implementuji, využívám software Tesseract OCR, který má vzory základních fontů a ze kterých následně rozeznává jednotlivá písmena. V případě, že bychom měli obrázek s tvarem písma křivým, zaobleným nebo neznámým fontem, je lokalizace a rozpoznání textu složitější. Tesseract OCR má sadu základním fontů, které dokáže rozpoznat. Cílem práce však už není všechen rozsah použitých fontů na testovacích datech správně rozpoznat pomocí nástroje Tesseract OCR.

Obrázek 2.3a: Několik druhů fontů.



Obrázek 2.3b: Výstup nástroje Tesseract.

Na obrázku 2.3b se nachází rozpoznání v podobě textu nástroje Tesseract ze vstupních dat z obrázku 2.3a. Většina OCR programů funguje na principu, kdy si větu rozdělí na slova a slova na jednotlivé znaky. V případě, že jednotlivá písmena ve slovech na sebe navazují bez viditelných mezer, nedokáže nástroj Tesseract oddělit jednotlivá písmena a rozpoznat je (obrázek 2.3).

I proto ve velkém počtu vstupních testovacích obrázků pro detekci a rozpoznání textu používám základní typy fontů, které jsou přívětivé pro Tesseract. Více o Tesseractu v části o extrahování textu 2.2.4.

Dále si musíme jednoznačně určit kritéria, která nám limitují velikost textu a zároveň budou zahrnovat co nejširší škálu použití. Například samotný Tesseract počítá s minimem výšky malého písmene *x* při 10pt a 300dpi okolo 10 pixelů [13]. Výška písmene menší jak 8 pixelů je označena jako *noise*[1]. Přitom standardní výška tohoto písmene při daném rozlišení je 20 pixelů. Záleží avšak také na použitém fontu, kdy se jeden od druhého může lišit velikostí jednotlivých písmen.

Druhým bodem je určit maximální velikost písmen. To je taky velmi obtížné vzhledem k tvorbě automatizovaného nástroje pro hledání textu v obraze. Představme si širokoúhlým objektivem vyfocený hřbet knihy, přes jehož celou výšku je vytisknutý její název. V tomto případě může výška textu zabírat 90% výšky celého obrazu. Anebo již zmíněný novinový článek, kde bude výška malého písmene veliká poměrově jedna ku stu oproti výšce obrázku.

Klíčovou roli při hledání textu hraje i obtékání kolem písmene. Tím je myšleno pozadí s odlišnou intenzitou oproti textu, které může být stejné barvy anebo odlišné. U video titulků je to jejich obrys. Velmi dobře se rozpoznají spojité vnitřní oblasti ohraničené stejnou intenzitou. Texty se stínováním mají taktéž identickou barvu okolí.

1 Anglické označení pro šum (věc, co nepatří do kontextu).

Obrázek 2.4a: Ukázka textu na černozeleném pozadí.

Obrázek 2.4b: Ukázka textu na bíločerném pozadí.

Obrázek 2.4c: Ukázka textu na bílomodrém pozadí.

Obrázky 2.4 poukazují na různé kombinace neohraničeného písma s barevným pozadím. Jak vidíme, nejčitelnější je první text, kdy je barva písma ve velkém kontrastu oproti pozadí. Naopak u dalších obrázků narážíme na části, kdy některá písmena splývají s okolím a není zcela jednoznačné jejich rozpoznání. Výstupy programu Tesseract jsou následující:

- **Obrázek 2.4a**: *But somebody sent them,*
- **Obrázek 2.4b**: *Exe m ' kurounii*
- **Obrázek 2.4c**: *Exa rTf'D|e ovfText bacl<qrounH*

V případě tvorby nebo vkládání textu do obrázků je třeba dbát ohled na tento fakt a používat písmo kontrastní oproti pozadí nebo s viditelným obtékáním jinými barvami.

Jeden z důležitých faktorů lokalizace textu je jeho natočení a pozice. Souvislý text v obraze může být ve vertikální, svislé nebo jinak natočené poloze. Metody založené na hledání pozice vyžadují striktně pevně zvolenou pozici textu. I v případě tvorby práce jsem pracoval se vstupními daty, které mají v drtivé většině případů text vodorovný.

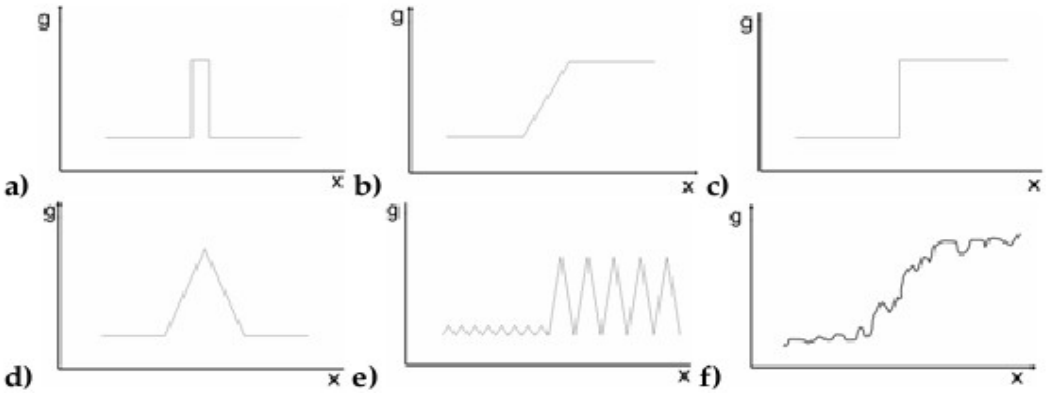
Algoritmy lokalizující text v obraze pracují faktem, že se text vyskytuje ve vodorovné pozici. Při hledání textu v obraze podle souřadnic *y* můžeme jednotlivé řádky textu od sebe odlišit. Naopak průchodem obrazu po ose *x* rozdělíme věty na slova a slova na písmena.

Dále je možné na základě charakteristických znaků pro text v obrázku dle osy *y* uspořádat objekty v obraze, které primárně určíme jako potencionální text. Tím je myšlena kontrola výšky písmene a oblastí, které si jsou vzájemně geometricky podobné (podle souřadnice *y)*. Tuto metodu používám jak při hledání spojitých oblastí, tak při postupu hledání ostrých hran.

Po průchodu obrázku podle osy x už máme kompletní informace o geometrických vlastnostech objektů v obraze. Podle tabulky 6.1 v rámci geometrické analýzy posoudíme, zda jsou objekty potencionálním textem.

Další vlastností textu je přítomnost ostrých hran. Záleží na použitém fontu, kterým je text na obrázku znázorněn. Většina základních fontů má ostré hrany, a tak metody založené na hledání ostrých hran můžeme použít na většinu fontů. Ostrá hrana je oblast v digitálním obraze, ve které

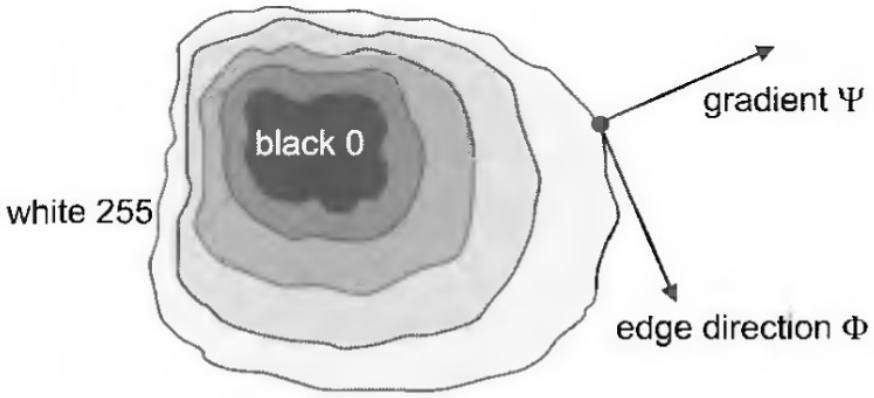
dochází k výrazné změně intenzity. Budeme-li na obraz nahlížet jako na funkci *g*, která pro každý pixel vrací hodnotu intenzity v daném bodě, můžeme za hranu považovat ta místa obrazu, kde má funkce *g* průběh jako na obrázcích 2.5.



Obrázek 2.5: Ukázka typů hran: a) line edge; b) ramp edge; c) step edge; d) roof edge; e) variance edge; f) noisy edge; Zdroj [3].

Z obrázků 2.5 je patrné, že hrana může nabývat několika různých forem. Určení polohy hrany je tedy značně obtížné.

Každá hrana je vlastnost související s individuálním pixelem a je vypočítána z funkce chování okolí daného pixelu. Je to proměnná typu vektor se dvěma složkami - rozsahem a směrem. Směr vektoru se vypočítá vzhledem ke gradientu pod úhlem 90°. Obrázek 2.6 poukazuje na vyhodnocení těchto dvou směrů.



Obrázek 2.6: Směr gradientu a hrany (edge). Zdroj [1].

Určit polohu hrany je obtížné zvláště ze dvou důvodů. Tím prvním je obsáhlá oblast, ve které dochází ke změně intenzity. Otázkou je, co vyhodnotíme jako hranu - jestli celou oblast nebo například jen její střed. Jako druhý důvod uvádíme šum. V případě, že je hrana v oblasti se šumem, nelze jasně určit, která změna intenzity je považována za hranu a která za šum.

K tomu, abychom omezili výskyt šumu v obraze, se většinou používá nízkofrekvenční Gaussův filtr nebo filtr nelineární (medián). Zde jsem převážně čerpal z [1].

Principům nalezení hrany v obraze se budeme dále věnovat v následující kapitole 3.

Stálost je zásadní vlastností textu vyskytujícího se ve videu. Video je sekvence rychle po sobě jdoucích digitálních snímků. Každý snímek je zobrazen pouze na jednu pětadvacetinu sekundy. Jestliže se ve videu objeví text, je jasné, že nebude jenom na jednom z těchto snímků, jelikož by si ho divák nestihl přečíst. Narazíme tedy na několik po sobě jdoucích snímků, ve kterých se text jeví staticky oproti pohybujícímu se pozadí.

Na jednotlivý snímek videa jsou nasazeny klasické algoritmy pro hledání textu v obrázku. V případě, že algoritmy ukážou na nějakou potencionální oblast s textem, dostaneme se do další fáze, kdy je na významné body v obraze představující text aplikován ještě jeden algoritmus, který vyhodnotí stálost těchto bodů v několika následujících snímcích (více v kapitole 5). Zkoumané body se buď nepohybují, nebo se narazí na problém. V případě, že se nepohybují, se zároveň v následujících snímcích algoritmy pro vyhledávání textu v obrázku opět označí jako potencionální možné body textu, a tím jsou označeny jako oblasti s textem. V případě, že se narazí na problém, mohou nastat tyto situace:

- významné body se nenacházejí v následujících snímcích na stejném místě
- významné body se nenacházejí v následujících snímcích vůbec
- významné body se nacházejí v následujících snímcích na jiném místě a zároveň na původním místě jsou objeveny nové významné body z následujících snímků

Z výše uvedeného textu vyplývá, že způsob hledání textu ve videu bude mít o něco lepší úspěšnost jako způsoby vyhledávající text v samotných obrázcích.

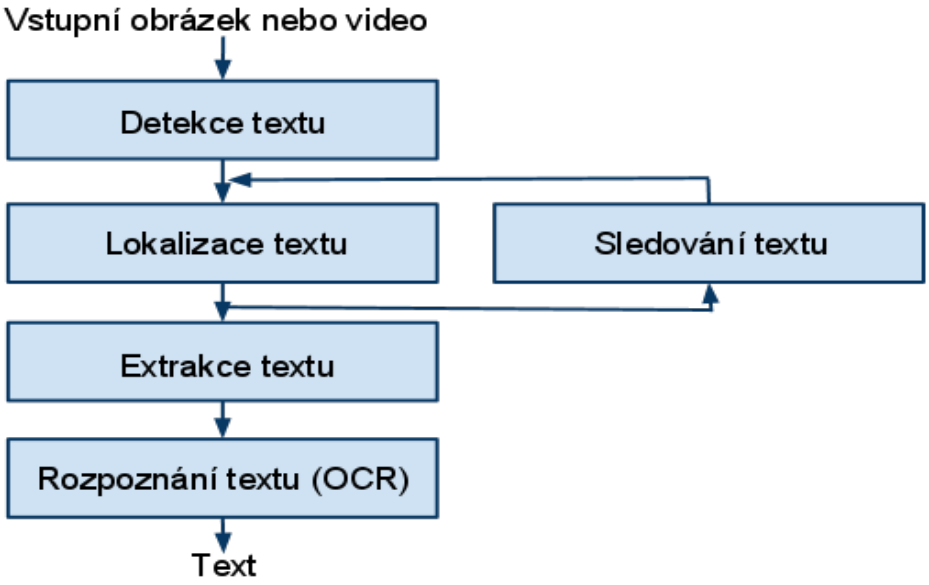
V podkapitole 2.1 jsme si uvedli podstatné vlastnosti textu, na základě kterých hledáme text v digitalizovaných snímcích. Jejich shrnutí je v tabulce 2.2.

Vlastnost textu	Varianta
Barva	Barevná
	Ve stupních šedi
	Černobílá
Tvar	Klasický (základní font)
	Atypický (zaoblené písmo, dekorativní font)
	Spojité písmo
Velikost	Příliš velké / v mezích / příliš malé
Zarovnání	Horizontální / vertikální
	Křivky
	3D perspektiva
Vzdálenost	Mezi slovy
	Mezi písmeny
Pohyb	Statický
	Lineární
	Kolem 2 os
	Kolem 3 os
	Nekontrolovatelný volný pohyb
Obtékání	S obrysem / bez obrysu
Ostré hrany	Při vysoké změně intenzity v obraze
Komprese	Nekomprimovaný snímek
	Komprimovaný snímek (JPEG, MPEG)
Stálost u videa	Přes několik snímků

Tabulka 2.2: Přehled základních vlastností textu. Zdroj [4].

2.2 Extrakce textové informace

Extrakce textové informace (TIE) je všeobecné označení pro získávání informací o textu z obrázků (z anglického *text information extraction*). Abychom mohli ze snímku (respektive série snímků) získat informace o nalezeném textu, je třeba rozdělit postup na několik jednotlivých částí: detekce, lokalizace, sledování a extrakce. Obrázek 2.9 poukazuje na funkčnost a rozdělení jednotlivých podproblémů systému TIE.



Obrázek 2.9: Přehled systému TIE. Zdroj [4].

Kapitola pojednává o jednotlivých částech systému TIE (obrázek 2.9). Informace zde popsané jsou čerpány především z [4].

2.2.1 Detekce textu

Detekce textu je první částí systému TIE, kdy na vstup přichází originální digitální snímek. Ve fázi detekování nemáme žádné údaje o textu, který ve snímku hledáme. Nevíme, jestli se v něm vůbec nachází, jakou má barvu, ani kde leží. Úkolem této fáze je určit, zda se v obrázku text nachází nebo nikoliv. U samostatných obrázků s větší pravděpodobností předpokládáme, že se v nich text bude nacházet, když uživatel program používá. Kdežto u videa je výskyt snímků s textem mnohonásobně menší, než jaký je počet snímků bez textu.

První primitivní a velmi rychlý způsob jak provést fázi detekce textu je každý vstupní obrázek považovat za obrázek obsahující text a při vstupu videa použít velmi nízký práh[2] pro redukci barevného prostoru. Předpokládá se, že oblast s textem tvoří jen velmi malou část celého snímku. Následně se porovnají série několika mála snímků (například snímků v intervalu dvou vteřin). V případě, že se některé oblasti při porovnání výše zmíněných vybraných snímků budou jevit jako statické a některé jako dynamické, předají se snímky další fázi systému TIE s podezřením, že se

2 Více informací o prahu naleznete v kapitole 3.

v nich text nachází. Za statickou oblast považujeme takovou oblast, ve které nedochází ke změnám intenzity pixelů v čase (v tomto případě po dvou vteřinách). Dynamická oblast je taková, kde ke změnám intenzity dochází.

Druhý způsob detekce textu ve videu využívá předpokladu, že počet vlastních bloků v P- a B- snímcích zkoumaných na MPEG komprimovaném videu se zvyšuje v případě výskytu textu.

Další variantou je odhad větší intenzity textu oproti pozadí a následný součet pixelů s nižší intenzitou než je předem definovaná hodnota prahu. Obdélníková oblast s velkým množstvím rozdílného podílu barev textu a okolí je považována za oblast s textem. Tato metoda je neobyčejně rychlá a účinná [4].

Z historického hlediska se tomuto stádiu nepřikládal tak velký důraz, jelikož použití nástrojů OCR pro rozpoznání textu z novinových článků, vytisknutých dokumentů a obálek knih sám o sobě už nesl informaci, že se text ve zkoumaném obrázku nachází.

S ohledem na zpracování snímků v reálném čase při přehrávání videa je nutné, aby byl proces detekce textu co nejméně časově náročný a zároveň přesný.

2.2.2 Lokalizace textu

Lokalizace textu je časově a programově nejvytíženější část systému TIE. Je to způsobeno tím, že v této fázi několikrát procházíme všechny pixely v obrázku, hledáme takové objekty v obrázku, které jsou potencionálním textem, a provádíme s nimi geometrickou analýzu. Tuto fázi můžeme rozdělit na dvě části. Jeden typ lokalizace textu zkoumá oblasti textu nalezené jednou ze dvou metod popsaných níže a druhý typ zkoumá text v závislosti na znalostech textur.

Lokalizace textu na základě regionů využívá znalosti barev nebo stupňů šedi jednotlivých pixelů a jejich vzájemného rozdílu oproti okolí (pozadí obrázku). Takové metody lze dále rozdělit ještě na dvě další, se kterými se v práci setkáme:

- **Metoda založena na hledání spojitých oblastí**
- **Metoda založena na detekci hran**

Druhý typ lokalizace se soustředí na vlastnosti textur v obraze. Využívají faktu, že text v obrázku má výrazně odlišné texturní vlastnosti oproti okolí. Lokalizační techniky jsou založeny na *Gaborova filtru*, *FFT* nebo na *prostorovém rozptylu*. Celkově se tedy využívají k detekci texturních vlastností oblasti s textem v obrázku.

V práci jsem se především zaměřil na první typ lokalizace textu a implementaci jeho metod.

Doposud jsme uvedli pouze základní vlastnosti lokalizačních metod, aby si čtenář dokázal udělat představu o způsobu nalezení textu. K oběma metodám se vracíme v kapitole 3, kdy si je podrobně popíšeme, znázorníme postupy s obrázky a následně v kapitole 4 a 5 použijeme v praxi.

2.2.3 Sledování textu

Sledování textu se týká výhradně videa a v dnešní době je standardní částí TIE, avšak ne nezbytnou. Vzhledem k tomu, že předešlé kroky systému TIE mohou být velmi časově náročné a je zapotřebí zachovat plynulý chod videa, může se tato část vynechat na úkor časové složitosti použitých algoritmů. Navíc na tuto část nebyla z historického hlediska pozornost moc upřena.

Po detekci a lokalizaci textu (v tomto případě ve videu) do této části vstupují body (respektive oblasti), které jsou označeny jako potencionální části textu. Sledování těchto bodů je těžištěm této pasáže TIE. V případě, že se body v bezprostředně následujících snímcích vytratí nebo posunou na jiné místo, můžeme vyloučit tyto body jako text, protože počítáme s textem nepohyblivým, tedy se statickými titulky. Kdybychom přesto chtěli detekovat i pohyblivý text, musely by všechny vektory tvořící se z počátečních bodů pozorování z obrázku *n* a koncových bodů pozorování z obrázku *n+1* mít stejnou délku i směr. Tento princip by se samozřejmě několikrát opakoval. Poté bychom vyhodnotili text na obraze jako pohyblivý. V závislosti na směru pohybu bychom poté mohli určit, zda se jedná o text pohybující se na jedné nebo více osách, či jestli se text pohybuje po obrázcích zcela náhodně a nekontrolovaně.

2.2.4 Extrahování textu

Poslední část systému TIE představuje samotné extrahování textu. Zde se pracuje s geometrií získaných bodů z předešlých kroků. Jde o to smysluplně (v obdélníku) uspořádat souvislý text. Záleží do jisté míry taky na metodách lokalizace, které mohou dvouřádkový text označit jako jedno pole nebo dvě pole. Nic to však nemění na způsobu extrahování textu. Úkolem je daný obdélník z původního obrázku vystřihnout a předložit jej programu Tesseract, který se postará o rozpoznání textu.

Tesseract je volně dostupný OCR nástroj, který byl vyvinut společností HP v letech 1984 až 1994. Teprve v roce 2005 byl zpřístupněn veřejnosti. Tesseract nejdříve rozdělí vstupní obrázek na oblasti textové a netextové. Má 3 metody pro zpracování textu z obrazu. Bottom-up metoda klasifikuje malé části obrazu (pixely, skupiny pixelů nebo spojité oblasti) a navzájem je slučuje. Top-down metoda rozdělí obraz rekurzivně ve vertikálním i horizontálním směru v závislosti na výskytu prázdných oblastí v obraze, které představují hranice sloupců nebo odstavců textu. Třetí metoda je založena na analýze prázdných oblastí v obraze. Ta řeší nedostatky top-down metody nalezením mezer mezi sloupci bottom-up metodou. Zdroj [13].

Před samotným extrahováním textu můžeme ještě provést optimalizaci. Ta spočívá ve zvýraznění nalezených hran v oblasti s textem, ustoupení pozadí například sjednocením do jedné barvy, a tím pádem dosažením většího kontrastu textu oproti pozadí.

Opět ale musíme počítat s časovou náročností v případě zpracovávání obrázků z videa a vyhodnotit, jestli si můžeme dovolit optimalizovat oblast s textem na úkor času.

Nyní známe všechny základní vlastnosti textu vyskytujícího se v digitálních snímcích a pohled na metody, které nám umožní text detekovat, lokalizovat, sledovat a extrahovat.

3 Segmentace obrazu

Tato kapitola se zabývá segmentací obrazu s cílem rozdělit obrázek na oblasti s textem a bez textu. Za účelem nalezení textu v obrazu se provádí transformační změny obrázků v podobě předzpracování obrazu, převzaty z [1] a [2], konkrétně prahování, zvýraznění hran a odstranění šumu. Dále se aplikují detekční a lokalizační metody systému TIE. Zaměříme se zprvu na celkovou segmentaci obrazu a následně na lokální segmentaci zajímavých oblastí.

V této kapitole je popsána metoda spojitých oblastí dle [1] a [2] se třemi aplikacemi, konkrétně split and merge, watershed a shlukové analýzy. V závěru třetí kapitoly je popsána metoda detekce hran z [1] a [6]. Obě tyto metody jsou určeny pro vyhledávání textu v obraze.

Všeobecným cílem segmentace obrazu je oddělit zkoumané objekty od ostatních.

3.1 Předzpracování

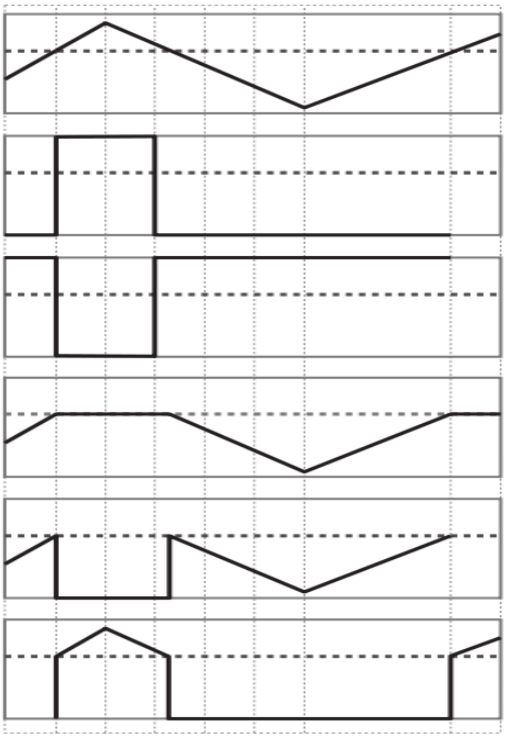
Předzpracování digitálního snímku je nedílnou součástí celého procesu nalezení textu v obrázku. Bez technik předzpracování bychom nemohli následně využívat lokalizační algoritmy. Jedná se o práci s barevností obrazu, světlostí a částečně i texturami. Používají se globální segmentační techniky, které se aplikují na všechny pixely v obraze, nejen na některé části. Postupně si ukážeme funkci *prahování*, *zvýraznění hran* a *odstranění šumu*.

3.1.1 Prahování

Za prahování považujeme jednoduchý segmentační proces, při kterém přepočítáme každému pixelu v obraze jeho novou hodnotu podle typu použitého prahování (obrázek 3.1). Ve většině případů se jedná o porovnání hodnoty pixelu s hodnotou prahu a následné operaci v závislosti na vyhodnocení těchto dvou veličin.

Prahování je metoda pracující s obrázkem ve stupních šedi. V případě, že je vstupní obrázek barevný, je ho třeba dle rovnice 3.1 převést do škály šedi, což můžeme považovat taky za část procesu týkající se předzpracování obrazu. Celočíselný výsledek *value* představuje novou hodnotu intenzity pixelu v rozsahu od 0 do 255 na základě hodnot proměnných *r*, *g* a *b*, které určují původní barvu pixelu.

$$value = 0.299*r + 0.587*g + 0.144*b \tag{3.1}$$



Obrázek 3.1: Druhy prahování. Zdroj [2].

Na obrázku 3.1 je několik typů prahování. Jako první shora je uvedena linie představující několik pixelů za sebou (vertikální osa) s proměnlivou barevností (svislá osa). Následně je znázorněno prahování binární, inverzní binární, ořezané, prahování na nulu inverzní a prahování na nulu. Například binární prahování z obrázku se škálou šedi udělá obrázek černobílý, kdy pixely s nižší hodnotou, než je úroveň prahu, budou černé a pixely s vyšší hodnotou, než je úroveň prahu budou bílé dle následujícího vztahu:

$$dst = (src > t)\ ?\ M : 0 \tag{3.2}$$

Dst představuje výslednou hodnotu pixelu, která se odvíjí od porovnání aktuální hodnoty vstupu *src* *s* hodnotou prahu *t*.

Základní průchod obrázkem při prahování je ukázán na algoritmu 3.1.

Algoritmus 3.1: Základní princip prahování.

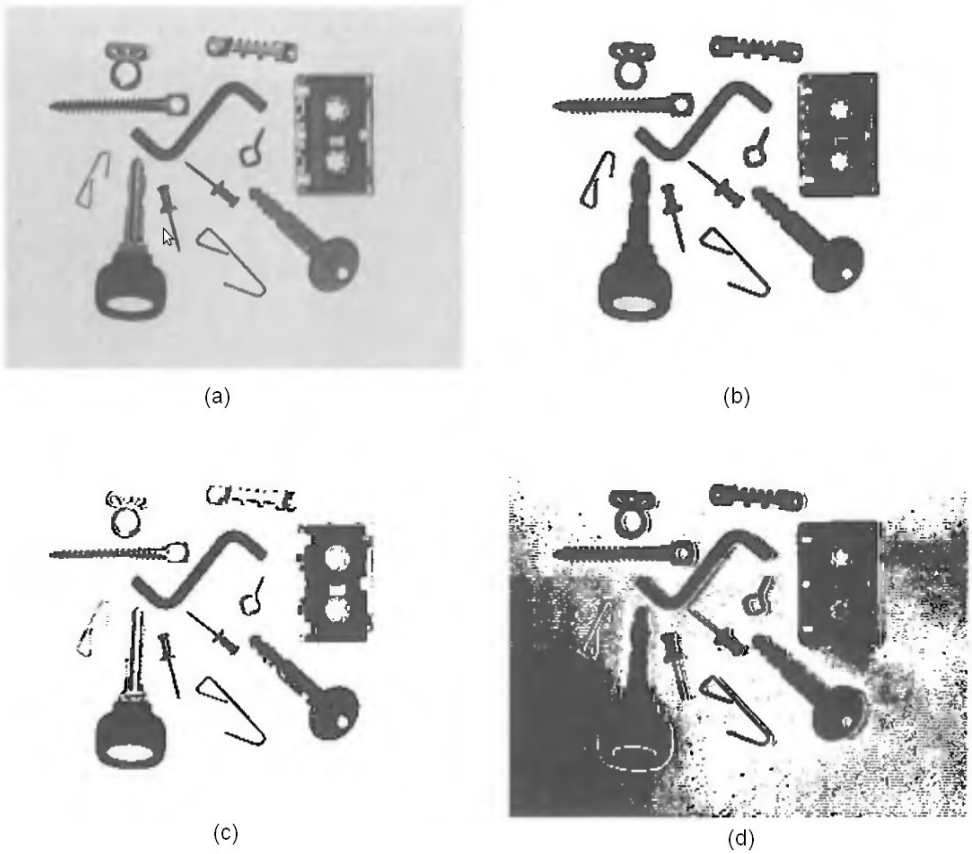
---

1) Urči hodnotu prahování *t* v rozmezí *0*-255 a druh prahování *g*.

2) Prohledej nezkoumaný pixel *f(i,j)* obrázku *f*.

3) Porovnej aktuální hodnotu pixelu *f(i,j)* s hodnotou prahu *t* a proveď operaci s *f(i,j)* dle *g(t)*.

Na následujícím obrázku jsou vidět rozdílné hodnoty prahu použité na jeden obrázek. Při tvorbě automatizovaného nástroje na hledání textu s využitím metod prahování je použitá hodnota prahu nesmírně důležitá. V případě její špatné volby může některý text v obrázku po prahování zaniknout.



Obrázek 3.2: Ukázka binárního prahování. a) originální obrázek; b) střední hodnota prahu t = 128; c) nízká hodnota prahu t = 64; d) vysoká hodnota prahu t = 192; Zdroj [1].

Kdybychom nevytvářeli plně automatizovaný nástroj na hledání textu, využili bychom manuálního výběru prahu při prahování. Na základě velkého vstupního množství testovacích dat a znalosti, že intenzita textu je vysoká, používáme práh o hodnotě 200. V kapitole 4 se ještě k prahování vrátíme s konkrétním typem prahování použitým v práci a ukázkami.

Doposud jsme popsali funkci klasického prahování. Nicméně existují ještě prahovací metody přizpůsobující se okolí (*adaptivní prahování*) a optimální metody prahování.

Adaptivní prahování je dle [1] modifikací klasického prahování, kdy může být práh proměnlivý v závislosti na okolí prohledávání. Pro každý zkoumaný pixel se vypočítá průměrná hodnota pixelů v jeho okolí (např. 3x3). Výsledkem může být, že všechny pixely v okolí mohou být vyhodnoceny stejným způsobem prahování se stejným prahem anebo hodnoty nových pixelů daného regionu mohou být vyhodnoceny pomocí *Gaussovy funkce* vzhledem ke vzdálenosti od středu okolí.

Druhým způsobem prahování je *prahování optimální*. Dle [1] využívá aproximace výsledků histogramu z původního obrázku s použitím váženého součtu dvou nebo více hustot pravděpodobností s normálním rozložením. Práh je nastaven na nejbližší stupeň šedi korespondujícího k minimu mezi maximem dvou nebo více pravděpodobností normálního rozložení.

3.1.2 Zvýraznění hran

Účelem zvýraznění hran v obraze je dosažení lepšího oddělení jednotlivých objektů od sebe. Používá se vysokofrekvenční filtr, který zdůrazní členy s velkými změnami a potlačí změny malé. V obraze uchová lokální kontrast a malé rozdíly ve stupních šedi.

Dle [1] se implementuje se ve třech krocích:

1) *vysokofrekvenční filtr* obsahující informace o hranách (použití masek 3x3 nebo 9x9)
2) k obrazu vzniklému filtrací přičteme původní obraz
3) roztáhneme kontrast složeného obrazu

Zvýraznění hran v obraze dosáhneme také použitím *Robertsova gradientu*, který přepočítá nové hodnoty *DH(i,j)* z původních hodnot pixelu *f(i,j)* následovně:

$$DH(i,j) = [f(i,j) - f(i+1, j+1)] + [f(i+1, j) - f(i, j+1)] \tag{3.3}$$

Nebo lze také použít *Prewittův operátor*, který považuje kolmý a diagonální pixel diferenciály za stejný. Aproximuje první derivaci, gradient je počítán v osmi možných směrech s maskou o rozměru 3x3. Výsledky konvoluce největšího rozsahu indikují směr gradientu. V následujících maticích 3.3a a 3.3b jsou příklady masek velikosti 3x3.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Obrázek 3.3a: Prewittův operátor.

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Obrázek 3.3b: Sobelův operátor.

3.1.3 Odstranění šumu

Jako šum jsou v digitálním obraze představovány místa s náhodnými chybami, někdy taky *spam*. Šumu je možné se zbavit například prahováním, detekcí hran nebo binarizací.

Binarizace je proces převedení barevného digitálního obrázku nebo obrázku v úrovni šedi na černobílý a v práci ji řadím do fáze předzpracování.

Dalšími metodami pro odstranění šumu jsou například využití nízkofrekvenčních filtrů, Gaussova filtru nebo použití anizotropní difuze.

3.2 Metoda spojitých oblastí

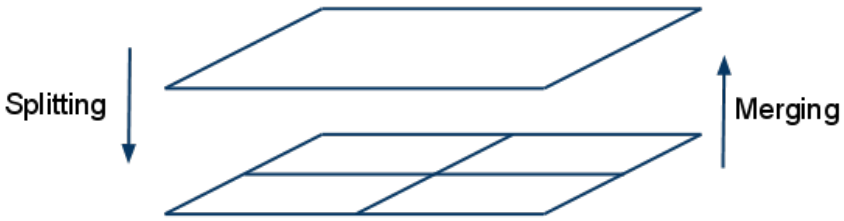
Metoda spojitých oblastí (*connected* component *method*) je označení procesu, kdy postupně z celého obrázku zkoumáme malé oblasti a na základě předem definovaných pravidel (viz níže) seskupujeme tyto oblasti do oblastí větších až do doby, kdy sousední oblasti již sloučit nelze dle stanovených pravidel. Samotný proces slučování se označuje anglickým termínem *merge*. Hledáme oblasti v obraze, které mají společné vlastnosti. Při vlastní implementaci metody založené na hledání spojitých oblastí s textem jsem využil principu *split and merge* algoritmu.

Geometrická analýza je potřebná ke sloučení textových oblastí pomocí prostorového uspořádání oblastí tak, aby odfiltrovala netextové oblasti a označila správně oblasti s textem.

V této podkapitole si uvedeme několik konkrétních aplikací.

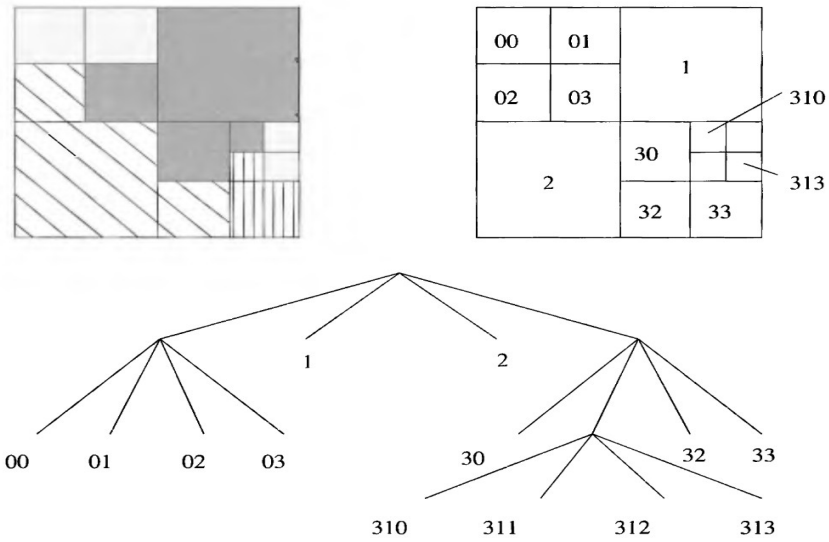
3.2.1 Split and merge

Split and merge je označení metody pro rozdělení a spojení obrázku, respektive jeho částí. Je to kombinace dvou algoritmů - rozdělení původního obrázku na několik oblastí a následné sjednocení těchto oblastí (*splitting* a *merging)*. Vše se děje na základě předem daných pravidel.



Obrázek 3.4: Ukázka splittingu a mergingu.

Algoritmus *split and merge* reprezentuje obrázek v podobě pyramidy. Jednotlivé regiony obrázku jsou převážně čtvercového tvaru a odpovídají jednotlivým úrovním pyramidy. Tento princip je znázorněn na obrázku 3.5.



Obrázek 3.5: Segmentace obrázku pomocí split algoritmu a znázornění jednotlivých regionů pomocí stromu. Zdroj [1].

Zkoumá se region obrázku (prvním bude celý obrázek) a pokud není homogenní (s výjimkou nejnižšího stupně pyramidy), rozdělí se na 4 podregiony o stejných rozměrech. Jestliže tyto 4 podregiony existují v jakémkoliv stupni pyramidy s přibližně stejnou hodnotou homogenity, jsou sloučeny do jediné oblasti ve vyšší vrstvě pyramidy. *Split and merge* metody obvykle ukládají informace o svých sousedech (regionech) v grafech.

Důležitým kritériem tohoto algoritmu je určení kritéria homogenity, dle které se regiony rozdělují a následně slučují. Za kritérium homogenity se dá považovat barva, intenzita, pozice nebo jakákoliv jiná významná vlastnost pixelu nebo regionu.

Algoritmus 3.2 uvádí základní postup při použití metody *split and merge*.

Algoritmus 3.2: Postup split and merge algoritmu. Zdroj [1].

---

1) Definuj počáteční segmentační oblast, kritérium homogenity a pyramidovou strukturu.

2) Jestliže nějaký region R v datové struktuře pyramidy H není homogenní [$H(R) = FALSE$], rozděl ho do čtyř regionů a označ je jako potomky. Jestliže jakékoli čtyři regiony se stejným předkem mohou být sjednoceny do jediného homogenního regionu, sjednoť je. Pokud žádný region nemůže být dále rozdělen, jdi na krok 3.

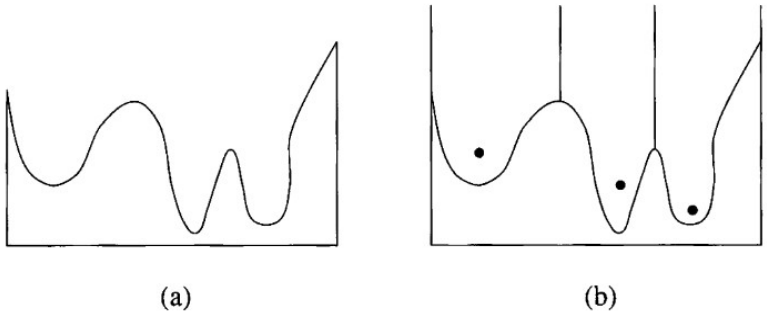
3) Jestliže nějaké dva sousední regiony R_i, R_j (i v případě že se nachází v jiných stupních pyramidy nebo nemají stejného rodiče) mohou být sjednoceny do homogenního regionu dle kritéria homogenity, sjednoť je.

4) V případě, že je nutné odstranit malé regiony, sjednoť malé regiony s nejvíce podobnými sousedními regiony.

Jestliže si například dle algoritmu 3.2 určíme kritérium homogenity intenzitu na obrázku ve škálách šedi o hodnotě 10, porovnáme dva sousední regiony R_i a R_j dle kritéria homogenity $abs(R_i - R_j)$, kde *abs* vyjadřuje absolutní hodnotu po rozdílu hodnot dvou intenzit R_i a R_j. V případě, že bude výsledná absolutní hodnota nižší než 10 (kritérium homogenity), můžeme tyto dva regiony spojit v jeden.

3.2.2 Watershed

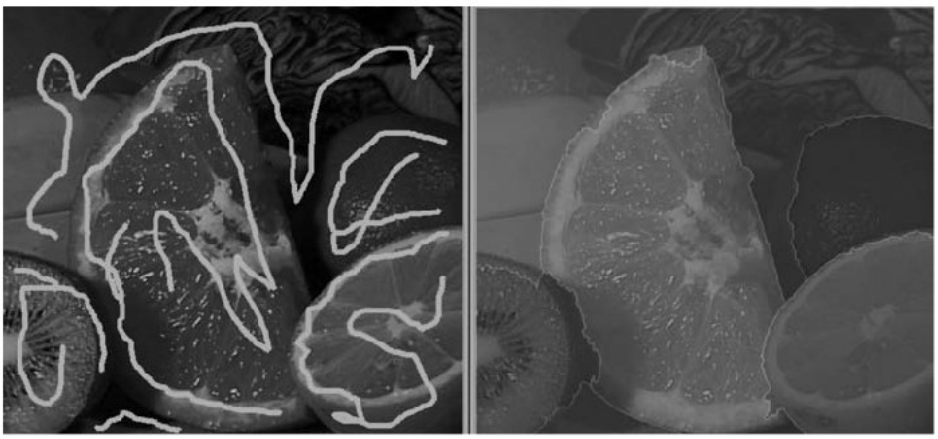
Watershed je označení další metody segmentace obrázku. Dle [1] je založena na rozdělení obrázků do tzv. povodí. Nejlepší bude, když si situaci vysvětlíme na obrázku 3.6.



Obrázek 3.6: Ukázka watershedu. Zdroj [1].

Na obrázku 3.6 je ukázka jedno-dimenzionálního příkladu na *watershed*. V části a) je profil dat z obrázku ve škále šedi, na b) je ukázka segmentace obrázku do jednotlivých povodí. Svislé čáry označují lokální maxima a definují hranici povodí. Tečky se nacházejí v lokálních minimech a určují oblasti patřící do daných povodí. Výsledná segmentace představuje rozdělení obrazu do několika povodí s podobnými vlastnostmi.

Algoritmus nejdříve určí střední hodnotu intenzity v obraze. Ta má důsledek na tvorbu jednotlivých povodí. Body v údolích se postupně zaplavují do doby než jsou všechna povodí spojena. Tento algoritmus taky umožňuje uživateli určit oblast, která je pozadím obrázku a tak usnadnit segmentaci všech objektů do jednotlivých povodí.



Obrázek 3.7: Algoritmus watershed. Zdroj [2].

Poté co si uživatel zvolil v obrázku 3.7 objekty, které patří k sobě (vlevo), algoritmus *watershed* sjednotil označené oblasti do segmentů (vpravo).

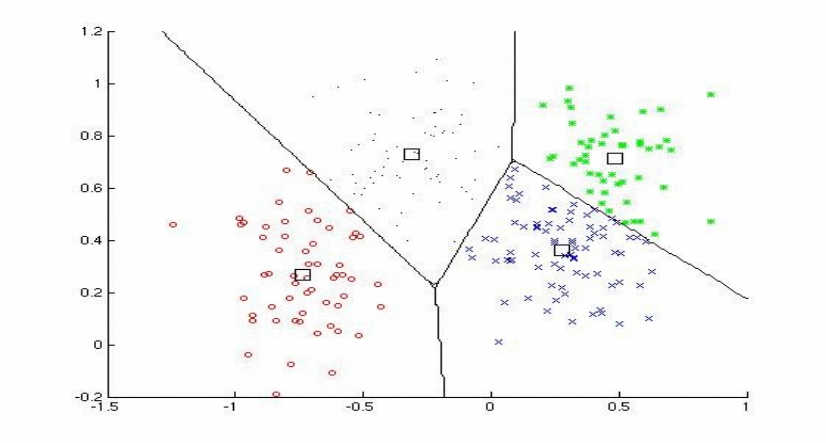
3.2.3 Shluková analýza

Shluková analýza je označení metody shlukování. Je to forma reprezentace dat, při které dochází ke snížení objemu relevantních dat. Dle [1] si na počátku určíme množinu atributů, dle kterých budeme data z obrázku segmentovat do tzv. *shluků*. Máme množinu objektů $O=\{O_1,...O_n\}$ a míru vzdálenosti objektů V . Poté se shluk uvádí jako podmnožina $X \in Y$, pro kterou platí:

$$max(V(O_i,O_j)) < min(V(O_k,O_i)), O_i, O_j \in X, O_k \notin X \tag{3.4}$$

Nejpoužívanější metodou shlukování je jednoznačně *k-means clutering* [2], kde je každý shluk reprezentován svým středem. Pro *k* shluků si nejdříve určíme *k* objektů, které budou představovat neprázdné shluky. Vzápětí se spočte střed (těžiště) každého shluku. Následně procházíme obrázek a jednotlivé pixely přiřazujeme do právě toho shluku, k jehož středu má pozice pixelu nejkratší vzdálenost. V případě, že nastane změna při přiřazení, vrátíme se o dva kroky zpět na přepočtení těžiště shluku.

Shluková analýza tak může v ideálním případě (správné určení míry vzdálenosti objektů a středů hluků) do několika shluků uložit pozice textových objektů v obraze a do jiných zase oblasti, ve kterých se text nenachází. Není to optimální řešení, a proto jej v práci neimplementuji.



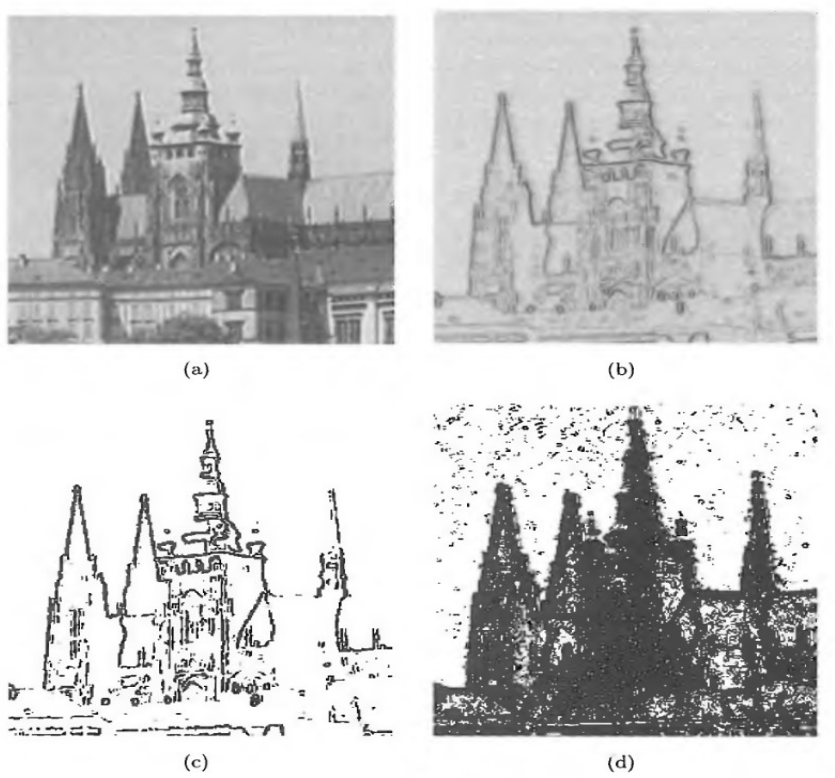
Obrázek 3.8: Ukázka k-means clusteringu.

3.3 Metoda detekce hran

Tato metoda se soustředí na hledání hran v obraze, tedy na oblasti, kde dochází k vysokému kontrastu mezi textem a jeho pozadím. Je jednou z nejstarších segmentačních technik v obraze, dodnes se využívá jako jedna z předních metod a je spojena s prahováním. Abychom nalezli v obraze hrany, je zapotřebí použít tzv. detektorů hran. Ty hledají taková místa v obraze, kde se výrazně mění intenzita sousedních pixelů. Po dalším průchodu algoritmů, které především řeší geometrické vlastnosti textu, jsou odfiltrovány netextové části a nalezené textové oblasti jsou zvýrazněny.
Existuje velké množství detektorů hran, které počítají s hledáním hran ve 2, 4 nebo více směrech. Některé jsme si uvedli v sekci o zvýraznění hran. Zde se budeme zabývat nejpoužívanějším detektorem hran - *Cannyho*.

Ještě se zmíníme o prahování spojeném s hledáním hran v obraze. V obrázku s hranami se téměř žádné nulové hodnoty pixelů nevyskytují. Malé hodnoty pixelů hran odpovídají nepodstatným změnám ve škálách šedi vyplývajících z šumu nebo světelných nepravidelnostech v obraze.

Na obrázku 3.9 je ukázka detekování hran v obraze s různě velkým prahem.



Obrázek 3.9: Ukázka prahování na obrázku s hranami. a) původní obrázek; b) nízký kontrast hran pro lepší zobrazení; c) práh nastaven na hodnotu 30; d) práh nastaven na hodnotu 10; Zdroj [1].

Jeden z rozdílů mezi *Cannyho detektorem hran* a jednoduššími způsoby (*Laplacův algoritmus*) je spočtení první derivace v *x* a *y* a následné spojení do čtyř směrů derivací. Body, kde tyto směrové derivace dosahují lokálního maxima, jsou označeny jako kandidáty pro nalezené hrany.

Je to algoritmus vhodný na užití obrázku s bílým šumem. Výhody této metody jsou v určení přesnosti výskytu hrany, její jednoznačnosti a malém počtu chybě detekovaných hran.

Dle [6] *Cannyho algoritmus* zahrnuje čtyři kroky:

- Odstranění šumu z obrazu pomocí Gaussova filtru
- Aplikace Sobelova filtru pro nalezení velikostí gradientů
- Ztenčování
- Prahování

Ztenčování (*thinning*) přitom představuje metodu redukce oblastí v obraze na pouhé úsečky či křivky o tloušťce jednoho pixelu.

[1] uvádí tři kritéria, která jsou důležitá pro optimální nalezení hran pomocí *Cannyho algoritmu*:

- **Detekce** je kritérium pro nalezení významných hran, které nesmějí být v obrázku přehlédnuty. Také je třeba se vyhnout falešným znakům netvořící hranu.
- **Lokalizace** říká, že vzdálenost mezi aktuální a lokalizovanou hranou by měla být minimální.
- **Odpovědní kritérium** minimalizuje více odpovědí (označení) na jednu hranu. Vypořádá se s problémem špatné hrany způsobené šumem a nehladkými operátory hrany.

Postup při nalezení hran pomocí *Cannyho detektoru* zobrazuje algoritmus 3.3.

Algoritmus 3.3: Cannyho detektor hran. Zdroj [1].

---

1) Odstraň šum z obrazu *f* pomocí Gaussova filtru σ.
2) Ohodnoť směr lokální hrany *n* použitím rovnice 3.3 pro každý pixel v obraze.
3) Lokalizuj hranu pomocí rovnice 3.4.
4) Vypočítej velikost hrany pomocí rovnice 3.5.
5) Využij prahování na hrany v obraze s hysterezí za účelem eliminace několikanásobného označení hrany.
6) Opakuj kroky 1 až 5 pro vzrůst hodnot směrodatné odchylky σ.
7) Vyhodnoť výsledné informace o hranách ve více ohledech na základě společných rysů.

$$n = \frac{\Delta(G * f)}{|\Delta(G * f)|} \tag{3.5}$$

Rovnice 3.5 spočte kolmý směr k hraně *n* na základě gradientu Δ, Gaussovy funkce G a obrázku *f*.

$$\frac{\delta^2}{\delta n^2} G * f = 0 \tag{3.6}$$

Rovnice 3.6 nám říká, jak najít lokální maximum ve směru kolmému k hraně, kde δ představuje parciální derivaci funkce G k *n*.

Nakonec je třeba znát ještě rovnici pro výpočet velikosti hrany:

$$|G_n * f| = |\Delta (G * f)| \tag{3.7}$$

Další zmínka o *Cannyho detektoru hran* bude v kapitole o návrhu a implementaci, kde pomocí obrázků ukáži jeho použití v praxi.

Informace k této sekci byly čerpány převážně z [1], [2] a [6].

4 Návrh řešení

V této kapitole je popsán vlastní návrh řešení automatizované aplikace na nalezení a extrakci textu z obrázků a videí. Nejdříve popíšu způsob načtení a zobrazení snímků s podporovanými vstupními formáty. Dále se zmíním o návrhu dvou metod pro nalezení textu v obraze a geometrické analýze. Následuje podkapitola o rozpoznání textu z obrázku a exportu textu do textového dokumentu. Na závěr popíšu nedokonalosti návrhu aplikace.

4.1 Načtení a zobrazení snímků

Vstupem aplikace je digitální obrázek ve tvaru **.bmp, *.jpeg, *.jpg, *.png, *.tiff* nebo **.tif*. Pro videa je možné použít formát s příponou **.avi* nebo **.mpg*. Uživatel je tedy částečně omezen podporovanými formáty. Má to jediný důvod - kompatibilitu formátů s vývojovým prostředím *QT verze 4.7* a knihovny *OpenCV 2.2*.

Pro uživatelskou přívětivost je zahrnuta ještě možnost otevření celé složky, ve které se vyskytují podporované soubory pro zobrazení. Po jejím nahrání má uživatel možnost pomocí grafického rozhraní aplikace přecházet mezi jednotlivými soubory z načtené složky a urychlit si tím práci při testování větší sady vstupních dat.

Obrázek 4.1: Panel pro ovládání aplikace.

4.2 Nalezení textu

V případě, že je načten obrázek, zahájí se sada algoritmů vedoucích k nalezení textu v obrázku. Abychom dokázali text nalézt s co největší pravděpodobností, zvolil jsem kombinaci několika metod zde uvedených. Největší podíl ze systému TIE tvoří část týkající se lokalizace textu.

Úkolem *detekce* textu je hlavně fáze předzpracování obrázku. Barevný obrázek se nejprve převede do stupňů šedi. Následně je na obrázek použito binární prahování s pevným prahem o velikosti 200. Do fáze detekování textu se dají zařadit ještě úvodní metody pro lokalizaci textu. A to fáze nalezení hran v obraze a fáze nalezení spojitých oblastí. V případě, že jsme nenalezli jakékoliv hrany nebo oblasti, fáze detekování textu se považuje za neúspěšnou, jinak jsou předány výsledky detekčních metod do druhé fáze k dalšímu zpracování.

Do části *lokalizace* jsou v případě zpracování algoritmu nalezení hran začleněny funkce, které pracují s oblastmi, ve kterých se ostré hrany vyskytují. Jsou to seznamy bodů reprezentující nalezenou křivku v obraze. V *OpenCV* pracujeme se sekvencemi objektů, kde se každý z nich

odkazuje na svého následovníka při vyjádření celého popisu křivky. Tato sekvence objektů je utvořena pomocí funkce hledající hrany v obraze *cvFindContours()*.

V případě, že jsou oblasti s ostrými hranami v obraze nalezeny, provede se s nimi geometrická analýza. V prvním průchodu jsou vyloučeny takové oblasti, které jsou příliš velké (nebo malé) na to, aby představovaly potencionální znaky textu. V dalším průchodu se provede porovnání vzdáleností jednotlivých oblastí mezi sebou - tedy na ose *x*. Jako text jsou vyloučeny takové oblasti, které jsou osamostatněné nebo mají příliš rozdílnou vzdálenost od dalších potencionálních znaků textu. V následující fázi se porovnají pozice oblastí vzhledem k ose *y*. Jde o spočtení průměrné výšky oblastí v jednotlivých řádcích obrázku a následné srovnání odchylky od této průměrné hodnoty. Jsou tak vyřazeny oblasti, které nevyhovují provedené geometrické analýze. Závěrem této metody je ještě jeden průchod oblastmi s ostrými hranami, kdy se zohlední minimální počet nalezených oblastí v jednom řádku.

Druhá metoda *spojitých oblastí* má obdobný průběh jako předcházející metoda. Vstupem je seznam oblastí, které se nalezly během metody hledání spojitých oblastí. Tyto oblasti jsou rozděleny podle intenzity jednotlivých pixelů do tří skupin (dle intervalů intenzity[3]). Po geometrické analýze (viz výše) je počet nalezených spojitých oblastí značně eliminován.

Výsledkem lokalizace je kombinace dvou výše uvedených metod. Porovnají se aktuální pozice nalezených hran a spojitých oblastí. V ideálním případě by každý znak nalezeného textu v obraze představoval dvojici oblast s ostrou hranou - spojitá oblast. V poslední části je třeba tyto oblasti zanalyzovat a v případě, že se dvojice nevytvoří a vše nasvědčuje chybné lokalizaci textu, je třeba považovat nalezené oblasti za takové, které text neobsahují.

Ve vlastní práci volím nastavení kritérií pro minimální a maximální velikost textu účelně a obezřetně a vždy vzhledem k aktuální velikosti obrázku dle tabulky 6.1. Minimální šířka písmene určit nejde. V případě malého obrázku a výskytu například slova *lily* fontem nepoužívající patky písmen, můžeme v některých případech obdržet šířku prvních třech písmen 1 pixel a tudíž nemá smysl v tomto směru určovat minimum šířky písmene.

V případě, že se načte video, jsou lokalizační algoritmy totožné jako v případě hledání textu v obraze. Navíc zde ale dochází ke sledování stabilních oblastí ve videu, a tak v prvním kroku můžeme vyloučit takové oblasti, které v několika po sobě jdoucích snímcích vykazují změnu intenzity, a tím pádem jsou to oblasti s pohybující se scénou, nikoliv stálým textem. Teprve poté, co lokalizační algoritmy určí oblasti s výskytem textu, shodnou se na stejných oblastech a zároveň algoritmy starající se o určení stabilních oblastí v textu poukáží na stejné oblasti, je oblast označena jako oblast obsahující text.

3 Více o konkrétních intervalech intenzity v metodě hledání spojitých oblastí v kapitole 5.

4.3 Export do textového dokumentu

Do této fáze vstupují objekty, které jsou detekčními a lokalizačními metodami vyhodnocené jako potencionální text. Před rozpoznáním je zapotřebí ohraničit tyto objekty, což je velice jednoduché vzhledem k tomu, že si objekty nesou informace o pozici a velikosti v obrázku. Určí se potřebné poziční souřadnice minim a maxim objektů a celý text se ohraničí rámečkem v podobě obdélníku.

S extrahováním jen potřebné části obrázku poté provedeme rozpoznání textu v obraze pomocí nástroje *Tesseract OCR*. Vrácenou hodnotou je holý text, který uživateli zobrazíme v aplikaci. Uživatel má možnost si vizuálně zkontrolovat nalezený text, uložit si jej do textového dokumentu, zahodit ho nebo si ho postupně ukládat, a až přijde čas (video skončí), uloží si ho jako celek.

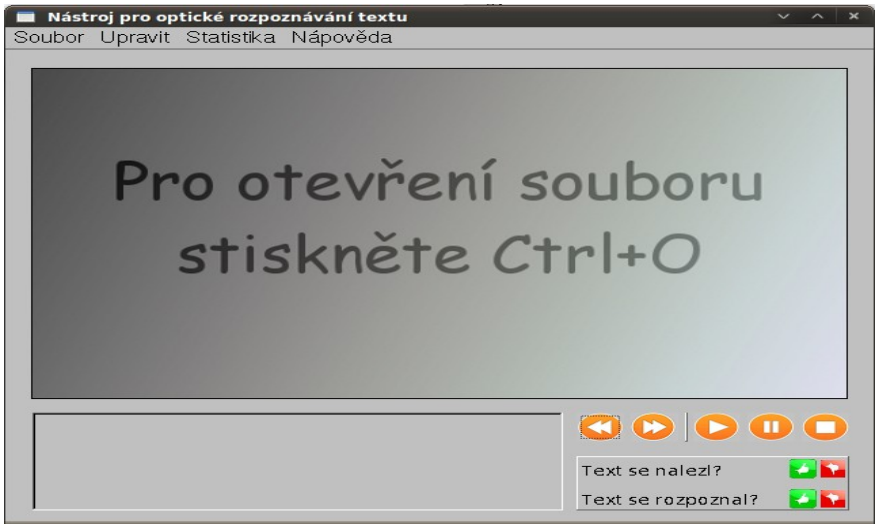
4.4 Nedokonalosti návrhu

V návrhu se nepočítá s časovou složitostí některých zamýšlených algoritmů, což způsobuje nemalé problémy při použití těchto algoritmů v praxi. Čas potřebný pro průchod všech implementovaných algoritmů několikrát přesahuje čas pro plynulé přehrávání videa, což může způsobit uživatelsky nepřívětivý program. Samotná implementace všech výše zmíněných algoritmů je proveditelná převážně za pomocí knihovny *OpenCV*. V následující kapitole se ke konkrétní časové složitosti algoritmů vrátíme.

Dále návrh předpokládá vhodně zvolená vstupní data. Tím je myšlena snaha se vyhnout například nadměrné velikosti obrázků, kdy na velké rozlišení připadá text v obraze velmi malý anebo v opačném případě vysoká komprese snímků, znekvalitnění čitelnosti zašuměním nebo malá velikost obrázku. Všechna tato fakta vedou k nepřesnému vyhodnocení lokalizačními metodami o nalezení textu a nástroj se tím pádem stává méně kvalitním a celkově nepřesným.

5 Implementace

V předchozí kapitole jsem uvedl návrh aplikace a popsal konkrétní použitých algoritmů a dalších metod pro nalezení textu v obraze. Zde si názorně ukážeme efektivitu daných algoritmů na konkrétních příkladech. Při implementaci jsem použil dostatečné množství testovacích příkladů, aby lokalizační algoritmy mohly správně najít text v rozsáhlé sadě vstupních dat. Zmíním se také o použitých datových strukturách.



Obrázek 5.1: Pohled na grafické rozhraní aplikace.

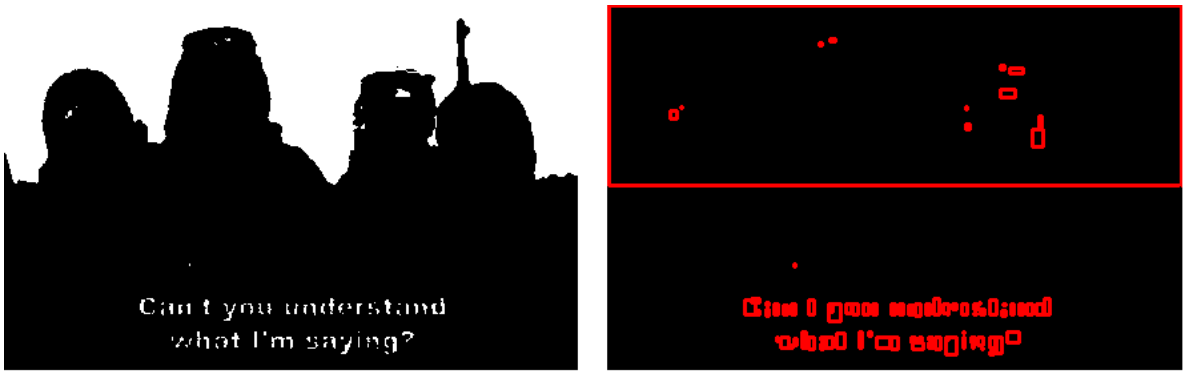
5.1 Hledání oblastí s ostrými hranami

Vybereme si jeden testovací obrázek, na kterém postupně ukážeme algoritmy použité v práci na hledání textu. Nejprve převedeme barevný snímek na snímek ve stupních šedi (obrázek 5.2).

a) b)

Obrázek 5.2: a) původní obrázek; b) obrázek ve škále šedi;

Následně použijeme binární prahování s prahem 200 (obrázek 5.3a) a využijeme funkci z *OpenCV* pro nalezení oblastí s ostrými hranami v obraze *cvFindContours()*. Na obrázku 5.3b jsou nalezené oblasti vyobrazeny pomocí funkce pro kreslení obdélníků *cvRectangle()*.



a) b)

Obrázek 5.3: Ukázka použitých funkcí z OpenCV knihovny. a) cvThreshold() s prahem 200;

b) cvFindContours() + cvRectangle();

Následuje geometrická analýza, která vyhodnotí velikosti a pozice jednotlivých oblastí s ostrými hranami. Pokud jsou oblasti příliš malé nebo příliš velké vůči celkové velikosti obrázku, nepočítáme s nimi v dalších částech systému TIE. Dále porovnáme, jestli jsou oblasti vyskytující se na podobných *y* souřadnicích přibližně stejně vysoké a široké. Výška oblastí se mezi sebou nesmí lišit o více jak 16% (zjištěno při testování na vhodně zvoleném vzorku dat). V některých případech může dojít k tomu, že počáteční velké písmeno se nemusí začlenit mezi další malá písmena při lokalizaci textu. Poslední fáze geometrické analýzy se věnuje vzdáleností jednotlivých oblastí mezi sebou na

ose *x*. Postupně jsou tedy vyloučeny oblasti s ostrými hranami, které nejsou považovány jako potencionální text. Výsledek je na obrázku 5.4.

Obrázek 5.4: Eliminace oblastí sostrými hranami po geometrické analýze.

V poslední fázi hledání textu v obraze pomocí hran dochází k seskupení oblastí představující jednotlivá písmena (nebo jejich části) do ucelenějších objektů představující spojení slov (obrázek 5.5). V některých případech se stane, že je dvouřádkový text spojen do jedné oblasti, což je způsobeno velmi malým rozdílem (jednotky pixelů) v *y* souřadnicích nalezených oblastí s ostrými hranami.

Obrázek 5.5: Nalezený text z původního obrázku.

5.2 Hledání spojitých oblastí

Stejně jako v případě hledání oblastí s ostrými hranami budeme využívat část předzpracování obrázku, kdy použijeme převod barevného obrázku do úrovně šedi. Prahování neprovádíme. Každý pixel v obraze má kromě svých vlastností ještě pravdivostní identifikátory, které se používají při prvních průchodech, kdy se tvoří spojité oblasti. V práci jsou implementovány dvě metody pro

nalezení spojitých oblastí. Jedna využívá vestavěné funkce knihovny *OpenCV* s názvem *cvPyrSegmentation()* a druhou jsem zpracoval sám pro ušetření časové náročnosti a získání takových výsledků, které jsou přínosnější pro další zpracování. V podkapitole 4.2 byly zmíněny intervaly, dle kterých se tvoří spojité oblasti. Prvním průchodem obrázku se zjistí všechny přítomné hodnoty škály šedi a vyhodnotí se maximum (*max(f)*) a minimum (*min(f)*) těchto barev. Dalším průchodem se již jednotlivé pixely přiřadí do spojitých oblastí, které se mezi sebou liší intenzitou jednotlivých pixelů ve třech intervalech:

- *< min(f); 1/3 * max(f))*
- *<1/3 * max(f); 2/3 * max(f))*
- *<2/3 * max(f); max(f)>*

Na obrázku 5.6 vidíte rozdíl mezi dvěma metodami použitými pro hledání spojitých oblastí v obraze.

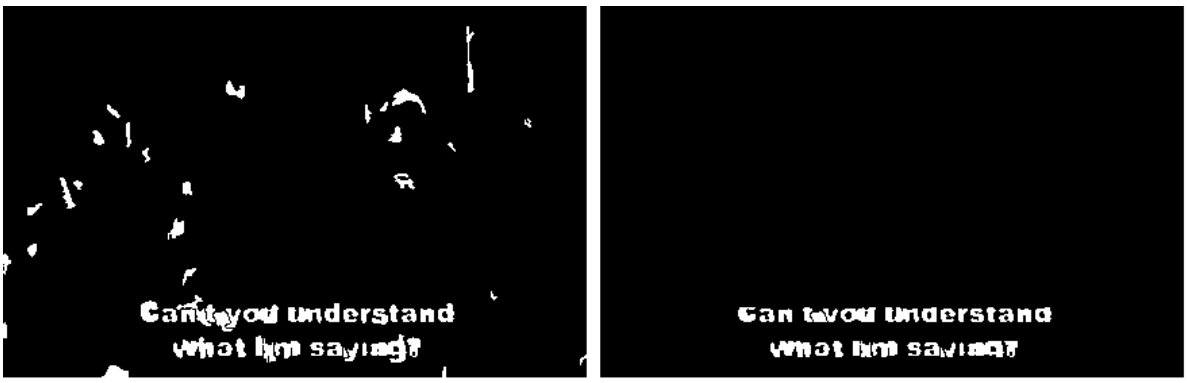
Obrázek 5.6: Segmentace obrázku pomocí spojitých oblastí. *a) vlastní metoda;*

b) cvPyrSegmentation() z OpenCV;

Jednotlivé spojité oblasti jsou implementovány jako vektory struktur obsahující informace o poloze a intenzitě každého pixelu náležícího do spojité oblasti.

Nevýhodou metody *cvPyrSegmentation()*, které je třeba v parametrech funkce předat dvě hodnoty prahu (jeden pro vytvoření vazeb mezi spojitými oblastmi a druhý pro jednotlivé segmenty oblastí), vstupní a výstupní obrázek a taky *úroveň* představující maximální stupeň pyramidy pro segmentaci, je nutnost změnit velikost obrázku na takovou velikost, aby se výška a šířka obrázku dala vydělit dvěma beze zbytku tolikrát, kolik je vrstev v pyramidě dané parametrem *level*. V praxi to znamená pro obrázek o velikosti 1024x1024 možnost užití výšky pyramidy 1-10, pro obrázek 1024x512 výšku 1-9 a obrázek o velikosti 80x100 výšku 1-4. Obrázky, které se nedají dělit dvěma tolikrát, kolikrát je zapotřebí, je třeba před procesem segmentace převést na vhodnou velikost.

Pro metodu *cvPyrSegmentation()* jsem na základě většího počtu testovacích dat využil první práh o hodnotě 40 a druhý o hodnotě 75. Výška pyramidy je dána konstantně čtyřmi patry.



Obrázek 5.7: Použití geometrické analýzy. a) velikosti oblastí; b) pozice oblastí;

Obrázek 5.7a znázorňuje použití geometrické analýzy na spojité oblasti z obrázku 5.6a s ohledem na velikost spojitých oblastí. Obrázek 5.7b poté ukazuje použití geometrické analýzy (podobné jako u hledání oblastí s ostrými hranami) na porovnání poloh jednotlivých oblastí mezi sebou. Výsledkem je stejné označení oblasti výskytu textu jako při použití metody hledání hran v obraze.

6 Testování

Tato kapitola se zabývá kompletním testovaním nad vhodně zvolenými daty. Popisuju zde ruční vyhodnocení přesnosti lokalizačních algoritmů a následnou úspěšnost rozpoznání nalezených písmen v obraze. Následuje podkapitola zabývající se časovou náročností implementovaných algoritmů z kapitoly 5, celkovou časovou náročností systému TIE i analýzou časové náročnosti jednoho pixelu. Na závěr kapitoly jsem navrhl možná vylepšení programu.

6.1 Vyhodnocení přesnosti

Na vhodně zvoleném vzorku dat (100 obrázků) jsem provedl ruční testování s implicitně zvolenými parametry, které jsou v implementaci začleněny. Za testovací data jsem použil volně dostupné obrázky z internetu, které obsahovaly text. Většinou se jednalo o snímek z videa obsahující titulky, obrázky obsahující cedule s názvy ulic, auta s SPZ, přední obálky knih, náhodně vložený text do obrázku nebo taky obrázky, které jsem nafotil v domácím prostředí (kalendáře, knihy, recepty). Text v obrázcích byl z 80% stejnobarevný. Pro testování jsem zvolil různé typy fontů písma s různou výškou i tloušťkou písmen. Vznikla tak široká testovací sada, na které jsem aplikaci testoval.

S ohledem na možnosti obsáhlého využití programu jsem přesto byl donucen k určení minim a maxim velikostí písmen. Tyto hodnoty jsem otestoval nad testovacími daty a dospěl k závěru, že více jak 95% textu se nachází právě uvnitř intervalů z tabulky 6.1.

	Šířka obrázku	Výška obrázku
Minimum	0,6%	3,9%
Maximum	5%	20%

Tabulka 6.1: Minimální a maximální velikosti textu oproti velikosti obrázku v %.

Z výše uvedeného vyplývá, že na obrázek vysoký 300 pixelů a široký 400 pixelů je minimum šířky písmene nastaveno na 2 pixely, minimum výšky na 11 pixelů, maximum šířky na 20 pixelů a maximum výšky písmene na 60 pixelů.

Ručně jsem zaznamenával správný počet nalezených písmen z celkového výskytu písmen v obrázku, počet špatně nalezených oblastí a také vyhodnocení extrakce textu pomocí externího programu *Tesseract OCR*. Zajímavým faktem je, že přes 40% špatně lokalizovaných oblastí neobsahující text tvořily oblasti s okny.

Výsledky jednotlivých testů jsou přiloženy v příloze. Na testovací sadě 100 obrázků bylo celkem správně lokalizováno 2442 znaků z celkového počtu 2932 znaků. Více jak 83% znaků bylo tedy lokalizováno úspěšně. V porovnáním s některými dalšími lokalizačními technikami z [4] a [12], kde se lokalizace testu pohybuje mezi 65% a 90%, uspěl zhotovený program obstojně. Nelze to ale porovnávat přímo, jelikož má testovací sada byla odlišná od sady, kterou používali jiní tvůrci. Dále bylo naměřeno celkově 79 oblastí neobsahujících text na 100 snímků. Je to způsobeno již špatnou

lokalizací textu zaměňující okna za text nebo oblasti, kde dochází k pravidelným změnám ostrých hran a intenzit pixelů, které si jsou zároveň geometricky podobné. Při použití externího programu *Tesseract OCR* pro extrakci textu z nalezené oblasti se dosáhlo 1439 úspěšně rozpoznaných znaků z celkových 2442 nalezených. To představuje více jak 58% správně rozpoznaných znaků. Menší procentuální úspěšnost mají na svědomí různobarevné texty, nestandardní fonty nebo příliš malé oblasti s velkým množstvím textu (novinový článek).

Při implementaci detekčních a lokalizačních algoritmů jsem testoval nastavení parametrů programu stále na stejném vzorku dat, abych docílil co nejobecnějšího a nejefektivnějšího výsledku. Ovšem pro každý obrázek vyhovují jiné parametry a docílit přesného a obecného nastavení všech parametrů je nemožné.

Uživateli jsem tak dal možnost se spuštěním programu nastavit *balance*, který v rozmezí od 0 do 10 představuje podíl správně nalezených částí textu oproti špatně označeným oblastem, kde se text nevyskytuje. Při nastavení *balance* na 0 jsou algoritmy zaměřeny na nalezení striktně textových částí s minimálním označením oblastí neobsahujících text. To má za následek i menší procento označených oblastí s textem. Naopak při nastavení *balance* na 10 je v obrázku více objektů považováno za text. Nalezne se tedy více oblastí s textem, ale také oblasti, ve kterých se text nenachází, se mohou vyhodnotit jako oblasti s textem.

Dále bylo provedeno testování na třech videích s titulky. Jednalo se o statický (nepohyblivý) text v různých barvách a velikostech. Výsledky jsou opět v příloze. Celkově se nalezlo 1003 znaků z celkového počtu 1157 znaků. Z nalezených 1003 znaků bylo 191 úspěšně rozpoznáno. Lokalizace textu tedy byla úspěšná ve více jak 93% a extrakce textu pomocí nástroje *Tesseract OCR* byla úspěšná pouze v 17% případů. Dále bylo špatně označeno 107 oblastí, ve kterých se text nevyskytoval. Za zvýšené procento správně lokalizovaných oblastí oproti testování na obrázcích může doplnění lokalizačního algoritmu o metody hledající stabilní oblasti ve videu a také stejnobarevný text titulků.

6.2 Časová náročnost

Implementace některých algoritmů nebo algoritmů vestavěných v knihovně OpenCV pracující s daty v obraze je příliš časově náročná. Jedná se například o algoritmy zkoumající pohyb zajímavých bodů (tzv. *trackery*), již zmíněný algoritmus *cvPyrSegmentation* nebo algoritmus pro nalezení oblasti s maximální stabilitou v závislosti na změně prahu *MSER*. Zvýšenou časovou náročností je myšlena doba v řádu desítek až stovek milisekund. Při zpracování obrázků bychom některé algoritmy mohli využít a uživatel aplikace by zpoždění postřehl jen v malé míře, ale při zpracování videosnímků je použití těchto algoritmů nereálné z důvodu zajištění uživatelské přívětivosti. Takovéto algoritmy jsou v práci implementovány, ale nepoužity. Pro přehlednost jsou doplněny předponou *unused*.

Chceme-li například získat informace o barvě všech pixelů v obraze o velikosti 512x256 pixelů, máme několik možností.

Algoritmus 5.1: Přístup k informacím pixelu pomocí cvGet2D().

```
IplImage *img = cvCreateImage(cvSize(512,256),8,1);
cvZero(img);
cvResize(inputImage,img);
cloct_t startTime = clock() / (CLOCKS_PER_SEC / 1000);
std::cout << startTime << std::endl;
for (int i=0;i < img->width; i++) {
        for (int j=0;j < img->height; j++) CvScalar s = cvGet2D(img,j,i);
}
cloct_t endTime = clock() / (CLOCKS_PER_SEC / 1000);
std::cout << endTime << std::endl;
```

Výstupem této části kódu jsou dvě čísla (740 a 750). Znamená to rozdíl 10 milisekund při jednom průchodu všech pixelů v obraze. Představme si, že máme videosnímek, který zobrazuje 25 snímků za vteřinu. To je 40 milisekund na jeden snímek. Při použití výše uvedeného algoritmu 4krát bychom nemohli videosnímek zobrazit uživateli plynule. Je třeba použít jiný způsob získávání informací o pixelech.

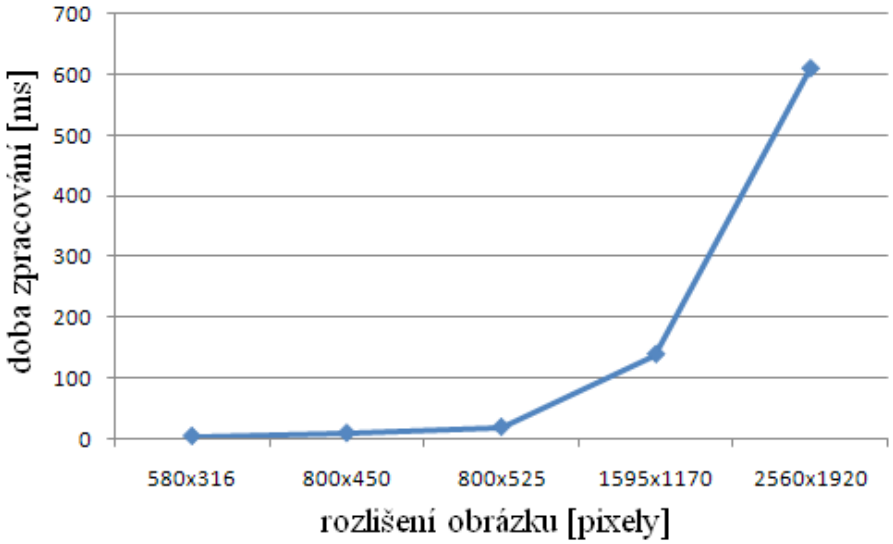
Přímým přístupem do struktury *IplImage* docílíme rychlého zjištění potřebných informací (v tomto případě informace o barevných složkách *b,g,r*) s využitím parametrů *char *imageData, int nChannels a int widthStep* následovně:

```
int b = image->imageData[y*image->widthStep + x*image->nChannels + 0];
int g = image->imageData[y*image->widthStep + x*image->nChannels + 1];
int r = image->imageData[y*image->widthStep + x*image->nChannels + 2];
```

x a *y* značí aktuální pozice na osách *x* a *y*. Nahrazením funkce *cvGet2D* přímým přístupem do struktury *IplImage* dostaneme na standardním výstupu po použití algoritmu 5.1 dvě stejné hodnoty, tedy průběh zpracování algoritmu pod 1 milisekundu.

Časová složitost použitých algoritmů v práci je kvadratická ($O(n) = n^2$). Graf 5.1 poukazuje na závislost celkové doby zpracování obrázku při průchodu systémem TIE (detekce, lokalizace, extrakce a rozpoznání) na velikosti vstupního obrázku.

Graf 5.1: Závislost doby zpracování na velikosti vstupního obrázku.



6.3 Možná vylepšení

Vzniklá aplikace je použitelná pro účely dané zadáním práce, avšak je možné na ni dále navázat a pokračovat tak v jejím rozšíření.

Možná vylepšení by byla v podobě zrychlení doby zpracování snímku při průchodu systémem TIE. Konrétně by se jednalo o algoritmy, které v práci nejsou použity (MSER, cvPyrSegmentation a cvCalcOpticalFlowPyrLK). Celkový chod aplikace by byl plynulejší a text by se ve videu mohl hledat a rozpoznávat v reálném čase.

Dalším vylepšením by mohlo být umožnění nastavení vstupních parametrů uživatelem (například barva a výška textu) při spuštění aplikace, což by vedlo k přesnějšímu nalezení textu v obrázku. Tím by ale nástroj přestal být plně automatizovaným.

Způsob, jakým lze dosáhnout ještě přesnějších lokalizačních výsledků, je automatické vyhodnocení výsledků nad aktuálními daty. Program by dokázal měnit parametry programu a vyhodnocovat úspěšnost lokalizace textu v závislosti na změnách parametrů.

V případě lokalizace textu ve videu by se v budoucnu mohli doplnit algoritmy hledající posuvný text.

Text v obrázcích nemusí být pouze v horizontální poloze. Doplnění aplikace o metody hledající text v jakémkoliv směru by byly přínosné.

I přes výše uvedené možnosti vylepšení by se aplikace dala ještě rozšířit o vlastní nástroj na extrahování textu. Vznikl by tak samostatný nástroj pro detekci, lokalizaci a rozpoznání textu z obrázků a videí.

7 Závěr

Cílem této práce bylo seznámit se s detekčními, lokalizačními a extrakčními metodami při optickém rozpoznání textu v obrázcích a videích, což bylo popsáno v kapitolách 2 a 3. Dále bylo zapotřebí identifikovat a vyřešit problémy týkající se správného nalezení textu ve snímcích (kapitola 4). Výstupem práce je implementace nástroje pro optické rozpoznávání textu. Bylo provedeno testování nástroje na vhodně zvolených vzorcích dat.

Zhotovená aplikace klade důraz na jednoduchost a uživatelsky přívětivé prostředí. Jedná se o nástroj, který dokáže nalézt a extrahovat text do uživatelem zvolené podoby. Uživatel má možnost získat a vyhodnotit výsledky a taky je ovlivnit změnou parametrů některých použitých algoritmů. Pro uživatelskou přívětivost umožňuje aplikace vícenásobné otevírání souborů a jejich vzájemné prohlížení pomocí ovládacího panelu aplikace.

Z historického hlediska se prvními nástroji na rozpoznávání znaků zabývali vědci již počátkem 30. let minulého století. S ohledem na současnost se nástroje na rozpoznání znaků z digitální podoby mnohonásobně zlepšily, avšak stále nedosahují 100% výsledků.

Výsledky při testování se odvíjely od zdokonalování lokalizační části implementace a jsou shrnuty v kapitole 6. S postupně přibývajícími testovacími daty jsem narážel na stále nově vznikající problémy a nutnost reimplementovat lokalizační algoritmy. Mým cílem bylo vyvinout plně automatizovaný nástroj, který by bez nutnosti znalosti barvy, pozice nebo velikosti textu dokázal lokalizovat textovou podobu písma v jakémkoliv obrázku nebo videu a zároveň vyloučit objekty netextového charakteru. Důsledkem toho byly použity adekvátní průměrné hodnoty pro vyhledávací algoritmy získané při testování nad vstupními daty. Nástroj tak dokázal lokalizovat více jak 83% znaků z testovací sady obrázků správně a z nich bylo více jak 58% přesně rozpoznáno pomocí nástroje *Tesseract OCR*. Při testování nástroje na videu se statickým textem (titulky) byly použity algoritmy, které nepřesahují časovou náročnost pro plynulé přehrávání videa.

Pro nekomerční účely má aplikace své využití při hledání a extrakci textu z obrázků a videa. Ve spolupráci s panem vedoucím Ing. Petrem Chmelařem jsem pomocí přepínače při spuštění programu docílil výpisem na standardní výstup lokalizaci textových oblastí v aktuálním snímku.

Pokud by se práce dále rozvíjela tak, jak jsem popsal v kapitole o možných rozšířeních, zvýšila by se procentuální úspěšnost lokalizace i extrahování textu z obrázků a videí.

Všechny body zadání se podařilo splnit.

Literatura

[1] SONKA, Milan; HLAVAC, Vaclav; BOYLE, Roger. *Image Processing, Analysis, and Machine Vision. Third Edition*. United States of America : THOMSON, 2008. 829 s. ISBN 0-495-08252-X.

[2] BRADSKI, Gary R; KAEHLER, Adrian. *Learning OpenCV*. Sebastopol : O'Reilly, c2008. 555 s. ISBN 978-059-6516-130

[3] STRAKA, Stanislav. *Segmentace obrazu* [online]. Brno, 2009. 57 s. Diplomová práce. Fakulta informatiky Masarykovy univerzity. Dostupné z WWW: <http://is.muni.cz/th/72784/fi_m_a2/dp.pdf>.

[4] JUNG, Keechul; IN KIM, Kwang; K. JAIN, Anil. *Text Information Extraction in Images and Video : A Survey*.CiteSeerX [online]. 2004, 37, [cit. 2011-05-13]. Dostupný z WWW: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.1103&rep=rep1&type=pdf>.

[5] TUYTELAARS, Tinne; MIKOLAJCZYK, Krystian. Local Invariant Feature Detectors: A Survey. *Foundations and Trends in Computer Graphics and Vision* [online]. 2007, 3, 3, [cit. 2011-07-22]. Dostupný z WWW: <http://homes.esat.kuleuven.be/~tuytelaa/FT_survey_interestpoints08.pdf>.

[6] POKORNÝ, Pavel. *Optické rozpoznávání znaků* [online]. Brno, 2010. 28 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=8081>.

[7] BĚHAL, Lukáš. *Tutoriál práce s OpenCV* [online]. Brno, 2010. 58 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z WWW: <www.fit.vutbr.cz/study/DP/rpfile.php?id=5132>.

[8] RETORNAZ, Thomas; MARCOTEGUI, Beatriz. *Scene text localization based on the ultimate opening* .International Symposium on Mathematical Morphology [online]. 2008, 1, [cit. 2011-07-22]. Dostupný z WWW: <cmm.ensmp.fr/~marcoteg/retornaz/ismm07_thomas.pdf>.

[9] *OpenCV 2.0 C Reference* [online]. 2009 [cit. 2011-05-15]. OpenCV 2.0 C Reference. Dostupné z WWW: <http://opencv.willowgarage.com/documentation/index.html>.

[10] LI, Huiping; DOERMANN, David; KIA, Omid. *Automatic Text Detection and Tracking in Digital Video*. LAMP-TR-028 [online]. 1998, [cit. 2011-05-16]. Dostupný z WWW: <http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=A1E2CBB59EF0F2AF508776654D988770?doi=10.1.1.56.2202&rep=rep1&type=pdf>.

[11] R. LYU, Michael; SONG, Jiqiang; CAI, Min. *A Comprehensive Method for Multilingual Video Text Detection, Localization, and Extraction*. IEEE Transactions on curcuits and systems for video technology [online]. 2005, 15, 2, [cit. 2011-05-16]. Dostupný z WWW: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1390999>.

[12] CAI, Min. *A new approach for video text detection* [online]. Hong Kong : SAR, 2002. 4 s. Oborová práce. The Chinese University of Hong Kong, Department of Computer Science & Engineering. Dostupné z WWW: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?Arnumber=1037973>.

[13] *Tesseract-ocr – Project Hosting on Google Code*. [online]. 2009 [cit. 2011-05-17]. Tesseract-ocr. Dostupné z WWW: <http://code.google.com/p/tesseract-ocr/>.

Seznam příloh

Příloha 1. Výsledky testování obrázků

Číslo obrázku	Nalezeno písmen	Špatně nalezených oblastí	Rozpoznáno písmen
1	13/15	0	7/13
2	26/26	0	20/26
3	4/16	0	0/4
4	15/15	1	15/15
5	22/23	2	12/22
6	24/24	0	24/24
7	0/14	0	0/0
8	0/8	0	0/0
9	12/19	3	5/12
10	34/34	0	27/34
11	39/39	0	39/39
12	27/27	1	14/27
13	19/25	2	7/19
14	17/29	0	12/17
15	28/28	0	28/28
16	38/46	3	29/38
17	19/20	0	18/19
18	21/21	1	21/21
19	27/32	0	17/27
20	44/44	0	0/44
21	30/30	1	23/30
22	78/78	33	46/78
23	0/9	2	0/0
24	24/25	0	22/24
25	13/36	0	9/13
26	7/16	2	7/7
27	18/29	1	15/15
28	27/33	0	25/27
29	57/71	44	33/57
30	19/19	0	19/19
31	11/27	55	9/11
32	30/30	1	30/30
33	0/14	1	0/0
34	23/23	0	21/23
35	32/32	0	32/32
36	36/36	0	0/36
37	27/27	0	22/27
38	0/5	0	0/0
39	15/15	0	15/15
40	71/71	2	0/71
41	34/34	1	19/34
42	15/25	0	13/15
43	14/14	1	14/14
44	0/0	1	0/0
45	0/0	00	0/0
46	24/30	10	18/24
47	22/22	1	22/22
48	47/53	1	28/47
49	37/37	0	32/37

50	0/28	0	0/0
51	26/26	1	24/26
52	33/33	0	8/33
53	16/16	2	16/16
54	45/45	3	36/45
55	42/67	1	26/42
56	15/21	2	13/15
57	40/51	1	40/40
58	39/39	0	30/39
59	0/12	0	0/0
60	14/14	0	11/14
61	7/7	0	7/7
62	7/7	0	7/7
63	7/7	1	6/7
64	0/7	0	0/0
65	7/7	0	7/7
66	7/7	0	3/7
67	7/7	0	7/7
68	7/7	0	7/7
69	3/7	1	3/3
70	7/7	0	7/7
71	47/71	4	32/47
72	27/27	1	24/27
73	36/40	0	29/36
74	25/25	0	25/25
75	48/55	1	33/48
76	32/32	0	12/32
77	0/0	0	0/0
78	20/21	0	13/20
79	17/17	0	17/17
80	22/22	0	4/22
81	28/29	2	25/28
82	14/14	0	14/14
83	0/18	0	0/0
84	19/30	4	17/19
85	23/23	0	23/23
86	27/27	2	16/27
87	11/36	0	9/11
88	0/0	0	0/0
89	26/33	1	20/26
90	21/21	0	20/21
91	37/37	1	18/37
92	22/22	0	15/22
93	29/63	0	0/29
94	43/54	2	17/43
95	71/80	1	0/71
96	69/69	0	5/69
97	70/76	3	13/70
98	73/73	2	0/73
99	59/84	2	0/59
100	57/65	1	11/57
1-100	**2442/2932**	**79**	**1439/2442**

Příloha 2. Výsledky testování videí

Číslo obrázku	Nalezeno písmen	Špatně nalezených oblastí	Rozpoznáno písmen
1	531/562	63	97/531
2	289/318	24	50/289
3	263/277	20	44/263
1-3	**1083/1157**	**107**	**191/1083**