VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MODUL PRO ROZPOZNÁVÁNÍ NÁPISŮ PRO ROBOTA

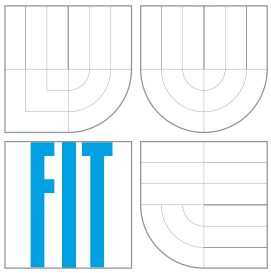
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK HARTMAN

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MODUL PRO ROZPOZNÁVÁNÍ NÁPISŮ PRO ROBOTA

AUTOMATIC TEXT RECOGNITION FOR ROBOTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK HARTMAN

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠPANĚL, Ph.D.

BRNO 2015

Abstrakt

Tato bakalářská práce popisuje návrh modulu pro detekci a rozpoznávání nápisů pro použití v robotických systémech. K detekci znaků je použita Stroke Width transformace, která je aplikována na vstupní hranový obraz. Ve výstupním obraze po Stroke Width transfromaci jsou nalezeny spojité oblasti. K seskupení znaků do slov a detekci orientace slov je použita Houghova transformace aplikována na vytvořený binární obraz, který obsahuje body odpovídající pozici nalezených spojitých oblastí. K rozpoznání nápisů v detekovaných oblastech je použita knihovna Tesseract. Před provedení rozpoznávání jsou detekované oblasti extrahovány a natočeny do vodorovné polohy. Takto navržený detektor dokáže detekovat i natočený text. Podařilo se dosáhnout úspěšnosti detekce 75% nad testovací sadou „informační tabule“.

Abstract

This bachelor thesis describe module design for text detection and recognition for use in robotic systems. To detect charters is used Stroke Width transform, which is applied on the input edge image. In the output image after Stroke Width trasnform are found connected components. For letter gruping into a words is used Hough transform, which is aplied on the created binary image. This image contains points, which corresponding positions of found connected components. To recognize signs in detected areas is used Tesseract library. Before recognition detected areas are extracted and rotated into a horisontal position. This proposed detector can detect even rotated text. Accuracy of detection of the text is 75% above the test set „informační tabule“.

Klíčová slova

Detekce textu, rozpoznávání textu, Stroke Width transformace, SWT, Canny Edge detector, Houghova transformace, spojité oblasti, hledání spojitých oblastí, semínkové vyplňování, OCR, Tesseract.

Keywords

Text detection, text recognition, Stroke Width transform, SWT, Canny Edge detector, Hough transform, connected components, CCs, Flood Fill, OCR, Tesseract.

Citace

Zdeněk Hartman: Modul pro rozpoznávání nápisů pro robota, bakalářská práce, Brno, FIT VUT v Brně, 2015

Modul pro rozpoznávání nápisů pro robota

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla, Ph.D.

.....
Zdeněk Hartman
18. května 2015

Poděkování

Za odbornou pomoc děkuji vedoucímu práce Ing. Michalu Španělovi, Ph.D.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Vývoj v robotice jde strmě kupředu. To, co bylo dříve pouze snem, je dnes skutečností. Například kvadroptéry, které před několika lety byly pouze výsadou větších technických univerzit, jsou dnes běžně k zakoupení za pár korun. Rapidní nástup robotů a především jejich dostupnost nabízí prostor pro vznik aplikací, které jim umožní autonomnější rozhodování.

Cílem práce bylo vytvořit modul pro detekci a rozpoznávání nápisů pro robota. Vstupem modulu je obraz z videokamery nebo z jejího záznamu, pro testování je možné použít i jednotlivé snímky. Výstupem modulu jsou informace užitečné pro navigaci robota a tvorbu modelu okolního prostředí. V této práci je uveden vlastní princip využití Houghovy transformace pro detekci slov a určení jejich natočení. Součástí práce je také vlastní implementace Stroke Width transformace a upravený algoritmus hledání spojitých oblastí s využitím Flood Fill algoritmu.

Při vytváření modulu byla použita metoda Cannyho hranového detektoru pro detekci hran, následovaná Stroke Width transformací, která transformuje hranový obraz na obraz, kde hodnoty pixelů odpovídají šířce tahu (pruhu), ve kterém se nachází. Po SWT jsou v SWT mapě nalezeny spojité oblasti. Nakonec jsou pomocí Houghovy transformace nalezeny řádky textu. Tyto metody byly použity pro detekci textu. Pro rozpoznávání byla použita open source knihovna Tesseract.

Principy všech použitých metod pro detekci textu jsou uvedeny v teoretické části – viz následující kapitola 2. K rozpoznávání používám knihovnu (API) Tesseract, jeho základní popis je uveden v kapitole 4. Kapitola 3 shrnuje současný stav, jsou zde uvedeny různé přístupy pro detekci textu. Vlastní návrh řešení je uveden v kapitole 5. Moje řešení částečně vychází z práce – viz [7], konkrétně se jedná o použití Cannahy hranového detektoru a následné použití Stroke Width transformace, jejíž princip byl zde popsán. Zajímavější části implementace naleznete v kapitole 6. Jsou zde uvedeny poznámky k mé implementaci SWT, dále pak způsob předzpracování vstupního obrazu pro Houghovu transformaci. Poslední částí této kapitoly je popis implementace algoritmu pro hledání spojitých oblastí s využitím FloodFill algoritmu. Dosažené výsledky tohoto řešení jsou prezentovány jednak v závěru (viz kapitola 8) této práce. Blíže (včetně definice testovací sady) jsou pak uvedeny v kapitole 7.

Kapitola 2

Metody předzpracování obrazu pro rozpoznávání textu

V této kapitole budou popsány všechny metody a pricipy použité zejména pro detekci textu. Nejdříve se dostaneme k popisu principu **detekce hran**, přes **Stroke Width transformaci** až k **Houghově transformaci**. V jednotlivých kapitolách se snažím popsat jen to podstatné pro pochopení mojí práce.

2.1 Detekce hran

Důležité vlastnosti obrazu jsou získávány právě z hran obrazu. Hrany jsou způsobeny změnou intenzity jasu v jisté oblasti obrazu, neboli pokud $f(x, y)$ je obrazová funkce, vracející hodnotu jasu pro pixel na pozici $[x, y]$, tak hrana se nachází tam, kde **první derivace** nabývá hodnoty různé od nuly 2.1. Respektive, tam kde **druhá derivace**, nabývá nulové hodnoty – viz rovnice 2.1.

$$f'(x, y) \neq 0 \qquad f''(x, y) = 0, \tag{2.1}$$

kde $f(x, y)$ je obrazová funkce. Hodnota **první derivace** určuje intenzitu hrany.

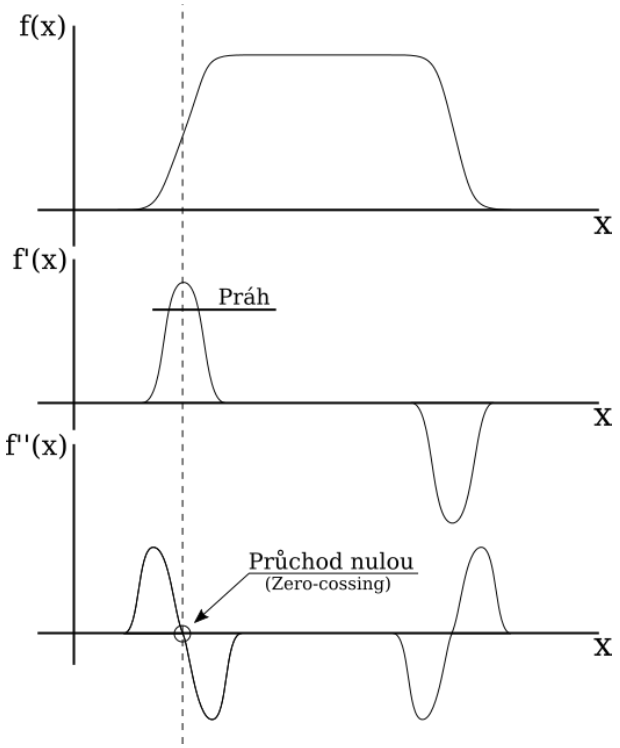
Metody pro detekci hran lze podle [1] rozdělit do dvou skupin. A to na **search based** a **zero-cossing based**. První skupina detektorů využívá pro nalezení hrany **gradient**, který je počítán pomocí **první** derivace, proto jsou tyto metody někdy nazývané jako metody založené na **první** derivaci. Druhá skupina metod (**zero-cossing based**) využívá průchodu druhé derivace nulou (zero-cossing). Pro aproximaci druhé derivace se využívá **laplacián**. Princip odezvy hrany (na první, resp. druhou derivaci) je patrný z obrázku 2.1. Taktéž je z něj vidět důvod rozdělení detektorů právě do dvou výše zmíněných skupin.

Dále bude v následující kapitole uveden princip metod založených na první derivaci (Search-based detektorů). Nakonec bude uveden Cannyho hranový detektor, který využívá právě první derivaci a je použit v této práci.

2.1.1 Search-based hranové detektory

Tyto detektory využívají pro detekci hran **gradient** [1]. Gradient funkce je definován jako vektor jejích parciálních derivací. V rovnici 2.2 je uveden gradient dvourozměrné funkce.

[1] http://cs.wikipedia.org/wiki/Gradient_%28matematika%29



Obrázek 2.1: Vztah mezi hranou, první a druhou derivací.

$$\bigtriangledown f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \tag{2.2}$$

Velikost gradientu se určí následovně:

$$|\bigtriangledown f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{2.3}$$

A směr gradientu:

$$\psi = \frac{\left(\frac{\partial f}{\partial x}\right)}{\left(\frac{\partial f}{\partial y}\right)} \tag{2.4}$$

Jelikož obrazová funkce je disktrétní, a tedy výpočet parciálních derivací není možný, aproximují se derivace pomocí konečných diferencí (viz snímky z přednášky [8]). Příklad jedné možné aproximace parciální derivace je uveden v rovnici 2.5. Jedná se o nesymetrickou aproximaci parciální derivace funkce v celočíselném bodě $[x, y]$.

$$\frac{\partial f}{\partial x} \approx f[x+1, y] - f[x, y], \tag{2.5}$$

kde $f(x, y)$ je obrazová funkce a $f[x, y]$ je diskrétní obrazová funkce.

Druhou více používanou aproximací parciálních derivací je konvoluce vstupního obrazu a operátoru s vhodným **jádrem**. Mezi nejpoužívanější kovoluční jádra patří **Sobelovy**

operátory 2.6 – viz [14].

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{2.6}$$

Potom aproximace parciálních derivací pomocí konvoluce je provedena následovně:

$$\frac{\partial f}{\partial x} \approx S_x * f[x,y] \qquad \frac{\partial f}{\partial y} \approx S_y * f[x,y], \tag{2.7}$$

kde S_x a S_y jsou konvoluční jádra a $f[x,y]$ je diskrétní obrazová funkce.

Velikost gradientu 2.3, představuje intenzitu hrany. Směr gradientu 2.4 definuje směr hrany. Vektor směru hrany je kolmý na směr gradientu (nebo-li tvoří jeho normál).

Detektory založené na první derivaci nejdříve určí velikost gradientu pro jednotlivé pixely. Tyto hodnoty uloží do matice. V této matici je provedeno prahování, tím jsou ponechány pouze ty hranové pixely, jejichž intenzita je větší než předem definovaný práh. Důsledkem prahování dostaneme **binární** obraz, kde hodnota pixelu rovna jedné znamená, že v daném místě (na souřadnicích $[x,y]$) se v původním obraze nachází **hranový** pixel. Tento postup je u všech **search-based** detektorů stejný, liší se až následující úprava hranového obrazu (např. Cannyho hranový detektor, viz 2.1.2).

2.1.2 Cannyho hanový detektor

Jedná se o detektor, který pracuje s první derivací obrazové (jasové) funkce. Tedy spadá do search based-detektorů. Výhody tohoto algoritmu jsou:

1. Minimální počet chyb
2. Jednoznačnost
3. Přesnost

Bod 1 lze jednoduše chápat tak, že musí být detekovány všechny hrany a nesmí být detekována místa, která nejsou hranami. Jednoznačnost (bod 2), neboli pouze jedna odezva na hranu – tedy jedna hrana nesmí být detekována dvakrát. Přesnos (bod 3) této metody spočívá v nalezení přesné polohy, tedy skutečného výskytu hrany. Ačkoli se tyto body mohou zdát triviální, jejich docílení není zas tak snadné.

Algoritmus toho detektoru je uveden zde 2.1.1.

Algoritmus 2.1.1. *Cannyho hranový detektor*
Vstup: *původní obraz*
Výstup: *binární hranový obraz*

1. *Proveď rozmazání vstupního obrazu* ***Gaussovým*** *filtrem.*
2. *Vypočti parciální derivace* $\frac{\partial f}{\partial x}$ *a* $\frac{\partial f}{\partial y}$*. Derivace aproximuj například konvolucí vstupního obrazu a Sobelových filtrů – viz kovoluční jádra 2.7.*
3. *Pro každý pixel spočítej velikost a směr gradientu. Velikost gradientu* $|\bigtriangledown f(x,y)|$ *urči podle vztahu 2.3. Směr gradientu* $\psi(x,y)$ *urči podle vztahu 2.4*
4. *V matici* $|\bigtriangledown f(x,y)|$ *proveď* ***ztenčení hran*** *(line thinning) na základě směru gradientů* $\psi(x,y)$*. Jeden z možných algoritmů je popsán v kapitole 2.1.3.*
5. *Proveď* ***hysteresní prahování*** $|\bigtriangledown f(x,y)|$*. Jeho princip je uveden v kapitole 2.1.4.*

2.1.3 Non-maxima suppression

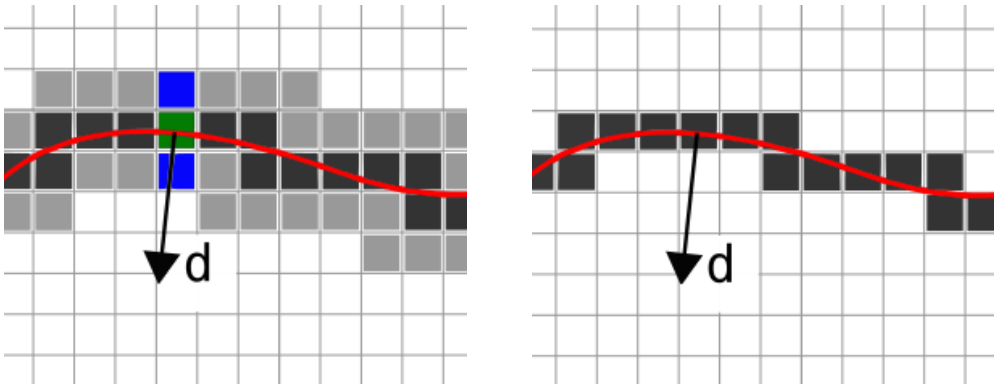
Hrany objektů ve vstupním snímku nejsou ostré (navíc je provedeno rozmazání **Gasouým filtrem** za účelem redukce šumu), tudíž po výpočtu velikostí gradientů dostaneme tlusté hrany – viz obrázek 2.2a. Pro ztenčení hran lze použít algoritmus **Non-maxima supression** (zkr. NMS) – viz algoritmus 2.1.2, česky algortimus pro „potlačení nemaximálních" hran. Popis tohoto algoritmu je parafrázován ze snímků k přednášce – viz [6].

Jedná se o poloprahovací metodu vycházející z toho, že hrana v místě svého skutečného výskytu dává **největší odezvu** – viz 2.2a. Úkolem této metody je, jak název napovídá, odebrat body, které nejsou **lokálním** maximem. Po provedení této metody dostaneme binární hranový obraz, kde šířka hrany je právě jeden pixel - viz 2.2b.

Algoritmus 2.1.2. *Non-maxima suppresion*
Vstup: *původní obraz; hranový obraz (matice velikostí gradientů)*
Výstup: *hranový obraz se ztenčenými hranami*

1. *Pro každý pixel původního obrazu urči podle 2.4 směr gradientu ψ.*
2. *Vyber souřadnice pixelu $X > 0$ z matice velikostí gradientů, pro který urči jeho sousední pixel ve směru a proti směru jeho gradientu ψ. Souřadnice těchto pixelů označ Y, Z.*
3. *Pokud $|\bigtriangledown f(Y)| > |\bigtriangledown f(X)|$ nebo $|\bigtriangledown f(Z)| > |\bigtriangledown f(X)|$, nastav $|\bigtriangledown f(X)| = 0$, neboli smaž hranový pixel.*
4. *Pokud nejsou zpracované všechny pixely, pokračuj bodem 2.*

Ilustrace algoritmu 2.1.2 je na obrázku 2.2. Na obrázku vlevo je znázorněn vstup, jedná se o matici velikostí gradientů ($|\bigtriangledown f(x, y)|$), kde intenzita pixelu představuje velikost gradientu. Zde můžeme vidět, že hrana dává největší odezvu (velikost gradientu) v místě, kde se skutečně nachází (červená křivka). Dále je zde zeleně zvýrazněn vybraný pixel, modře jsou pak jeho sousedi ve a proti směru jeho gradientu ψ. Jeden z těchto pixelů bude v dalším kroku odstraněn, druhý bude odstraněn později. Výsledná matice gradientů po aplikování **NMS** je na obrázku 2.2a.



(a) Před NMS. (b) Po NMS.

Obrázek 2.2: Princip Non-maxima suppresion.

2.1.4 Hysterezní prahování

Abychom získali z matice velikosti gradientů pouze výrazné hrany, musíme provést prahování. Cannyho hranový detektor používá **hysterezní** prahování. Na rozdíl od klasického prahování používá hysterezní prahování dva prahy, *low_tresh* a *high_tresh*. Algoritmus hysteresního prahování je uveden zde 2.1.3.

Algoritmus 2.1.3. *Hysterezní prahování*
Vstup: *hranový obraz (matice velikostí gradientů); dvojice prahů – low_thresh a high_thresh*
Výstup: *binární hranový obraz*

1. *Vhodně zvol velikosti prahů low_tresh a high_tresh.*
2. *Každý pixel (v matici velikostí gradientů) s hodnotou **větší** než high_tresh nazvi **silným**.*
3. *Každý pixel (v matici velikostí gradientů) s hodnotou **menší** než hight_tresh **a větší** než low_tresh nazvi **slabým**.*
4. *Pixely s hodnotou **menší** než low_tresh odstraň.*
5. *Každý slabý pixel sousedící se silným, nazvi také silným.*
6. *Pokud se změnil počet slabých pixelů, jdi na krok 5, jinak pokračuj.*
7. *Všechny slabé pixely odstraň.*

Jinými slovy - do výsledku zahrneme jen ty pixely, jejichž hodnota gradientu je větší než předem definovaný horní práh *higt_tresh*. Dále zahrneme ty pixely, jejichž hodnota je mezi *low_tresh* a *high_tresh*, a zároveň tvoří **spojitou** oblast s některým ze silných pixelů. Velikost prahů se volí většinou experimentálně, navíc pro různé obrázky mohou nabývat různých hodnot.

Po provedení hysterezního prahování dostáváme výsledný hranový obraz, jehož obrazová funkce nabývá pouze hodnot 0 nebo 1 (binární funkce).

Alternativní popis algoritmu je uveden v přednášce [6]. Efektivní implementace tohoto algoritmu je pak uvedena v [11].

2.2 Stroke Width transformace – SWT

Stroke je taková část obrazu, která tvoří **tah** (pruh) s konstantní šířkou. **Stroke width** jednotlivých pixelů je rovna sířce **Stroke** oblasti, která tento pixel obklopuje. Následující popis alogritmu vychází z [7]. Vstupem je původní a hranový obraz a výstupem je matice obsahující hodnoty **Stroke Width**, tato matice má stejnou velikost jako vstupní obraz a můžeme ji nazvat například **SWT mapa**.

Algoritmus 2.2.1. *Stroke Width transformace*
Vstup: Původní a hranový obraz; velikost jádra mediánového filtru.
Výstup: SWT mapa (obraz).

1. *Nastav hodnoty SWT mapy na ∞.*
2. *Detekuj hrany ve vstupním obraze - viz kapitola 2.1.2.*

3. *Pro každý* ***hranový*** *pixel p z vstupního hranového obrazu.*

   (a) *Vypočti směr gradientu ψ_p podle rovnice 2.4.*

   (b) *Pohybuj se ve směru paprsku $r = p + n \times \psi_p$, pro $n > 0$, dokud nenarazíš na* ***hranový*** *pixel g.*

   (c) *Pokud směr gradientu pixelu p je přibližně opačný než gradient pixelu q, neboli pokud platí:*

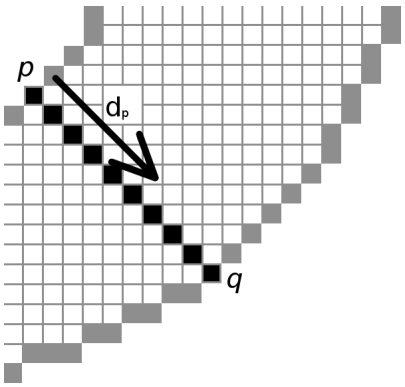
$$\psi_p = -\psi_q \pm \frac{\pi}{6} \tag{2.8}$$

   pokračuj krokem 3d, jinak pokračuj krokem 3.

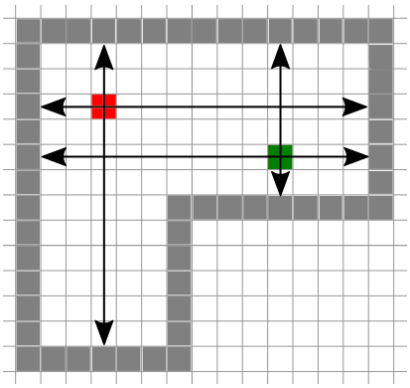
   (d) *Všem pixelům ležícím na paprsku mezi pixely p a q nastav hodnotu (**Stroke Width**) odpovídající jejich vzdálenosti, neboli $|\overrightarrow{p-q}|$. Toto se provede tak, že hodnota jednotlivým pixelů na paprsku se nastaví na minimum z jejich původní hodnoty a hodnoty nově spočítané.*

4. *Na SWT mapu aplikuj* ***mediánový*** *filtr* [2].

Princip výpočtu SWT je ukázán na obrázku 2.3a. Černé pixely představují pixely ležící na paprsku. Počet těchto pixelů odpovídá **Stroke Width** hodnotě, která je těmto pixelům nastavena. Směr **gradientu** pixelu p je na obrázku naznačen jako d_p.



(a) Princip SWT – pozn.: d_p odpovídá ψ_d.



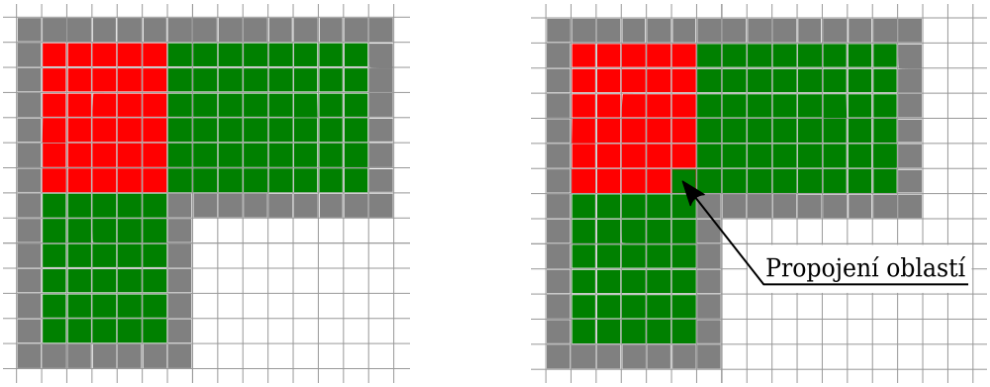
(b) Způsob určení SW, zelený pixel – správně, červený pixel – špatně (viz text).

Obrázek 2.3: Princip SWT.

Nedostatkem tohoto algoritmu je nekorektní výpočet SW u specifických geometrických tvarů. Na obrázcích 2.3b a 2.4 je vidět chybná hodnota **Stroke Width**, je znázorněna červeným pixelem. I kdybychom vybrali nejnižší z obou hodnot, tak je zjevné, že výsledek nebude korektní. Toto nastává například u znaků: A, H, L.

Částečným řešením tohoto problému je aplikace zmíněného **mediánového** filtru. Protože minimálně jeden pixel spojí tyto dvě oblasti (viz 2.4b), neboli tyto dvě oblasti vytvoří jednu spojitou oblast.

[2] http://en.wikipedia.org/wiki/Median_filter



(a) Před aplikací MF. (b) Po aplikaci MF.

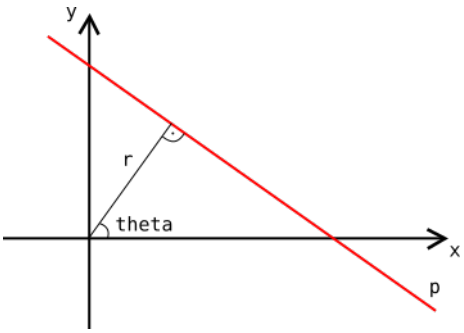
Obrázek 2.4: Důvod aplikace mediánového filtru.

2.3 Houghova transformace

Houghova transformace je metoda pro nalezení **parametrického** vyjádření objektů nebo křivek (přímek, kružnic, elips, atd.) v obraze. Proto takto hledané objekty (křivky) musí být parametricky popsatelné. V této kapitole budeme dále uvažovat pouze přímky. Původní verze, patentovaná v roce 1962, pracovala se **směrnicovým** [3] vyjádřením přímek. Tento způsob popisu však neumožňoval detekci svislých přímek (parametr k by se musel limitně blížit nekonečnu). Proto později Duda a Hart přišli s možností tohoto 2.9 popisu přímky:

$$x\cos(\theta) + y\sin(\theta) = \rho \tag{2.9}$$

jedná se o **normálový** popis přímky [4] - viz [5], kde ρ je délka normály a θ úhel mezi normálou a osou x, což je patrné z obrázku 2.5. Na rozdíl od směrnicového popisu umožnujě tento způsob popsat přímky s libovolným úhlem natočení.



Obrázek 2.5: Popis přímky v normálovém tvaru.

Princip HT spočívá v transformaci bodů obrazového prostoru (x, y) do parametrického prostoru (θ, ρ) podle rovnice 2.9. Každý bod v obrazovém prostoru - viz obrázek 2.6a, tvoří křivku v prostoru parametrickém - viz obrázek 2.6b. Toto plyne z dosazení do rovnice 2.9 souřadnic libovolného bodu $p[x, y]$. Pokud takto transformujeme všechny body vstupního obrazu, dostaneme v parametrickém prostoru stejný počet křivek, přičemž místa jejich

[3] s rovnicí přímky ve tvaru: $x = kx + q$

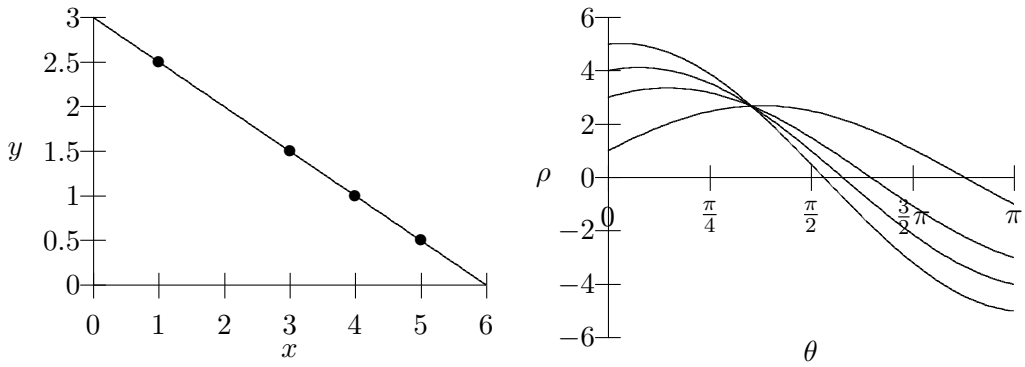
[4] Často nazývané – ró-theta parametrizace.

průsečíků nám určují parametry (θ, ρ) hledaných přímek. Počet průsečíků v jednom místě určuje počet pixelů v detekované přímce. Toto umožnujně definovat jakýsi práh, který nám umožní detekovat pouze přímky s daným počtem bodů. Pro implementaci parametrického prostoru se používá dvourozměrné pole (matice), nazývaná **akumulátor**. Akumulátor se skládá z buněk.

Algoritmus vycházející z [**?**] je uveden v následujícím popisu algoritmu 2.3.1.

Algoritmus 2.3.1. *Houghova transformace*
Vstup: *binární obraz; práh T*
Výstup: *množina přímek (resp. jejich parametrů)*

1. *Vytvoř akumulátor.*
2. *Vynuluj položky akumulátoru.*
3. *Pro každý bod $p[x, y] = 1$ vstupního binárního obrazu:*
   (a) *Souřadnice $[x, y]$ bodu p vyjádři v parametrickém prostoru, podle rovnice 2.9. Tímto dostaneš rovnici křivky.*
   (b) *Na všech souřadnicích (θ, ρ) v akumulátoru, kudy křivka prochází, inkrementuj hodnotu.*
4. *Proveď prahování v akumulátoru prahem T.*
5. *Parametry přímek (θ, ρ) odpovídají souřadnicím nenulových buněk akumulátoru.*



(a) Obrazový prostor (x, y). (b) Parametrický prostor (θ, ρ).

Obrázek 2.6: Princip Houghovy transformace.

Implementace této metody je obsažena v knihovně OpenCV [10].

2.3.1 Pravděpodobnostní Houghova tramsformace

Na rozdíl od klasické HT hledá tato vylepšená metoda **úsečky**. Její princip spočívá v postupném **náhodném** vybírání bodů ze vstupního binárního obrazu a jejich následné transformaci do parametrického prostoru (tyto body jsou odstraněny ze vstupního obrazu). Další zásadní rozdíl oproti klasické HT spočívá v tom, že pokud dosáhne hodnota libovolné buňky akumulátoru hodnoty větší jak předem zvoleného prahu, tak tato metoda prochází vstupní

obraz podél této přímky a hledá úsečky. Přimčemž je možné u této metody zvolit maximální velikost mezery mezi pixely.

Do výsledku se poté zahrnou pouze ty úsečky, které jsou delší než předem zvolená minimální velikost. Poté jsou všechny body ležící na této přímce odstraněny ze vstupního obrazu. Takto se pokračuje, dokud nejsou odstraněny všechny pixely ze vstupního obrazu. Tato metoda není tak výpočetně a paměťově náročná jako klasická HT.

Její implementace je opět součástí OpenCV [10].

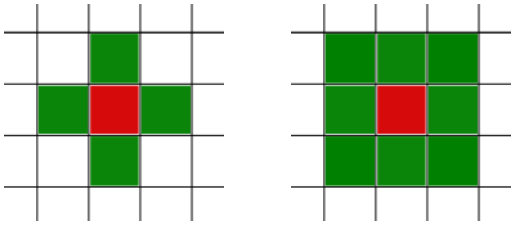
2.4 Flood Fill - semínkové vyplňování

Jedná se o algoritmus pro vyplňování oblastí **rastrového** obrazu. Využívá se sousednosti bodů stejné (podobné) barvy, které jednak mohou tvořit hranici nebo spojitou oblast.

Na počátku je vybráno semínko. Následně jsou obarveny všechny sousední pixely, které mají podobnou barvu. Nově obarvené pixely se stávají semínky. Obarvování končí jakmile nejsou žádná semínka k obarvení.

Podle způsobu obarvování rozlišujeme čtyři různé varianty této metody. Hraniční, záplavové, měkké a prahové – viz [9].

Výběr sousedních pixelů může být realizován jako 4-okolí resp. 8-okolí. Rozdíl je patrný z obrázku 2.7, kde červeně je naznačen aktuální pixel a zeleně jeho sousedi.



Obrázek 2.7: Ukázka 4-okolí a 8-okolí červeného pixelu.

Algoritmus semínkové vyplňování je uveden 2.4.1 a je převzat z [9].

Algoritmus 2.4.1. *Postup vyplňování rastrové Σ oblasti semínkovým algoritmem s vlastním zásobníkem pro semínko $S[x, y]$ je následující:*

1. *Otestuj sousední pixely aktuálního semínka, zde jsou ještě neobarvené a patří do dané oblasti Σ.*
2. *Obarvi vyhovující sousední pixely a vlož je na zásobník.*
3. *Získej semínko z vrcholu zásobníku a pokračuj bodem 1.*
4. *Skonči, pokud je zásobník prázdný.*

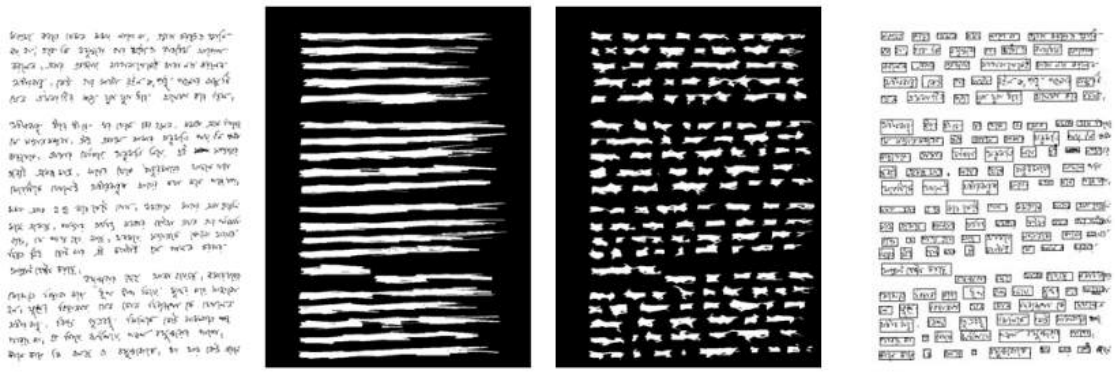
Kapitola 3

Současný stav - State of the Art

V oblasti detekce textu existuje mnoho různých řešení, ale využití SWT a zároveň Houghovy transformace jsem nenašel. V [7] je uvedeno využití Stroke Width transformace a v [13] Houghova transformace pro detekci textu. Dále v [12] je uvedena detekce textu s využitím Edge-based metody. Tyto tři metody budou stučně popsány v této kapitole.

3.1 Využití Houghovy transformace pro segmentaci textu

Autory tohoto [13] řešení jsou Satadal Saha, Subhadip Basu, Mita Nasipuri a Dipak Kr. Basu. Předzpracování obrazu se skládá jako u většiny detektorů textu z převodu obrazu na šedotónový a detekce hran. V binárním obraze je následně provedena pravděpodobnostní Houghova transformace (dále jen HT) s parametry, které umožní detekovat řádky (přesné hodnoty jsou uvedeny v popisované práci). Účelem je nalezení čar představujících řádky textu. Následuje nalezení spojitých oblastí v matici po první HT. K hledání využívají 4-spojitosti oblastí. Poté je v původním obraze v místě jednotlivých spojitých oblastí provedena další HT, s parametry pro nalezení slov. Neboli požadavky na maximální velikost mezery se zmenší, a tím pádem se najde více slov na řádku. V posledním kroku jsou vytvořeny ohraničující rámečky pro jednotlivá slova. Tato metoda detekce textu je zejména vhodná pro detekci slov/řádků, psaného textu, kde má dobré výsledky. Princip metody můžeme názorně vidět na obrázku 3.1.



Obrázek 3.1: Využití segmentace pomocí HT pro detekci textu. Převzato z [13]

Postupně zleva vidíme na obrázcích: text určený pro detekci, první aplikování HT a nalezení spojitých oblastí, druhé aplikování HT a naposledy nalezení ohraničujících rámečků.

3.2 Edge-based matoda pro detekci textu

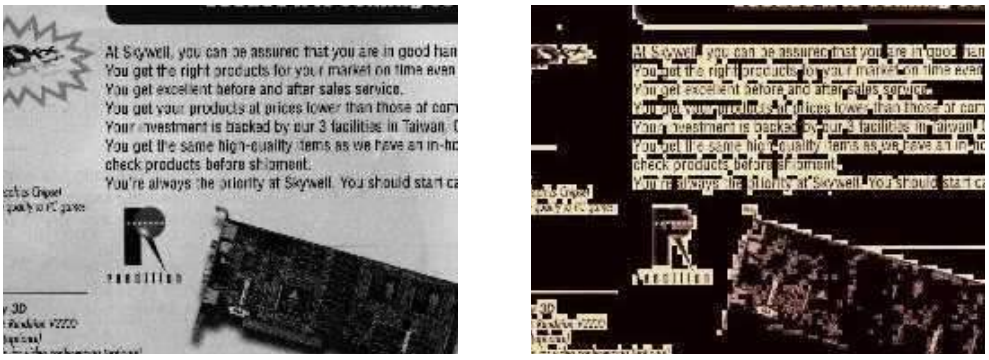
Autory metody jsou Matti Pietikäinen a Oleg Okun. Tato [12] metoda byl navržena pro detekci textu v dokumentech, jako jsou například časopisy. Jejich postup řešení je popsán následujícími kroky:

1. Převod na šedotónový obraz.
2. Detekce hran s použitím Sobelových filtrů, nonmaximum suppresion (potlačení nemaximálních hran) a prahování.
3. Rozdělení hranového obrazu na malé nepřekrývající se oblasti o velikosti $m \times m$. A výpočet vlasností $\mathcal{F}$ pro jednotlivé oblasti. Vlastnosti jsou počítány na základě hran – viz níže.
4. Klasifikace bloků na textové a netextové na základě spočítaných vlastností.

První vlastností jednotlivých bloků je počet hranových pixelů. Druhou vlastností je průměrná velikost gradientu **hranových** pixelů v jednotlivých blocích. A poslední, třetí, vlastností je velikost gradientu pixelů uvnitř bloku. Algoritmus využívá právě toho, že počet hranových pixelů a velikost jejich gradientů je pro textové oblasti větší, než pro oblasti netextové. Autoři definují práh, se kterými je porovnávána daná vlastnost, a pokud platí $\mathcal{F} \geq th$, tak je daný blok označen jako textový. Určení prahu t je však náročné a nelze jej provést automaticky. Proto autoři pouze na základě histogramu vstupního šedotónového obrazu určí minimální $\mathcal{F}_{min}$ a maximální $\mathcal{F}_{max}$ hodnotu vlastnosti $\mathcal{F}$. Jedná se o první, resp. poslední **nenulovou**, hodnotu **histogramu**. Následně podle 3.1 určí interval, ve kterém se spočtená vlastnost musí nacházet, aby se jednalo o text.

$$[\mathcal{F}_{min},\ \mathcal{F}_{min} + k\Delta\mathcal{F}] \tag{3.1}$$

kde $\Delta\mathcal{F} = \mathcal{F}_{max} - \mathcal{F}_{min}$ a $k \in \langle 0.1, 0.3\rangle$. Přesnou hodnotu parametru k zvolili na základě experimentů. Na obrázku 3.2b je uveden výstup jejich metody (vstup je uveden na obrázku 3.2a). Na obrázku jsou vidět vybrané oblasti s textem – bílé oblasti. Hodnota parametru k pro tento výstup je 0.15 a parametr m je 8. Parametr m představuje velikost nepřekrývajících se oblastí (čverců velikosti $m \times m$), pro kterou jsou počítány výše zmíněné vlastnosti $\mathcal{F}$.



(a) Vstupní obraz.

(b) Výstup detektoru.

Obrázek 3.2: Ukázka Edge Based metody pro detekci texu. Části obrázků převzaty z [12].

Tato metoda je, alespoň podle autorů, vhodná pro detekci textu ve složitých (komplexních) dokumentech, jako jsou například různá periodika, brožury, plakáty a tak dále.

3.3 Využití SWT pro detekci textu

Autory této práce jsou Boris Epshtein, Eyal Ofek a Yonatan Wexler. Na základě v této práci popsaném algoritmu SWT jsem vytvořil jeho vlastní implementaci, kterou jsem použil ve své práci.

K problému detekce textu přistoupili tak, že nejdříve je obraz převen na binární hranový obraz. V tomto obraze je provedena Stroke Width transformace. Následně jsou znaky seskupovány do řádků. K seskupování znaků do řádků jsou použity jejich vlastnosti jako: y-nová pozici v obraze, vzdálenost mezi znaky (rozdíl x-ových souřadnic sousedních znaků), barva a hodnota SW. Dále jsou řádky rozděleny na slova. Nakonec je pro jednotlivá slova určen ohraničující rámeček (bouding box).

Výhodou takto implementované detekce textu je malý počet chybně detekovaných oblastí. Další výhodou je rychlost tohoto algoritmu.

Kapitola 4

Knihovna Tesseract

Podrobnější popis najdete v [2], resp. [4]. Jedná se pravděpodobně o nejpřesnější Open source OCR. Tato knihovna je dostupná pro Linux, Windows i Mac OSX. Dále může být zkompilována i pro Android a iPhone.

Původně byla vyvíjena firmou HP jako proprietární software. V roce 2005 byla uvolněna jako Open source knihovna, a nyní je sponzorována společností Google.

Základními rysy Tessearctu ver. 3 jsou víceřádková detekce textu a podpora mnoha formátů vstupních obrazů. Tesseract ver. 2 uměl rozpoznávat text pouze v jednom řádku a obrázek musel být ve formátu TIFF [1]. Tesseract ver. 3 také podporuje formátovaný výstup využívající **hOCR**, což je podle [2] otevřený standard pro reprezentaci dat pro formátování textu získaného z OCR. Popis textu je uložen v XML souboru a obsahuje například definici kódování, styl, informace o uspořádání, úspěšnost rozpoznání a další. Tesseract využívá knihovnu Leptonica [3], což umožnilo podporu více formátů, na mém PC mám k dispozici podporu formátů: gif, jpeg, png a tiff. S příchodem třetí verze taktéž přibyla podpora více jazyků, nyní podporuje něco kolem 60 jazyků, včetně čestiny.

Před rozpoznáváním obrazu Tesseractem musí být nejdříve tento obraz předzpracován. Obraz po předzpracování by měl mít tyto vlastnosti:

1. musí být zvětšen tak aby, výška znaků byla alespoň 20px
2. text nesmí být natočen
3. text nesmí být zkosený
4. také je důležité vyříznout pouze oblast s textem.

Tyto poznatky jsou reflektovány v mém řešení při předzpracování částí obrazu obsahujícího text. Na následujícím obrázku 4.1 je vidět špatné rozpoznání natočeného textu.

Pokud by byly z textu extrahovány pouze oblasti s textem a dále by byly tyto oblasti natočeny tak, aby byl text vodorovně, rozpoznávání by bylo úspěšné.

[1] Neboli Tag Image FileFormat, slouží pro ukládání rastrových obrázků a je vhodný zejména pro tisk
[2] http://en.wikipedia.org/wiki/HOCR
[3] Je open source stránka obsahující software užitečný pro zpracování obrazu a jeho analýzu

Výstup Tesseractu
EST/U , , filigi'flu'th;'n/' '. SEAAGENTS

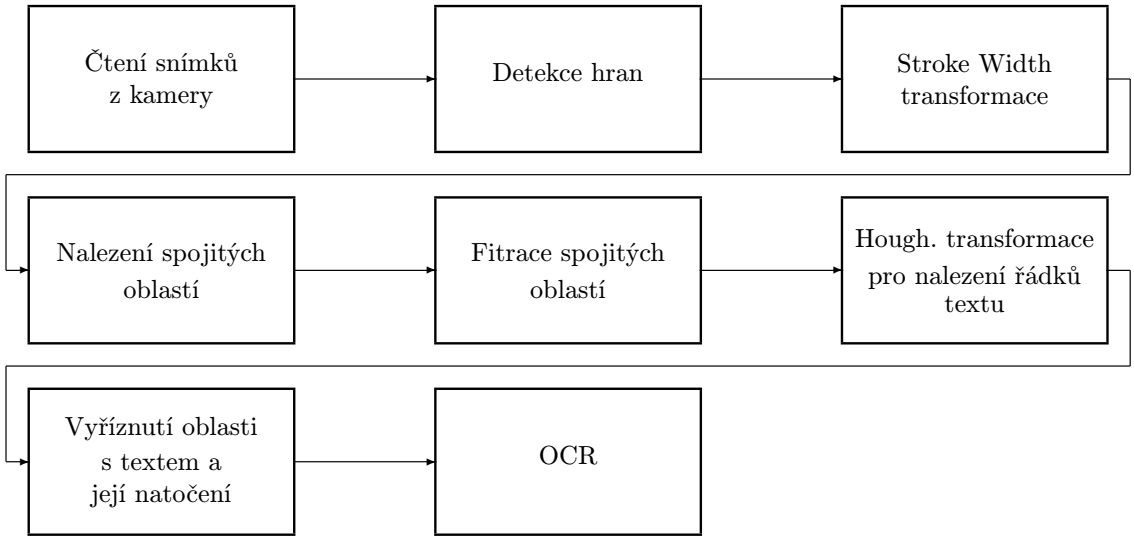
Tabulka 4.1: Chybný výstup Tesseractu (vlevo), kde vstupem je celý obraz (vpravo).

Kapitola 5

Návrh modulu pro detekci a rozpoznání textu

Cílem práce bylo vytvořit modul pro robota, který bude číst snímky z kamery nebo ze záznamu. Následně bude v těchto snímcích detekovat a rozpoznávat text. Výstupem modulu bude množina rozpoznaných nápisů (v textové podobě).

Jelikož je modul vytvářen pro robota, musí být schopný detekovat a rozpoznat text v přirozeném prostředí (město, budova, ...) a v krátkém čase.



Obrázek 5.1: Schéma návrhu řešení.

1. **Čtení snímků z kamery**
   Součástí je převod na rozlišení $640 \times 480\,px$ *a převod na šedotónový obraz.*

2. **Detekce hran**
   Je použit Cannyho hranový detektor, který je aplikován na vstupní šedotónový obraz.

3. **Stroke Width Transformace**
   Transformuje vstupní hranový obraz na ***SWT*** *obraz, postup – viz kapitola 2.2.*

4. **Nalezení spojitých oblastí**
   *Je využit **Flood Fill** algortimus. Výsledkem je množina ohraničujích obdélníků pro jednotlivé spojité oblasti.*

5. **Filtrace spojitých oblastí**
   Jsou ponechány pouze ty oblasti, které mohou představovat znaky. Pro filtraci jsou využita pouze jednoduchá pravidla, vycházející z šířky, výšky znaku a jejich poměru (aspect ratio).

6. **Houghova transformace**
   Využita pro seskupení spojitých oblastí (potenciálních znaků) do slov. Takto jsou seskupeny pouze ty znaky, jejichž nejspodnější pixely spolu tvoří přímky.

7. **Vyříznutí oblasti s textem a její natočení**
   Vyříznutí oblasti s textem a následné natočení tak, aby byl text v něm obsažený vodorovně.

8. **OCR**
   Rozpoznávání předzpracovaných oblastí a tisk výsledků. Dále je v této části provedena finální filtrace.

5.1 Předzpracování snímků z kamery, detekce hran a SWT

Snímky z kamery (záznamu z kamery) jsou čteny v pevně stanovených časových intervalech. Velikost snímků je upravena na požadované rozlišení (viz závěr kapitoly). Na obrázku 5.2a je uveden reálný příklad snímku získaného z kamery.

(a) Původní obraz. (b) Šedotónový obraz.

Obrázek 5.2: Snímek z kamery.

Následně je každý snímek převeden na šedotónový obraz (viz obrázek 5.2b). Důvodem je snažší nalezení hran v něm. Převod probíhá pomocí vzorce 5.1.

$$Y = 0.299R + 0.587G + 0.144B \tag{5.1}$$

který je postupně aplikován na každý pixel. Tímto se také redukuje použitá paměť pro uložení obrázku.

K nalezení hran jsem se rozhodl použít Cannyho hranový detektor – viz kapitola 2.1.2. Důvodem byly jednak jeho vlastnosti, jako je minimální počet chyb, přesnost a jednoznačnost. Druhým důvodem byla jeho již existující implementace v **OpenCV**.

Tento detektor pracuje se dvěma prahy. Hodnoty jednotlivých prahů jsem volil na základě doporučených hodnot, které jsem experimentálně dolaďoval – viz kapitola 7.2. Výstupem hranového detektoru je binární obraz, který je uložen do samostatné matice. Na následujím obrázku 5.3 je vidět obrazová reprezentace výstupu hranového dektektoru.

Obrázek 5.3: Hranový obraz získaný Cannyho hranovým detektorem.

Dále je provedena Stroke Width Transformace, podrobnosti k implementaci jsou uvedeny v 6.1. Vstupem algoritmu je původní obraz a hranový obraz. Z původního obrazu jsou určeny směry gradientů jednotlivých pixelů. Hranový obraz obsahuje hranové pixely, pro něž je **SWT** počítána.

Na obr. 5.4 je ukázán výstup po **SWT**. Jedná se o matici stejné velikosti jako je původní obraz a v dalším textu bude nazývána **SWT mapa**.

Obrázek 5.4: Hranový obraz po SW transformaci.

Intenzita pixelů odpovídá **Stroke Width** – čím vyšší tato hodnota je, tím je daný pixel světlejší. Tyto hodnoty jsou v intervalu $\langle 0; 30\rangle$ plus hodnota 255. Hodnota 255 pixelu, znamená, že tento pixel se nachází v pásu, jehož šířka je rovna 50 a více (viz optimalizace **??**). Hodnoty SW větší jak 30 nejsou s ohledem na maximální velikost detekovaných znaků důležité.

5.2 Nalezení spojitých oblastí a jejich filtrace

Hledání spojitých oblastí (angl. Connected Components, zkr. CCs) se provádí na základě hodnoty **Stroke Width**, v `SWT Mapě`. Toto jsem implementoval s využitím **FloodFill** algoritmu, implementace je uvedena zde 6.2.1. Výstupem algoritmu je jednak množina ohraničujících obdélníků jednotlivých spojitých oblastí, o kterých se bude později rozhodovat, zda daná oblast může představovat znak, a jednak množina **nejspodnějších pixelů** jednotlivých spojitých oblastí. Souřadnice těcho pixelů jsou později využity při detekci slov – viz kapitola (5.3).

Spojité oblasti jsou ukázány na obrázku 5.5.

Obrázek 5.5: Spojité oblasti nalezené semínkovým vyplňováním v SWT mapě.

Z takto nalezených spojitých oblastí se ponechají pouze ty, které splňují jednoduchá filtrační pravidla, jako je *velikost znaku*, dalším pravidlem je *Aspect Ratio* [1], posledními pravidly je *průměrná barva* oblasti. První dvě pravidla jsou definována intervaly, jejichž hodnoty jsou uvedeny v kapitole 7.2. Poslední pravidlo zajišťuje to, že pokud je detekován tmavý text na světlém pozadí (viz dále), tak jsou vyloučeny všechny světlé spojité oblasti. A naopak, pokud je detekován světlý text na tmavém pozadí, jsou vyloučeny všechny tmavé spojité oblasti. Mezní hodnoty intenzit (barev) jednotlivých spojitých oblastí (tmavé vs. světlé) byly experimentálně zjištěny – viz sekce 7.2.Dále se vyloučí takové spojité oblasti, které jsou součástí jiných.

Tato jednoduchá pravidla však **nezajistí**, že vyfiltrované spojité oblasti představují pouze znaky (naopak pouze zlomek z této množiny skutečně představuje znaky). Množinu těchto znaků můžeme nazvat jako **Množina kandidátů na znaky**, graficky je znázorněna na obrázku 5.6. Toto se řeší až ve fázi detekce slov, která je popsána v následující kapitole 5.3, kde je využita nejzásadnější vlastnost znaků, a to že jsou umístěny v řádcích, ve kterých tvoří slova.

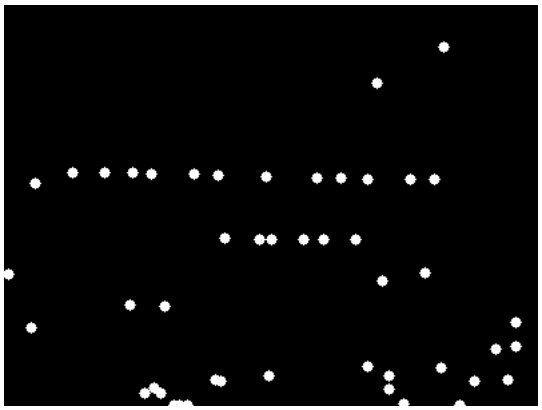
[1] Poměr výška/šířka.

Obrázek 5.6: Kandidáti na znaky.

5.3 Seskupení potenciálních znaků do slov a předzpracování pro OCR

Téměř všechen text je umisťován v řádcích, které tvoří v 2D prostoru přímky. Tyto přímky mohou být ve snímku různě natočené, což může být způsobeno například perspektivní projekcí nebo jen jeho prostým natočením. Proto jsem se pro nalezení přímek představujících řádky textu rozhodl použít **Houghovu transformaci**.

K tomu, aby mohla být použita, musel jsem patřičně vytvořit vstupní binární obraz – viz obrázek 5.7. Bílé kruhy představují znaky, bližší informace naleznete v kapitole – implementace – viz 6.3.



Obrázek 5.7: Body představující polohu znaků.

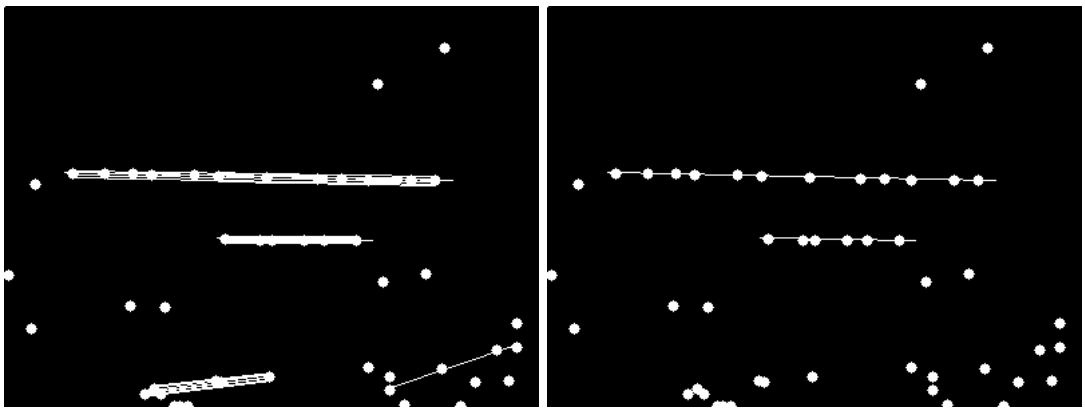
K nalezení přímek, resp. úseček, jsem použil **Pravděpodobnostní Hougovu transformaci** (dále jen HT), jejímž výstupem je množina souřadnic koncových bodů nalezených úseček. Vstupní parametry této metody jsou práh (určuje min. počet bodů, které úsečka protíná), maximální mezera a minimální délka úsečky. Hodnoty těchto parametrů jsou uvedeny v kapitole 7.2.

Aplikováním HT jsme získali množinu všech úseček, jak vidíme na obrázku 5.8. Dále je nutné vybrat pouze nejdelší z podobných úseček.

Toto je v programu řešeno tak, že se nejdříve seřadí vektor úseček dle jejich délky. Následně je daný vektor iterativně procházen a do nového vektoru jsou ukládány ty úsečky, pro které platí, že se tam **nenachází** nějaká úsečka se stejnými hodnotami parametrů k a q. Význam parametrů je vidět v rovnici 5.2 (jedná se o směrnicové vyjádření přímky).

$$y = kx + q \tag{5.2}$$

Pokud má daná úsečka podobné hodnoty parametrů k a q, tak se přidá pouze pokud její x-ové souřadnice koncových bodů neleží uvnitř intervalu x-ových souřadnic podobné úsečky. Jinak by bylo nalezeno pouze jedno slovo na řádku. V další fázi jsou eliminovány ty úsečky, jejichž směrnice k je větší než 0.5, resp menší než -0.5 (tedy maximální sklon textu je 22.5 °). Nakonec jsou vyřazeny ty úsečky jejichž koncové body leží ve znacích s různou průměrnou barvou (resp. průměrnou intenzitou), tady se předpokládá, že daný text bude stejné barvy. Pokud by nebyl, tak nebude detekován.



Obrázek 5.8: Vlevo: úsečky nalezené HT; vpravo: vyfiltrované úsečky.

Po detekci řádků textu je nutné extrahovat pouze ty části původního obrazu, kde se nachází text. A následně provést natočení tak, aby byl text vodorovně. Pokud by tyto dva body nebyly provedeny, následné rozpozání by nebylo úspěšné. Jinými slovy, čím přesnější bude „vyříznutí“ a správné natočení textu, tím úspěšnější bude i jeho rozpoznání.

K vyříznutí a natočení textové oblasti jsou využity vyfiltrované úsečky a velikost prvního a posledního znaku. Nejdříve se musí jednotlivé úsečky „protáhnout“ tak, aby zahrnovaly celý nápis (doposud byly pouze mezi nejspodnějším pixelem prvního a posledního znaku).

Dále se určí ohraničující obdélník, který je dán rozměry a středem. Šířka obdélníku je dána délkou dané úsečky, výška je dána velikostí **většího** ze znaků.

Následně se provede natočení původního obrazu. Úhel natočení odpovídá natočení úsečky a lze ho určit následovně:

$$\arctan2(P_2.y - P_1.y, P_2.x - P_1.x) * \frac{180}{\pi}, \tag{5.3}$$

kde P_2 a P_1 jsou koncové body zpracovávané úsečky.

Nakonec se z natočeného obrazu vyřízne ohraničující obdélník. Důvodem, proč to nedělám opačně, ač by se to zdálo smysluplnější, je to, že pokud bychom prováděli natočení vyříznuté oblasti, byly by součástí extrahované oblasti i černé oblasti, resp. oblasti s hodnotou nula. Výsledek je vidět na obrázku 5.9.

Obrázek 5.9: Extrahované oblasti s textem.

Výše popsaný postup detekce slov je navržený pro relativně malá slova. Stejně tak **Stroke Width transforamce** je provedena pouze pro jeden směr gradientu, tedy jsou detekovány pouze tmavé nápisy (texty) na světlém pozadí. Proto je ve výsledném detektoru nejdříve provedeno zmenšení vstupního obrazu, v něm jsou detekovány nápisy a uloženy do obrazových matic. Následně je provedena detekce slov pro původní (nezmenšený) obrázek. Toto zaručí poměrně přesnou (paramtery detektoru jsou nastaveny pro úzkou množinu detekovaných nápisů) detekci nápisů téměř libovolné velikosti.

Postup napsaný v předchozím odstavci je následně aplikován pro opačný směr gradientu u **Stroke Width transformace**, což zaručí detekci světlých nápisů (textů) na tmavém pozadí.

Takto získáme množinu extrahovných obrazů obsahující nápisy libovolné velikosti a světlé nápisy na tmavém pozadí, resp. tmavé nápisy na světlém pozadí.

Tato množina extrahových oblastí je následně předána k **OCR**.

5.4 OCR a finální filtrace slov

K OCR jsem použil knihovnu **Tesseract**. Na začátku se provede inicializace API, kde se nastaví jazyk. Dále se nastaví mód, ve kterém bude pracovat. Já jsem zvolil „jednotlivé bloky“ textu. Následuje postupné nastavování vstupních obrázků a čtení textu z nich. Takto přečtený text je dále filtrován. Za korektní text považuji pouze takový, který obsahuje pouze alfanumerické znaky, apostrofy a některá interpunkční znaménka (.!?). Výsledek detekce pro náš příklad je to množina o třech slovech 5.4.

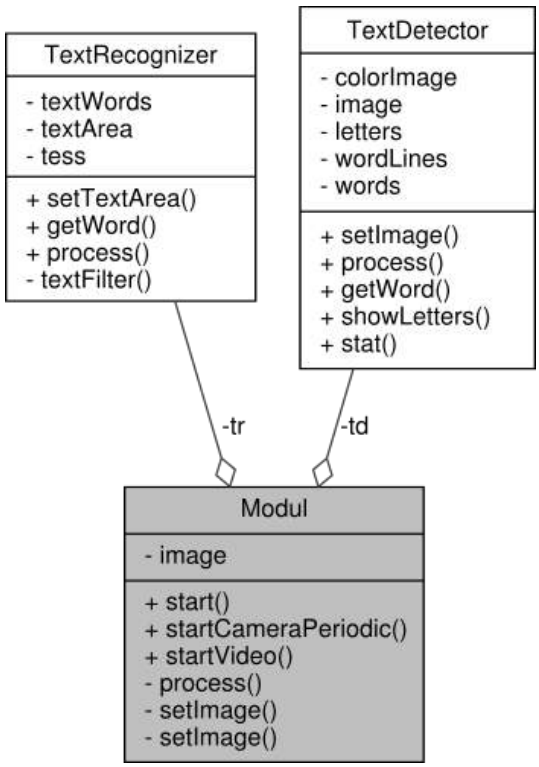
Jelikož je Tesseract při natočených nápisech poměrně nepřesný, musí být nápisy nejdříve natočeny tak, aby byly vodorovně. Lehce perspektivně zkreslený obrázek nečiní **Tesseractu** problém.

$$\mathrm{OCR}_{\mathrm{out}} = \{\mathrm{ESTATE}, \mathrm{AGENTS}, \mathrm{SAXONS}\} \tag{5.4}$$

Kapitola 6

Implementace

Program je implementován v jazyce C++ na 64-bit architektuře s operačním systémem Linux (konkrétně **Fedora** 21). K implementaci jsem použil Open Source knihovnu **OpenCV**[1] ve verzi 2.8. Tato knihovna je zejména určena pro počítačové vidění (CV – Computer Vision). K rozpoznání textu je použita knihovna **Tesseract**[2] ve verzi 3.03. Diagram tříd je na obrázku 6.1. Dále budou uvedeny pouze zajímavější části implementace, podrobný popis tříd a funkcí naleznete v *programové dokumentaci*. Tato dokumentace je automaticky generována aplikací **Doxygen**.



Obrázek 6.1: Diagram tříd.

[1] http://opencv.org/
[2] https://code.google.com/p/tesseract-ocr/

6.1 Poznámky k implementaci SWT

Implementace přesně vychází z algoritmu 2.2.1, popsaného v [7].

Pro aproximaci derivací používám **Sobelovy** fitry, které jsou aplikovány na původní šedotónový obraz. Pro aplikování filtrů je použita funkce `Sobel()`. Výpočet směru gradientu je proveden funkcí `phase()`, jejíž vstupem jsou právě výstupy (matice diferencí ve směru osy x resp. osy y pro jednotlivé pixely) funkce `Sobel()`.

Pozn.: Jelikož směr gradientu je potřeba pouze pro pixely v místech, kde se nachází hrany, jsou zde některé hodnoty gradientů spočítány zbytečně. Nicméně doba výpočtu je v porovnání s rozpoznáváním znaků zanedbatelná.

Pro simulování pohybu po paprsku ve směru gradientu je použita třída `LineIterator`. Jedná se, jak název napovídá, o úsečku, po jejíchž pixelech se lze postupně pohybovat (klasicky inkrementací iterátoru této třídy).

Tato úsečka je definována dvěma body, které jsou určeny následovně takto 6.1.

$$P_1 = (x, y) \qquad P_2 = (x + \text{delka} * \cos(\text{grad}(\text{x}, \text{y})), y + \text{delka} * \sin(\text{grad}(x, y))), \tag{6.1}$$

kde grad je matice směru gradientů jednotlivých pixelů a delka je délka úsečky.

Během „pohybu“ po úsečce jsou ukládány souřadnice pixelů, přes které se prochází, do vektoru. Těmto je pak nastavena hodnota odpovídající délce úsečky.

Hodnotu **delka** jsem zvolil 255, protože je to maximální hodnota 1B, kterými je tvořena matice pro uložení **SWT**.

Algoritmus jsem optimalizoval tak, že pokud hodnota pomocné proměnné vzroste nad 30 pixelů, tak se ukončí iterace pro danou úsečku. Toto si můžu dovolit s přihlédnutím k danému rozlišení vstupního obrazu a maximální velikosti znaků.

6.2 Obarvování spojitých oblastí

Využil jsem algoritmus **Flood Fill**. Algoritmus obarvování spojitých oblastí je uveden zde 6.2.1.

Algoritmus 6.2.1. *Implementace algoritmu pro nalezení spojitých oblastí.*

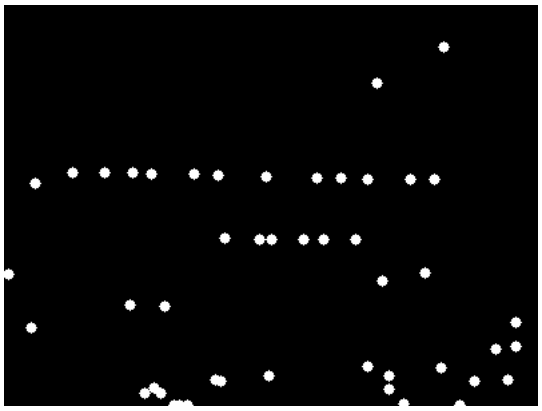
1. *Proveď kopii SWT mapy do proměnné* ***ccs****. (Původní SWT mapa je ponechána pro testování.)*
2. *Nastav „barvu“ –* ***color*** $= 256$
3. *Vyber pixel* $s[x, y] \in$ **ccs**, *pro který platí* $s[x, y] < 254$. *(Tento pixel představuje semínko (seed), pro Flood Fill algoritmus.)*
4. *Algoritmem* ***Flood Fill*** *se semínkem s „obarvi“ spojitou oblast „barvou“* ***color****, pro obarvování využij* ***8-okolí*** *pixelů viz obrázek 2.7.*
5. *Najdi ohraničující obdélník obarvené oblasti. (Výstupem algortimu (v OpenCV) je také tento ohraničující obdélník.)*
6. *Najdi souřadnice* ***nejspodnějšího*** *pixelu.*
7. *Inkrementuj* ***color*** *a pokračuj bodem 3.*

„Obarvovnání“ spojitých oblastí je prováděno „barvou“ s hodnotou 256 a vyšší. Důvodem je, že „obarvování“ je prováděno v kopii **SWT Mapy** a hodnoty do 255 včetně, jsou hodnoty v ní použité.

6.3 Příprava vstupního obrazu pro Houghovu transformaci

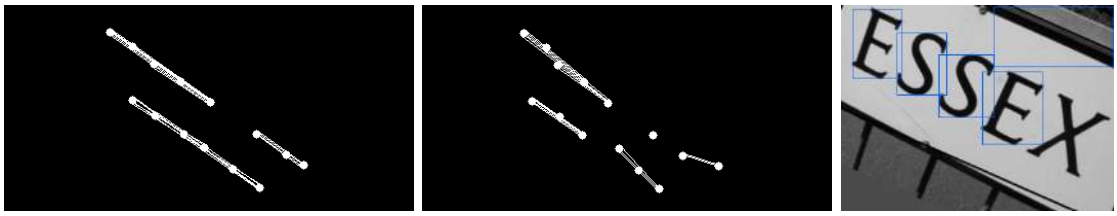
Vytvoření vstupního obrazu pro Houghovu transformaci jsem provedl tak, že pro každý znak z množiny **kandidátu na znaky** (nalezených a vyfiltrovaných spojitých oblastí), jsem vložil do obrazu na pozici **nejspodnějšího pixelu** kruh o průměru **větším** než 1 (konkrétní hodnota – viz kapitola 7.2). Kruh jsem volil, protože má v libovolném směru stejnou délku sečny jdoucí přes jeho střed (např. na rozdíl od čtverce). Nastavení pouze jediného pixelu by nestačilo, protože pozice znaků na řádku se **mírně** liší, a tedy přímka by nebyla detekována.

Vše je názorně ukázáno na obrázku 6.2.



Obrázek 6.2: Body představující polohu znaků.

Důvod, proč je nutné vkládat **kruh** na **nejspodnější pixel** (a tedy nutné i tento pixel najít) a ne například na levý spodní roh ohraničujícího obdélníku (který známe z ohraničujícího rámečku spojité oblasti), je vidět na obrázku 6.3. Zatímco v prvním případě je přímka správně detekována, v druhém případě jsou body rozptýleny tak, že přímka není správně detekována. Toto má vliv pouze na natočený text, kdy spodní strana ohraničujícího obdélníku kandidátů na znaky *nelícuje* se spodní částí znaku – viz obrázek 6.3 vpravo (pozn.: tento obrázek nesouvisí s dvěmi předchozími).



Obrázek 6.3: Vlevo: Pro tvorbu vstupího obrazu pro HT jsou použity levé dolní rohy oh. obd.; uprostřed: jsou použity nejs. pixel znaku; vpravo: ukázka „nelícování“ spodní strany a spodní části znaku.

Kapitola 7

Experimenty

S programem jsem provedl hned několik experimentů. První experimentování bylo provedeno za účelem doladění parametrů detektoru – viz kapitola 7.2. V druhém experimentování byla zjištěna kvalita textového detektoru pomocí funkcí **recall** a **precision**. Způsob určení kvality je uveden v kapitole 7.1.3.

Také byly experimentálně zjistěny mezní hodnoty natočení textu a vlivu perspektivní transformace tak, aby byl text správně detekován (viz kapitola 7.2.3 a 7.2.4). Poslední experimenty se týkaly správného rozpoznání textu – viz kapitola 7.3.

7.1 Detekce textu

Testování bude provedeno na množině a podmnožině snímků z [3]. Jedná se o veřejně dostupnou sadu snímků, která vznikla za účelem Sedmé mezinárodní konference pro analýzu a rozpoznávání dokumentů [1] již v roce 2003. Tato sada podle mne obsahuje snímky vhodné pro simulaci výstupu kamery reálného robota. Obsahuje různé nápisy na budovách, etikety, cedule, ukazatele, a další nápisy vyskytující se v přirozeném prostředí.

Testování jsem nejdříve provedl na celé testovací sadě. Následně jsem z ní vybral množinu obrazů, na kterých se vyskytovaly cedule, dopravní ukazatele a tak dále. V následujícím textu bude tato monožina obrazů nazývána testovací sada „informační tabule“. Toto mi umožnilo utvořit dílčí závěr pro danou množinu obrazů. Popis obou testovacích sad je uveden v následujících dvou kapitolách.

Pro testování vlivu natočení a vlivu perspektivní deformace jsem vytvořil vlastní testovací sadu. V této kapitole bude uvedeno pouze pár obrázků pro ilustraci funkce detektoru, více obrázků naleznete v příloze – viz kapitola 8.

7.1.1 Celá testovací sada

Tato sada obsahuje 243 snímků. Tuto sadu jsem zvolil, protože se mi jevila komplexní, neboť obsahuje rozličné druhy nápisů (různý řez, formát, velikost a barva písma) na různých podkladech (stěny budov, cedule, etikety, atd. . .). Vzorek testovací sady je vidět na obrázku 7.1.

[1] Bližší informace zde: http://algoval.essex.ac.uk/icdar/Competitions.html

Obrázek 7.1: Vzorek obrázků z celé sady.

7.1.2 Podmonožina testovací sady – informační tabule

Tato podmnožina obsahuje zejména různé cedule, dopravní ukazatele a tak dále. Její velikost činní 40 snímků. Vzorek této sady je uveden na následujícím obrázku zde 7.2.

Obrázek 7.2: Vzorek obrázků první sady.

7.1.3 Metriky testování textového detektoru

Níže jsou popsány metriky použité pro vyhodnocení kvality detektoru na dvou výše popsaných testovacích sadách. Tyto metriky se běžně používají pro určení kvality detektoru textu – viz [3] a jsou definovány následovně. Pro každý obrázek se určí *precision* a *recall* – viz rovnice 7.1, resp. 7.2.

$$prec = \frac{\sum_{e \in E} m(e, T)}{|E|} \tag{7.1}$$

$$recall = \frac{\sum_{t \in T} m(t, E)}{|T|}, \tag{7.2}$$

kde E je množina detekovaných oblastí, a T je množina správných oblastí (součástí testovací sady). Obě množiny obsahují pozici a rozměry ohraničujících obdélníků. Funkce m(e, T) je definována 7.3.

$$m(e, T) \begin{cases} 1 & \text{pokud } e \in T \\ 0 & \text{jinak} \end{cases} \tag{7.3}$$

Příslušnost $e \in T$ provádím **vizuálně** tak, že pokud ohraničující rámeček (bounding box), **přibližně** obklopuje text.

Jinými slovy – funkce **precision** určuje poměr nalezených správných textových oblastí vůči všem nalezeným oblastem. A **recall** určuje poměr nalezených správných textových oblastí a všech správných oblastí.

Z těchto dvou hodnot se následně určí kvalita detektoru následovně 7.4:

$$f = \frac{1}{\frac{\alpha}{prec} + \frac{1-\alpha}{recall}} \tag{7.4}$$

Pomocí parametru α lze ovlivnit váhu jednotlivých vlastností. Zvolil jsem $\alpha = 0.5$, tedy vlastnosti mají stejné váhy.

7.2 Volba jednotlivých parametrů detektoru

Toto experimentování sloužilo k natrénování vhodných parametrů a probíhalo na celé testovací sadě. Přehledný seznam parametrů a jejich hodnoty jsou uvedeny v tabulce 7.1. Nejdříve jsem zkoumal velikost dolního prahu (**low_thresh**) u Cannyho hranového detek-

Metoda	**Parametr**	**hodnota**	**Poznámka**
Cannyho detektor hran	spodní práh	200	
	horní práh	280	$1.4 \times low\_thresh$
	velikost Gaussova filtru	3×3	
Hledání spojitých oblastí	max. výška	50 px	
	min. výška	17 px	
	aspect ratio	$\langle 0.1, 1.6 \rangle$	
Pr. Houghova transf.	min. délka úsečky	60 px	
	max. mezera mezi znaky	60 px	
	min. počet pixelů	18 px	3 znaky – (3×6)
	směrnice přímky	$\langle -0.5, 0.5 \rangle$	$\pm 22.5°$

Tabulka 7.1: Experimentálně dolaďované parametry a jejich hodnoty.

toru. Experimentálně jsem zjistil, že optimální hodnota prahu je 200, hodnota horního prahu (**high_thresh**) je pak $1.4 \times$ **low_thresh** neboli $1.4 \times 140 = 280$. Další hodnotu, kterou bylo nutné zjistit, byla velikost **Gausova** filtru. Jako optimální se jevila minimální velikost, tedy velikost 3×3. Při větších rozměrech byl vstupní obraz příliš rozmazán, a proto bylo nalezeno málo hran.

V dalším kroku bylo zapotřebí ověřit hodnoty jednoduchých filtrovacích pravidel pro filtrování **spojitých** oblastí za účelem eliminace těch, které nemohou představovat znaky. Mezi tato pravidla patří minimální a maximální velikost znaku. Minimální výšku znaku (spojité oblasti) jsem zvolil $17px$ a maximální $50px$. Pro **Aspect ratio** jsem zvolil interval $\langle 0.1, 1.6 \rangle$. Tímto intervalem projdou i znaky spojené dohromady (toto se stává, pokud je mezi znaky malá mezera nebo pokud je text psán psace).

Následně bylo nutné doladit **Pravděpodobnostní Houghovu transformaci**, zde bylo nutné zvolit hodnoty parametrů: **minLineLength**, **maxGap**, **thrash**. Parametr **minLineLength**, který představuje minimální velikost detekované úsečky, jsem zvolil 60. Maximální mezeru, neboli parametr **maxGap**, jsem zvolil 60. Nejpodstatnější parametr **thresh** jsem na základě experimentů zvolil na 18. Shrnu-li hodnoty těchto parametrů, tak to znamená, že HT je schopna detekovat nápisy, které jsou alespoň 60px dlouhé, maximální vzdálenost znaků je přibližně 60px. A minimální počet znaků je zhruba 3, protože znak je v obraze představován jako kruh o průměru 6px.

Jako poslední jsem hledal vhodné mezní hodnoty směrnice k detekovaných přímek, zvolil jsem interval $\langle -0.5, 0.5 \rangle$. Toto poměrně zúžilo množinu detekovaných přímek, aniž by byly odstraněny ty, které představují text. Dalším dobrým pravidlem – poměr vzdáleností

prvních a posledních koncových bodů horní a dolní textové ohraničující přímky, jsem volil v intervalu $\langle 0.6, 1.4 \rangle$. Tímto pravidlem projde i mírně perspektivně transformovaný text, který ještě dokáže **Tesseract** rozpoznat.

S takto nastaveným detektorem bylo následně provedeno testování jednotlivě na obou sadách.

7.2.1 Výsledky detekce pro celou testovací sadu

Kvalita detektoru nad touto testovací sadou je podle vzorce 7.4 – 0.5 (hodnota precision je 0.6 a recall 0.43). S ohledem na složitost testovací sady jsou výsledky detekce dobré. Mnoho nápisů nebylo detekováno kvůli tomu, že daný text byl moc malý. Dalším důvodem bylo to, že vstupní snímek obsahoval odlesky (z blesku fotoaparátu, resp. kamery), které výrazně zasahovaly do oblastí s textem, a tím znemožnily správnou detekci hran hned v úvodní fázi detekce textu. Dalším problémem při detekci textu byla velká, resp. malá, vzdálenost jejich sousedních znaků. Problémem při malé vzdálenosti bylo „slití“ více znaků dohromady ve fázi hledání **spojitcýh oblastí**.

Na obrázku 7.3 můžete vidět příklad úspěšné detekce textu. V příloze 8 jsou uvedeny další příklady, včetně neúspěšné detekce.

Obrázek 7.3: Ukázka správně detekovaných oblastí s textem.

V [7] dosáhly funkce kvality $f = 0.66$, taktéž je zde uvedena tabulka kvalit jiných řešení, kde minimální hodnota kvality je 0.08, průměr pak činní 0.38. Tento detektor má kvalitu $f = 0.5$. Nicméně srovnání je pouze orientační, jelikož v mé práci není provedena segmentace na slova (není nutná, a navíc by prodloužila dobu zpracování snímku), nýbrž jsou detekovány celé řádky.

7.2.2 Výsledky detekce pro testovací sadu – informační tabule

V tomto testování byla kvalita detektoru $f = 0.79$. Tedy kvalita detektoru nad touto testovací sadou je zhruba o 58% lepší ($0.79/0.5 \times 100\%$) než nad celou testovací sadou. Proto můžu utvořit dílčí závěr – tedy, že je možné použití tohoto detektoru na informační tabule, s dobrou kvalitou detekce f.

Na obrázku 7.4 je uveden příklad **správně** detekovaného textu.

Špatně detekovaný text je vidět na obázku 7.5.

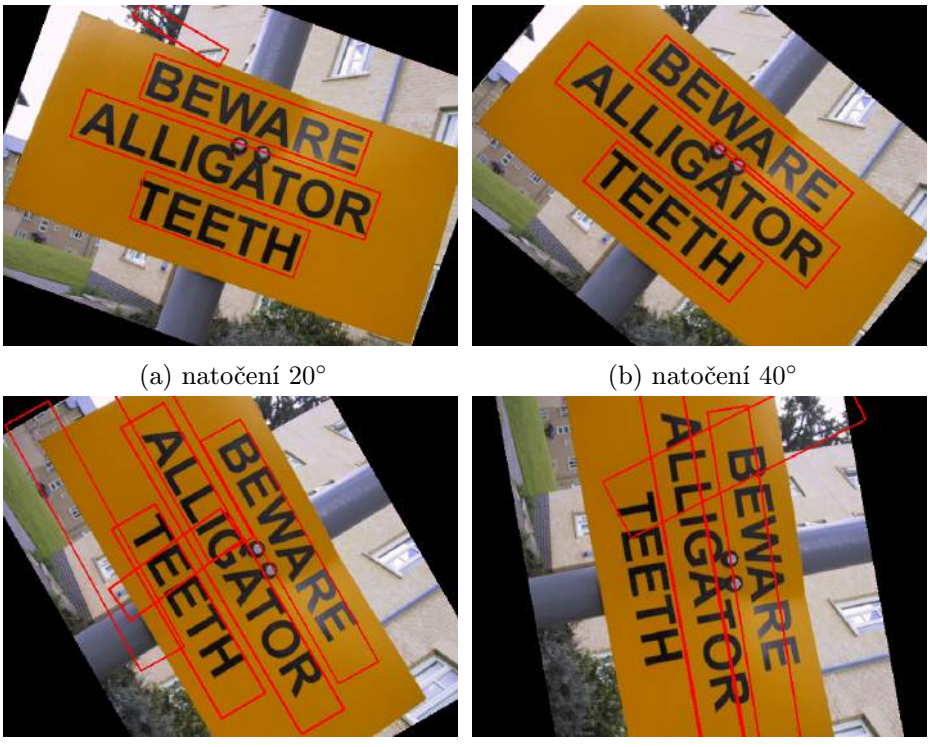
7.2.3 Natočení textu

Pro toto testování jsem jednak vybral 4 obrázky z testovací sady, které měly v předchozím testu 100% úspěšnost detekce textu. Tyto obrázky jsem natáčel o 20° po a proti směru hodinových ručiček a vizuálně jsem zkoumal, zda je správně detekována textová oblast. Zde uvádím pouze test pro dva obrázky 7.6 a 7.7, zbytek můžete nalést v příloze 8.

Obrázek 7.4: Ukázka správně detekovaných oblastí s textem.

Obrázek 7.5: Ukázka špatně detekovaných oblastí s textem.

Dále jsem provedl testování na testovací sadě – informační tabule (viz kapitola 7.1.2), ve které jsem jednotlivé obrázky natočil o 20°, podle jejich středu (natočení jsem provedl v aplikaci).

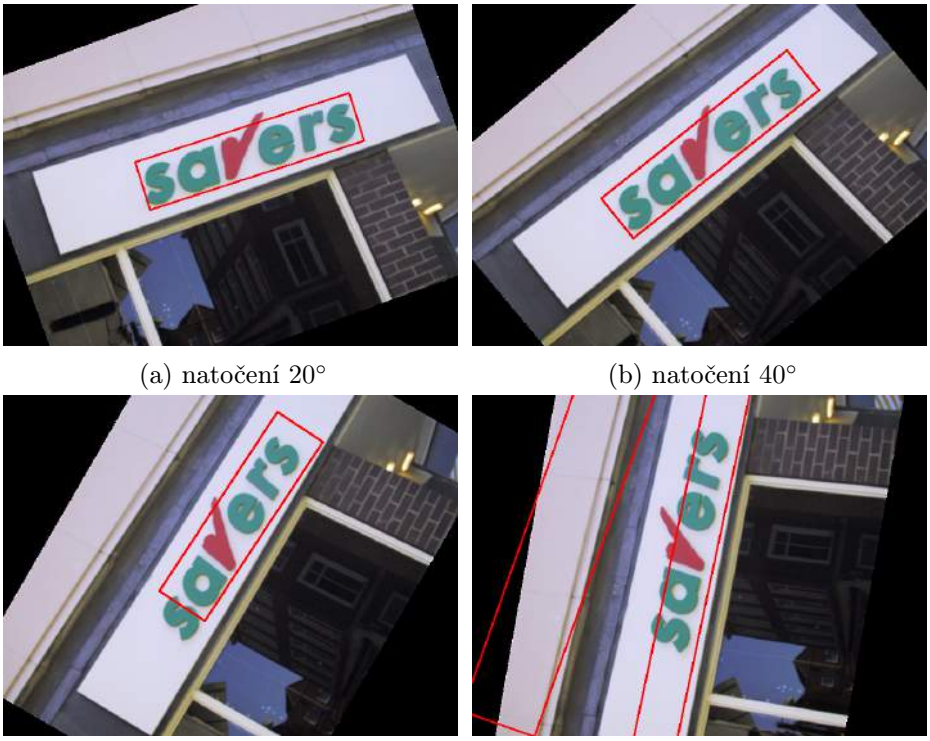


(a) natočení 20° (b) natočení 40°

(c) natočení 60° (d) natočení 80°

Obrázek 7.6: Detekce natočeného textu (natočení po směru hod. ruč.).

Z obrázků z první části testování natočení textu je patrné následující: Pokud je text natočený v libovolném směru o úhel z intervalu $\langle 0°; 60° \rangle$, je text obsažený v obrázku detekován



(a) natočení 20° (b) natočení 40°

(c) natočení 60° (d) natočení 80°

Obrázek 7.7: Detekce natočeného textu (natočení proti směru hod. ruč.).

ve většině případů správně. Pokud je natočení větší než 60°, je text detekován špatně, což je způsobeno pozicí „nejspodnějších pixelů“ znaků, které již netvoří (ani přibližně) přímku, a proto nejsou detekovány **Houghovou transfomací**. Navíc jsou v natočeném textu s více řádky detekovány přímky mezi znaky, které nejsou na řádku, ale které jsou na různých řádcích – viz 7.6d, taktéž ohraničující rámeček není určen korektně, což je vidět v tomtéž obrázku a i na obrázcích 7.7c a 7.7d. U prvního snímku byl ve všech správně detekovaných obdélnících po zpětné transformaci (natočení) správně rozpoznán text. U snímku 7.7 nikoliv, což však nebylo způsobeno výběrem a transformací ohraničujícího obdélníku, nýbrž výskytem netypického znaku uvnitř nápisu.

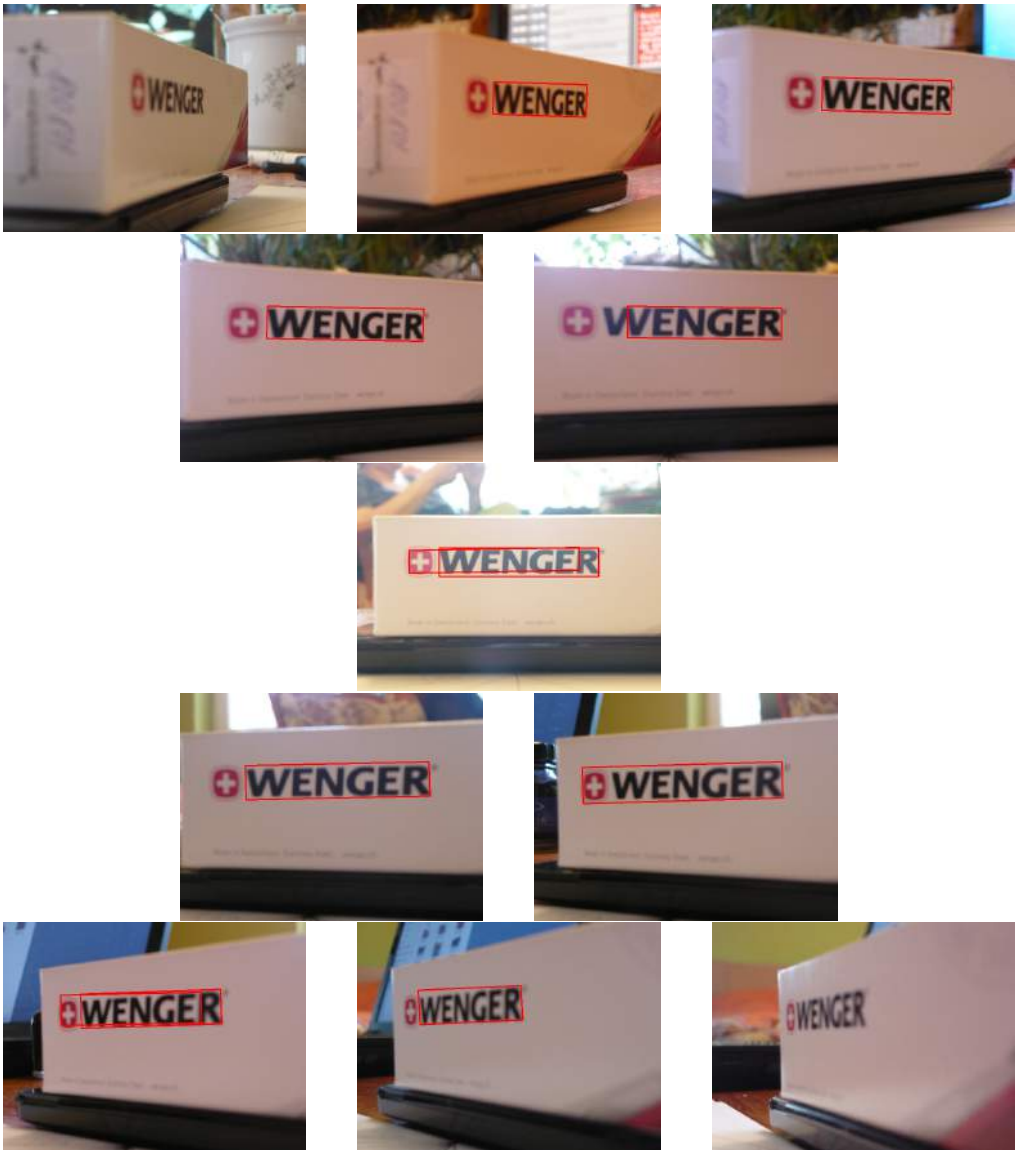
Co se týče testování na podmnožině testovací sady „informační tabule“, byla kvalita detektoru $f = 0.75$ (podle vzorce 7.4), což je jen o 1% horší, než pokud by text nebyl natočen (respektive byl natočen pouze mírně).

7.2.4 Vliv perspektivního zkreslení

Pro toto testování jsem vytvořil sadu o velikosti 5×11 snímků. Vybral jsem 5 objektů s textem. Pro každý jsem provedl focení z různé perspektivy, celkem 11 snímků. Při focení jsem se přibližně pohyboval po kružnici (se středem v místě s objektu s textem) o zhruba 15°. Výsledek detekce pro jeden z pěti snímků je uveden na obrázku 7.8.

Dále jsem upravil (perspektivně deformoval) několik obrázků z původní testovací sady. Příklad detekce textu v těchto obrazech je vidět na obrázku 7.9, zbytek obrázků je uveden v příloze 8.

Vliv perspektivního zkreslení pouze podle osy y nemá zásadní vliv na správnou detekci



Obrázek 7.8: Detekce perspektivně zdeformovaného textu.

Obrázek 7.9: Detekce perspektivně zdeformovaného textu.

textu, jak je vidět z obrázku 7.8. Text není detekován až tehdy, kdy je úhel natočení od

směru pohledu kolmého na sledovaný objekt větší než 60°. Co se týče obecné perpektivní deformace, tak ta také nečiní detektoru zásadní problémy – viz obrázek 7.9.

7.3 Rozpoznání textu

Úspěšnost rozpoznání textu závisí především na výběru správné oblasti s textem a následnému natočení textu do vodorovné polohy. Na obrázku 7.2 je vidět výsledek experimentu, kde v prvním případě je předáno **Tesseractu** jako vstup celý obraz, zatímco v druhém případě jsou předány pouze ty části, které obsahují text. Z obrázku 7.2 je patrný rozdíl v úspěšnosti rozpoznání textu. Zejména je zřejmá chybná detekce slov, které se v obraze nevyskytují. Nicméně je nutné konstatovat, že většina nápisů, které se v obraze vyskytovaly, byly **Tesseractem** správě detekovány a rozpoznány. V případě, že je text v obraze natočen, Tesseract selhává – viz kapitola 4, obrázek 4.1.

detekce + OCR	**pouze OCR**
[Alarm **Fitted** **Alarm**	if” iAA’fl´x”; :9 ” ¿ ” **Fitted** Ala

Tabulka 7.2: Srovnání (viz tabulka vlevo) rozpoznávání pouze na extrahovaných oblastech obrazu versus na celém obrazu (vpravo).

Dalším kritériem úspěšnosti rozpoznání, které já neovlivním, je samotná kvalita **Tesseractu**. Nicméně na základě experimentů je zřejmé, že je pro tuto práci více něž dostačující.

Na testovací sadě „informační tabule“, bylo správně rozpoznáno 92% textu ve správně detekovaných textových oblastí, tuto hodnotu můžeme nazvat jako úspěšnost rozpoznávání. A tedy pokud funkce **recal** byla rovna hodnotě 0.83 a úspěšnost rozpoznání správně detekovaných oblastí 92%, tak je modul jako celek schopný v obrázku správně „přečíst“ 75% nápisů ($0.82 * 0.92 = 0.75$) nad testovací sadou „informační tabule“.

7.3.1 Závěr testování

Z experimentů je patrné, že kvalita f detektoru pro celou testovací sadu je 0.5, pro „informační tabule“ 0.79, pro natočené „informační tabule“ 0.75. Toto je přehledně (včetně hodnot **pression** a **recall**) ukázáno v tabulce 7.3.

Dále je patrný malý vliv na kvalitu detekce při natočení textu o méně než 40°. Perspektivní projekce také nečiní detektoru problémy. U rozpoznávání, které je dáno především kvalitou použitého OCR API (Tesseractu), byl však zjištěn zásadní vliv natočení textu na úspěšnost rozpoznávání. S natočeným textem si Tesseract neporadí, a proto je nutné provést natočení jednotlivých oblastí s textem.

Co se týče porovnání s ostatními textovými detektory, stojí si tento detektor se svou kvalitou $f = 0.5$ poměrně dobře. Například v [7] dosáhly kvality $f = 0.66$, což však byla

kvalita nejlepšího detektoru. Průměr kvalit zmíněných v tytéž práci pak činil 0.38. Toto srovnání je však pouze orientační, jelikož v mé práci není provedena segmentace textu na slova.

Testovací sada	**počet obrázků**	**precision**	**recall**	**quality**
celá sada	243	0.6	0.43	0.5
„informační tabule“	30	0.76	0.82	0.79
natočené „informační tabule“ o 20°	30	0.69	0.82	0.75

Tabulka 7.3: Tabulka výsledků detekce textu.

Kapitola 8

Závěr

Zadáním této práce bylo vytvořit modul pro detekci a rozpoznávání textu. Toto zadání bylo splněno. Pro detekci textu byla zejména použita metoda Cannyho hranového detektoru v kombinaci s metodou Stroke Width transformace. Pro seskupení potenciálních znaků do řádku byla použita Houghova transformace. K rozpoznávání detekovaných řádků jsem použil knihovnu (API) Tesseract.

Experimentálně byla ověřena funkčnost modulu a určena kvalita detektoru, jejíž hodnota pro celou testovací sadu činila 0.5, pro její podmnožinu, obsahující informační tabule, pak 0.79. Dále byl zkoumán vliv natočení a perspektivní deformace textu. Zde bylo zjištěno jednak to, že na detekci nemá zásadní vliv jeho natočení, které je menší než 40°, na čemž má zásadní podíl Houghova transformace, a jednak to, že rozpoznání natočeného textu činí Tesseractu problém. Nicméně pokud se provede natočení textu do vodorovné polohy, rozpoznání bude úspěšné. Co se týče perspektivní deformace textu, tak bylo zjištěno, že také nemá zásadní vliv na úspěšnost detekce a následného rozpoznávání. Bližší informace naleznete v kapitole 7.1.

Z orientačního srovnání kvalit mého a jiných textových detektorů uvedených v [7] plyne, podle mne, slušný závěr. Neboli kvalita mého detektoru ($f = 0.5$) se nachází nad průměrnou hodnotou kvality (ta činí 0.38), nejlepší detektor v této práci má kvalitu $f = 0.66$. Srovnání je pouze orientační, jelikož v mé práci není provedeno rozdělení řádků textu na slova. Není nutné pro následné rozpoznávání a navíc by zpomalilo proces detekce textu.

Do budoucna bych chtěl zlepšit kvalitu detektoru. Zde by byla na zvážení možnost vlastní implementace Houghovy transformace, která by však nepracovala s připraveným binárním obrazem, nýbrž přímo s množinou potenciálních znaků. Toto by umožnilo nastavení přísnějších pravidel pro seskupování znaků do slov. Toto je myšleno tak, že by při seskupování znaků do slov byla použita i jejich velikost, barva, hodnota SWT a tak dále.

Literatura

[1] Edge detection. 2001-, [online], [cit 2015-04-20].
URL http://en.wikipedia.org/wiki/Edge_detection

[2] Tesseract (software). 2001, [online], [cit 2015-04-20].
URL http://en.wikipedia.org/wiki/Tesseract_%28software%29

[3] Datasets. 2003, [online], [cit 2015-04-20].
URL http://algoval.essex.ac.uk/icdar/Datasets.html

[4] Popis khihovny Tesseract. 2003, [online], [cit 2015-04-20].
URL https://code.google.com/p/tesseract-ocr/

[5] Bartsch, H. J.: *Matematické vzorce*, kapitola Analytická geometrie. Praha, Česká republika: Academia, Čtvrté vydání, 2006, ISBN 80-200-1448-9, str. 407.

[6] Csetverikov, D.: Basic Algorithms for Digital Image Analysis: a course. PDF prezentace.
URL http://visual.ipan.sztaki.hu

[7] Epshtein, B.; Ofek, E.; Wexler, Y.: Detecting text in natural scenes with stroke width transform. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, Březen 2010, ISSN 1063-6919, s. 2963–2970, doi:10.1109/CVPR.2010.5540041.

[8] Hlaváč, V.: Hledání hran. PDF prezentace.
URL http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/22EdgeDetectionCz.pdf

[9] Kršek, P.: *Základy počítačové grafiky*. FIT, Vysoké učení technické, Brno, Česká Republika.

[10] Laganiére, R.: *OpenCV 2 computer vision application programming cookbook*. 1, Brimmingham: Packt Publishing, quick answers to common problems vydání, 2011, ISBN 978-1-84951-324-1, s. 167–175.

[11] Marita, T.: Image Processing: Laboratory 11: Edge detection. Technická zpráva, Jud. Cluj, Romania, [online], [cit. 2015-04-07].
URL http://users.utcluj.ro/~tmarita/IPL/IPLab/PI-L11e.pdf

[12] Pietikainen, M.; Okun, O.: Edge-based method for text detection from complex document images. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, University of Oulu, FINLAND, 2001, s. 286–291,

doi:10.1109/ICDAR.2001.953800.
URL http://www.ee.oulu.fi/mvg/files/pdf/pdf_149.pdf

[13] Saha, S.; Basu, S.; Nasipuri, M.; aj.: A Hough Transform based Technique for Text Segmentation. In *Journal of Computing*, ročník 2, Únor 2010, ISSN 2151-9617, s. 134–141.
URL http://arxiv.org/pdf/1002.4048.pdf

[14] Schowengerdt, R.: *Remote Sensing: Models and Methods for Image Processing*. Elsevier Science, 2006, ISBN 9780080480589, s. 245–246.
URL https://books.google.cz/books?id=KQXNaDH0X-IC

Příloha A

Obsah CD

- /src/ – zdrojové kódy aplikace
- /src/CmakeLists.txt – specifikační soubor pro aplikaci cmake a následný překlad
- /doc/ – automaticky generovaná programová dokumentace
- /text/ – zdrojové kódy technické zprávy
- /testImg/ – testovací obrázky
- /testOut/ – dílčí výstupy detekce textu (pro kontrolu)
- /xhartm01.pdf – technická zpráva
- /Doxyfile – konfigurační soubor pro generování programové dokumentace
- /README – návod k překladu, spuštění a ovládání aplikace
- /plakat/plakat.pdf – plakát demonstrující funkčnost aplikace

Příloha B

Výstup detektoru pro část testovací sady

Obrázek B.1: Obrazová reprezentace výstupu textového detekoru.

Příloha C

Výstup detektoru pro testovací sadu – „dopravní ukazatele“

Obrázek C.1: Obrazová reprezentace výstupu textového detekoru – 1/3.

Obrázek C.2: Obrazová reprezentace výstupu textového detekoru – 2/3.

Obrázek C.3: Obrazová reprezentace výstupu textového detekoru – 3/3.