ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

DIPLOMOVÁ PRÁCE

Optimalizace procesu rozpoznávání textu pomocí Vision Builder

vedoucí práce: **Ing. Radek Holota, Ph.D.**

autor práce: **Bc. Petr Havránek**

2012

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Petr HAVRÁNEK**
Osobní číslo:	**E10N0143P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Telekomunikační a multimediální systémy**
Název tématu:	**Optimalizace procesu rozpoznávání textu pomocí Vision Builder**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

Práce se týká implementace rozpoznávání obrazu v prostředí NI Vision Builder. Cílem je studium metod OCR, jejich porovnání s již implementovanou metodou v prostředí NI Vision Builder a na základě výsledků zahrnout možnost využití vybrané metody OCR do vývojového prostředí.

1. Vysvětlete problematiku OCR.
2. Porovnejte implementaci OCR v NI Vision Builderu s jinými známými metodami OCR.
3. Detailně rozeberte chování jednotlivých metod OCR a porovnejte jejich výhody a nevýhody s využitím reálných dat.
4. Implementujte vybrané metody v prostředí NI Vision Builder.
5. Navrhněte další možnosti rozvoje OCR v NI Vision Builder.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí diplomové práce: **Ing. Radek Holota, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Konzultant diplomové práce: **Ing. Karel Matouš, Ph.D.**
Škoda Auto a. s.

Datum zadání diplomové práce: **17. října 2011**

Termín odevzdání diplomové práce: **11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 17. října 2011

Abstrakt

Téma: Optimalizace procesu rozpoznávání textu pomocí Vision Builder

Diplomová práce je zaměřena na prozkoumání metod optického rozpoznávání textu. V diplomové práci je prostudováno rozpoznávání textu v aplikaci Vision Builder od firmy National Instruments a je do něj implementována další metoda pro rozpoznávání textu z obrazu.

Klíčová slova

Vision Builder, optické rozpoznávání textu, OCR, Tesseract OCR

Abstract

Subject: Optimalisation of character recognition with Vision Builder

The thesis is focused on the exploring of the optical character recognition. In the thesis there is studied the optical character recognition in the Vision Builder from The National Instrumensts Company. There is also implemented another method of the optical character recognition to the Vision Builder.

Keywords

Vision Builder, Optical character recognition, OCR, Tesseract OCR

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 11.5.2012 Petr Havránek

…….

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce panu Ing. Radku Holotovi, Ph.D. za profesionální rady a vedení práce, dále panu Ing. Karlu Matoušovi, Ph.D. ze společnosti Škoda Auto a.s. za umožnění vypracování diplomové práce a za cenné rady.

Obsah

Seznam použitých symbolů a zkratek

OCR	Optical character recognition- anglické označení pro optické rozpoznávání textu
Sn./s	Snímků za sekundu
NI	National Instruments
VBAI	Vision Builder for Automated Inspection
VI	Virtual Instrument- označení pro kód napsaný v LabVIEW
DPI	Dots per inch- anglické označení pro rozlišení obrazu, určuje počet obrazových bodů na jeden palec
ROI	Region of interest- oblast zpracování ve Vision Builder

Seznam obrázků

Seznam tabulek

Seznam příloh

Úvod

Tato diplomová práce je zaměřena na prozkoumání možností optického rozpoznávání textu neboli OCR (optical character recognition). Optické rozpoznávání textu neboli OCR umožňuje uživateli převádět tištěné dokumenty do počítačem čtené podoby. První stopy OCR mechanizmů se datují přibližně k roku 1940. V této době vznikaly mechanické stroje pro rozpoznávání textu, které dosahovaly velmi malé úspěšnosti. S postupným rozšiřováním vzniku tištěných dokumentů, jako jsou například knihy, kopírované dokumenty, formuláře, byly tyto systémy nedostačující a bylo nutné vynalézt pokročilejší metody optického rozpoznávání textu. Další rozvoj OCR je datován do 70. let 20. století. Tyto zdokonalené systémy však umožňovaly převod obrázku pouze s jedním typem písma, které bylo pro tyto účely speciálně navrženo. První font, který byl pro tyto účely automatického rozpoznávání vynalezen, byl takzvaný font OCR - A, který vznikl v Americe za účelem strojového čtení. Tento typ písma byl velmi dobře strojově rozpoznatelný, avšak pro lidské oko působil velmi nepřirozeným dojmem. V Evropě byl pro tyto účely o několik let později vynalezen font, který nesl označení OCR - B. Pro lidské oko již tento font působil přirozeněji, ale byl o něco hůře strojově rozpoznatelný, než v Americe objevený font OCR - A. Postupným vývojem docházelo k dalšímu zlepšování OCR systémů, které dokázaly rozpoznávat již několik fontů. Dnešní moderní algoritmy dosahují již mnohem lepší úspěšnosti než historické metody optického rozpoznávání. Toto je možné díky rozvoji počítačové techniky a moderním algoritmům zpracovávání obrazu [1],[2]. Cílem této diplomové práce je prozkoumat možnosti volně dostupných a volně šiřitelných programů poskytující rozpoznávání textu, otestovat je na reálných obrázcích displejů autorádií a některý z programů implementovat do programu Vision Builder for Automated Inspection od firmy National Instrument, který již disponuje algoritmem pro OCR. Výsledkem práce by mělo být porovnání implementované metody s algoritmem již obsaženým ve Vision Builder a rozhodnutí o užitečnosti nově implementované metody.

1 Problematika optického rozpoznávání textu z obrazu

Problematika optického rozpoznávání textu (OCR) je rozsáhlý proces, který v sobě zahrnuje několik podprocesů. Podle obecného rozdělení je možné proces rozpoznávání textu rozčlenit do tří podprocesů, předzpracování obrazu, vlastní OCR algoritmus a nakonec takzvaný postprocessing neboli úprava rozpoznaného textu. Předzpracování obrazu (tzv. preprocessing) může být rozdělen na uživatelský a automatický. Uživatelské předzpracování může, ale nemusí být provedeno před vlastním rozpoznáváním textu. Pokud je však obraz nejprve předzpracován, je dosaženo lepších shod rozpoznaného textu s původní obrazovou předlohou. Předzpracováním obrazu je myšlena např. filtrace obrazu, kdy jsou z obrazu odstraněny nežádoucí artefakty, jako je například šum [3] nebo natočení obrázků tak, aby linie textu byla vodorovná. Druhým krokem předzpracování je předzpracování prováděné vlastním programem na rozpoznávání textu. Tímto předzpracováním je myšleno například korekce natočení strany, přičemž program je schopen doladit pouze mírné odchylky od vodorovné linie, binarizace a segmentace obrazu a tyto algoritmy jsou implementovány do vlastního programu. Tyto operace jsou tedy prováděny vždy před průběhem rozpoznávání textu [4]. Existuje celá řada nejrůznějších typů písma, která se od sebe liší tvarem, tloušťkou a například poměrem šířky a výšky [5]. Tento předpoklad je nutné brát v úvahu tam, kde nemáme znalost o typu použitého písma. V případě, že chceme rozpoznávat text z obrázků, které jsou určitým způsobem jednotné, např. je u nich použito jednoho typu písma, můžeme pak rozpoznávací program na tyto typy obrázků adaptovat a dosáhnout tak lepší účinnosti. V opačném případě je nutné rozpoznávací program natrénovat na celou škálu typů písma.

1.1 Interpretace obrázku

Pro zjednodušení zde bude uvažován obrázek ve stupni šedi. Digitální obraz v počítačové grafice je reprezentován mřížkou bodů, kdy každý bod nese určitou jasovou informaci. Jeden bod této mřížky je označován jako pixel neboli jeden obrazový bod. S rostoucím počtem obrazových bodů roste rozlišení obrazu, které hraje klíčovou roli v metodách rozpoznávání textu. U většiny aplikací pro rozpoznávání textu je totiž uvedeno požadované minimální rozlišení uváděné v obrazových bodech na jednotku délky, obvykle na jeden palec (2,54 cm). Dalším pojmem, který je spjat s digitálním obrázkem, je kvantizační krok. Kvantizační krok je nejmenší rozdíl mezi dvěma sousedními jasy. Obvykle je používáno 256 odstínů šedé barvy [6].

1.2 Filtrace obrazu

Jak již bylo naznačeno v kapitole 1, je filtrace vhodným nástrojem pro zlepšení účinnosti programu pro rozpoznávání textu. Každý obraz je do určité míry znehodnocen přidanými artefakty neboli šumem a míra znehodnocení do jisté míry závisí na podmínkách snímání obrazu. Obvykle se jedná o tzv. bílý šum, který má rovnoměrné spektrum. Filtraci obrazu uživatel provádí před vlastním rozpoznáváním textu. Uživatel ji může provádět pomocí nejrůznějších programů, jako je například profesionální grafický editor Adobe Photoshop nebo například profesionální vývojový program od firmy National Instruments LabVIEW nebo Vision Builder [6].

1.3 Lineární filtrace

Do skupiny lineární filtrace patří tzv. filtry typu horní propusť nebo dolní propusť. Tak jako např. u zvukových signálů je frekvence dána změnou úrovně v čase, tak analogicky v obraze je frekvence dána změnou jasu v prostoru (prostorová frekvence). Malá prostorová frekvence je při malých prostorových změnách intenzit v obraze a naopak vysoká prostorová frekvence je při prudkých změnách jasu v obrazu, kde se vyskytují hrany. Nevýhodou lineárních filtrů je to, že do obrazu zanáší další nové intenzity obrazových bodů a způsobují tak do určité míry degradaci obrazu. Filtr typu horní propusť, označován také jako gradientní operátor, slouží ke zvýraznění detailů, nevýhodou však je to, že do obrazu zavádí šum. Oproti tomu filtr typu dolní propusť, označován také jako vyhlazovací filtr, slouží k eliminaci šumu ve zpracovávaném obraze, ale také snižuje detaily a rozostřuje hrany. Lineární filtrace je v podstatě prováděna konvolucí. Pomocí konvoluce může být prováděno mnoho typů filtrace obrazu, jako například redukce šumu, detekce hran, zostření apod. Konvoluce je počítána pro každý pixel v obrázku. Matice, kterou je váhován daný pixel, je takzvané jádro konvoluce. Koeficienty matice konvolučního jádra určují typ filtru [7]. Pro vybrané operace s obrázkem jsou v nejrůznějších literaturách a na internetu uvedena vhodná konvoluční jádra [8].

1.4 Nelineární filtrace

V kapitole 1.3 byly naznačeny techniky lineární filtrace obrazu. Toto však není jediný způsob filtrace obrazu. Druhým typem filtrace je nelineární filtrace. Nelineární filtrace nepočítá novou hodnotu pixelu, ale vhodnou jasovou hodnotu vybírá z okolí bodu a nahrazuje

jí daný bod a nevytváří tak nové hodnoty intenzit pixelů. Podle způsobu volby nové jasové hodnoty je rozlišováno několik typů filtrů [9], [7]. Do skupiny nelineárních filtrů patří filtr minimum, filtr maximum a filtr medián [7].

1.4.1 Mediánový filtr

Mediánový filtr je jeden z nejvíce používaných filtrů v oblasti zpracování obrazu sloužící k vyhlazování obrazu. Mediánový filtr vybere a podle intenzity seřadí obrazové body z okolí bodu a daný bod nahradí z řady hodnot tou, která je právě uprostřed posloupnosti intenzit jasu obrazových bodů z okolí (medián). Pro jednoznačnost se volí lichý počet prvků. Na rozdíl od průměrovacího konvolučního filtru má mediánový filtr pouze minimální vliv na ostatní části obrazu, především nepůsobí příliš negativně na hrany v obraze a nezpůsobuje tak jejich rozmazávání. Pro názornost je na obrázku níže (Obr. 1) uveden příklad intenzit jasu. V takovém obrázku je žádáno nahrazení hodnoty 255 uprostřed obrázku.

64	64	64	64	64
64	64	64	64	255
64	64	255	255	64
64	64	64	255	64
64	64	64	64	255

Obr. 1 - Znázornění mediánového filtru

$$\{64+64+64+64+\underline{64}+64+255+255+255\}$$

Z výše uvedeného příkladu je patrné, že mediánový filtr je velice užitečným nástrojem pro odstranění impulsního šumu v případě, že body, ovlivněné šumem, pokrývají méně než polovinu oblasti. V případě, že je obraz šumem ovlivněn více, je nutné zvolit vetší oblast působení mediánového filtru, např. 9 x 9. V tomto případě pak ale může docházet k poškozování hran objektů v obraze [6], [7].

1.5 Segmentace obrazu

Segmentace obrazu je velmi důležitý proces v analýze a zpracování obrazových dat při rozpoznávání textu. Při segmentování obrazu dochází k oddělování určitých částí obrazu, v případě rozpoznávání textu se jedná o písmena a znaky.

1.5.1 Prahování

Jedním řešením je nastavení prahové hodnoty ručně. Tento přístup je však neuskutečnitelný v případě plně automatického počítačového vidění. Jistou alternativou by bylo prahovou hodnotu nastavit jednou a tuto hodnotu nechat jako výchozí [7]. I toto však přináší určitá omezení, a jak bude vidět v kapitole 2.4, bude toto znepříjemňovat proces rozpoznávání a co je mnohem důležitější, bude zhoršena spolehlivost algoritmu

Sofistikovanější metody určování prahové hodnoty vycházejí z histogramu obrázku. Pokud černobílý obraz má histogram obsahující dvě maxima, z nichž jedno určuje světlou barvu a druhé tmavou barvu, prahová hodnota je určena jako minimální hodnota mezi maximami [6]. Jak ruční, tak automatické metody využívá i program Vision Builder (kapitola 2.4 na straně 21).

1.6 Systém rozpoznávání textu

Rozpoznávání textu nemůže probíhat jen tak samo osobě, vždy musí existovat určitá znalost o rozpoznávaném objektu. Existuje více metod rozpoznávání objektu.

1.6.1 Neuronové sítě

Umělé neuronové sítě vznikaly již zhruba před padesáti lety [McCulloch a Pitts, 1943] [6] . V dnešní době existuje mnoho typů umělých neuronových sítí, každý typ je vhodný pro různé aplikace. Umělé neuronové sítě mají analogii v biologii, ale nedosahují takových rozměrů. Nejjednodušší neuronová síť je tvořena několika vstupy, které slouží pro vkládání příkladů, a jedním výstupem s řešením. Výstupní řešení je funkcí vstupů *v* vážených váhou *w*. S pojmem neuronové sítě je také spjat pojem aktivační neboli přenosová funkce σ, která udává funkci neuronové sítě. Matematicky by bylo možné neuronové sítě popsat rovnicí 1.1.

$$y = \sigma\left(\sum_{i=1}^{n} v_i \cdot w_i - \theta\right) \tag{1.1}$$

Kde θ je takzvaný práh neuronové sítě. Proces rozpoznávání pomocí neuronových sítí je členěn do dvou fází. První fáze zahrnuje trénování nových znaků a je označována jako adaptivní a druhá fáze je rozpoznávací též označována aktivní [10].

V praxi se vyskytují několikavrstvé neuronové sítě, které mohou mít několik vstupů a několik výstupů a několik skrytých vrstev [6]. Jedním typem je takzvaná neuronová síť se zpětným šířením. Jedná se o neuronovou síť s minimálně jednou skrytou vrstvou [6], [11]. Síť se zpětným šířením porovnává výstup sítě s očekávaným výsledkem a počítá chybu mezi nimi a snaží se o její minimalizaci změnou vah. Toto je prováděno během takzvaného trénovacího procesu. Během trénovacího procesu dochází ke změnám jednotlivých vah, které byly před započtením trénování nastaveny na určitou hodnotu. Na vstup je přivedena takzvaná trénovací sada *x* a na výstup správné řešení *d*, odpověď sítě je *y*. Chyba v případě jednoho vzoru je tedy dána $E = d - y$. Chybová funkce v případě *j* učících vzorů a *j* správných výsledků je dána sumou

$$E = \sum_{k=1}^{j} \left(d_k - y_k \right)^2 . \tag{1.2}$$

Jelikož trénovací proces probíhá opačným směrem než aktivní fáze (rozpoznávání), je síť nazvána jako umělá neuronová síť se zpětným šířením [6][11].

Výše popsaný způsob trénování není jediným způsobem nastavení správných vah neuronové sítě. Existují také takzvané samoučící neuronové sítě, které ke svému trénování nepotřebují zadávání správných hodnot na výstup. [6] Do skupiny samoučících neuronových sítí, které ke svému trénování nepotřebují vnější informaci, patří Kohonenova neuronová síť. Kohonenova samoučící umělá neuronová síť má dvě vrstvy a ke své funkci používá tzv. algoritmus shlukové analýzy. To znamená, že existuje určitá možnost nalézt nějaké vlastnosti přímo z předložených dat bez nutnosti vkládání vnější informace [10].

1.6.2 Grafové rozpoznávání

Rozpoznávání znaků pomocí grafové analogie je dalším způsobem rozpoznávání objektů. Tento postup vychází z teorie grafů. Grafem je v této teorii spojení uzlů pomocí hran, přičemž každý uzel i hrana jsou ohodnoceny. Cílem tohoto algoritmu je každému objektu, vyjádřenému graficky, přiřadit grafový ekvivalent. Aby však bylo možné jednoznačně určit rozpoznávaný objekt, je nutné, aby jemu odpovídající graf popisoval objekt jednoznačně. Tomuto požadavku se z teorie grafů říká grafový izomorfismus. Řekneme, že graf je

izomorfní, pokud existuje $F:V(G)\to V(G')\colon\{x,y\}\in E(G)\Leftrightarrow\{f(x),f(y)\}\in E(G')$, neboli existuje hrana spojující body x, y v grafu G, pak existuje hrana, která spojuje body f(x) a f(y), v grafu G´ [6], [12]. Testování izomorfismu je relativně výpočetně složitý proces. Jisté zjednodušení výpočtu přináší ohodnocení hran i uzlů v grafu. Dva izomorfní grafy musí mít stejný počet stejně ohodnocených hran i uzlů [6].

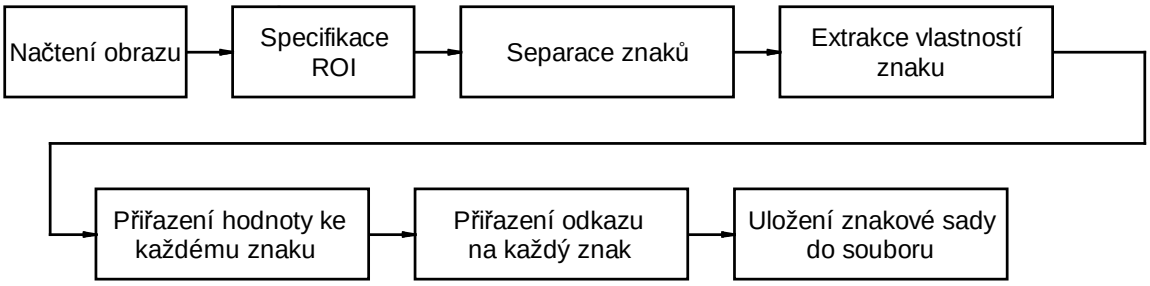
2 Program Vision Builder for Automated Inspection od firmy National Instruments

2.1 Stručná charakteristika programu

Program Vision Builder for Automated Inspection od firmy National Instruments (dále jen Vision Builder nebo VBAI) je profesionální program na zpracování obrazu a na provádění měření. Jedná se o významný nástroj při řízení kontroly jakosti výrobků, detekci vadných výrobků a jejich automatickou evidenci. Vytvořené inspekční úlohy je možné spouštět jak na počítačích s operačním systémem Windows, tak na inteligentních kamerách od firmy National Instruments. V prvním kroku při vytváření inspekcí obrazu, jak je nazýván sled operací, které jsou provedeny, je nutné načíst obraz do programu VBAI. Program VBAI je kompatibilní s mnoha zařízeními pro získávání obrazu a je možné je připojit pomocí různých rozhraní, například USB, IEEE 1394 nebo Gigabit Ethernet. Samozřejmostí je možnost provádět filtraci obrazu, prahování, je možné provádět detekce hran, rohů, měřit různé vzdálenosti atd. V neposlední řadě je také možné detekovat a identifikovat různé objekty, jako jsou například čárové kódy, QR kódy a text.

2.2 Rozpoznávání textu v programu Vision Builder for Automated Inspection

OCR algoritmus, který je používán aplikací Vision Builder pro rozpoznávání textu, je pro tyto účely speciálně vyvíjen firmou National Instruments a jeho podrobný popis není k dispozici, neboť se jedná o patentovaný algoritmus firmy. Rozepíšu tedy zde algoritmus z uživatelského pohledu. Aplikace OCR je rozdělena do dvou částí. Nejprve je nutné provést naučení neboli natrénování textu. Trénování je proces, při kterém jsou uloženy všechny znaky, které následně budou čteny. Proces trénování může být proveden pouze jednou a může tak být vytvořena rozsáhlá databáze znaků ve všech možných fontech. Čtecí procedura následně porovnává předložené znaky se znaky ze znakové sady, která byla vytvořena během procesu trénování v první fázi. Proces učení v programu VBAI probíhá dle blokového schématu na obrázku níže (Obr. 2).



Obr. 2 - Učení znaků v programu Vision Builder [13]

2.3 Specifikace oblasti působení

Prvním krokem po načtení obrázku je v trénovací proceduře nastavení oblasti působení neboli ROI (region of interest). Oblast působení v programu Vision Builder se týká jak trénování nových znaků, tak čtení znaků. Během trénování je oblast působení jen ta oblast, která obsahuje objekt, který bude trénován. Během čtení je oblastí zájmu ta oblast, která obsahuje znaky, které budou rozpoznávány. Zájmová oblast ROI by měla být vybrána záměrně jen kolem požadovaného objektu a to hned ze dvou důvodů. Prvním důvodem je zefektivnění algoritmu rozpoznávání a druhým důvodem je to, že program nedokáže rozpoznávat víceřádkové texty. Tento problém ukazuje vybraný snímek displeje autorádia (Obr. 3). Pro zlepšení dosažených výsledků je vhodné při trénování vybírat oblast kolem znaku v obrázku opatrně tak, aby neobsahoval rušivé artefakty, jako jsou například části objektů, vyskytujících se v blízkosti písma. Během čtení je možné uzavřít oblast působení jen kolem daného objektu a zmenšit tím tak analyzovanou plochu a urychlit proces rozpoznávání a zlepšit přesnost výsledku. Nemožnost rozpoznávání textu ve více řádcích není nevýhodou v případě, že je nutné číst pouze jednoduché texty nebo číselné kódy. V případě nutnosti čtení víceřádkových textů, například čtení displeje autorádia, je použití této aplikace komplikované, neboť v takovém případě je nutné, aby jednotlivé řádky byly čteny postupně.

Obr. 3 - Čtení víceřádkového textu

2.4 Segmentace znaků

Nejjednodušší metoda segmentace obrazu je založena na takzvaném prahování obrazu. Prahování obrazu je proces, kdy je černobílý obrázek převeden do takzvané binární podoby, kdy je pixelům o intenzitě jasu, která je větší než zvolený práh, přiřazena maximální intenzita a naopak pixelům, jejichž intenzita je menší, než zvolená prahová hodnota intenzity, je přiřazena nulová intenzita. Matematicky je prahování zapsáno rovnicemi (2.1) a (2.2). Tento zápis je pro prahování obrazu, kde užitečná informace je v podobě tmavých barev.

$$g(i, j) = 1 \text{ pro } f(i, j) \geq T \tag{2.1}$$

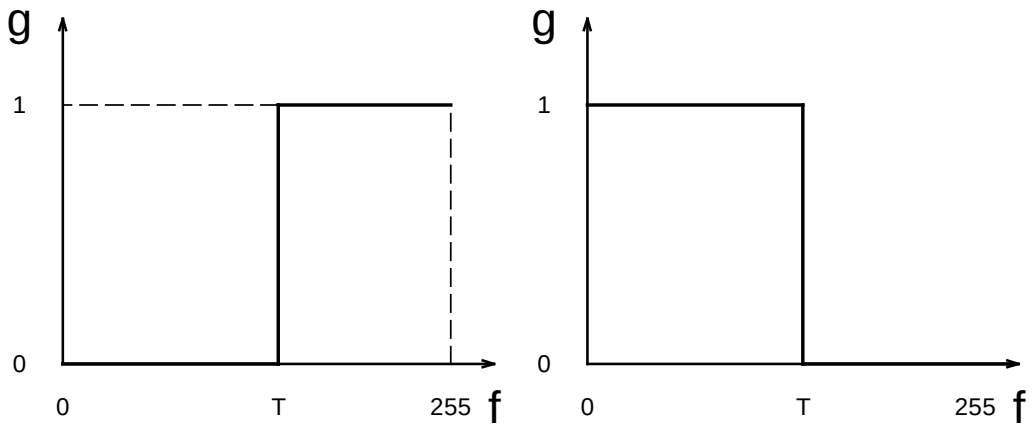
$$g(i, j) = 0 \text{ pro } f(i, j) < T \tag{2.2}$$

V případě, že by bylo nutné prahovat obraz, ve kterém by užitečná informace byla světlejší nežli tmavé pozadí, platily by rovnice (2.3) a (2.4). Toto by bylo aplikováno v případě bílého textu na tmavém pozadí.

$$g(i, j) = 1 \text{ pro } f(i, j) \leq T \tag{2.3}$$

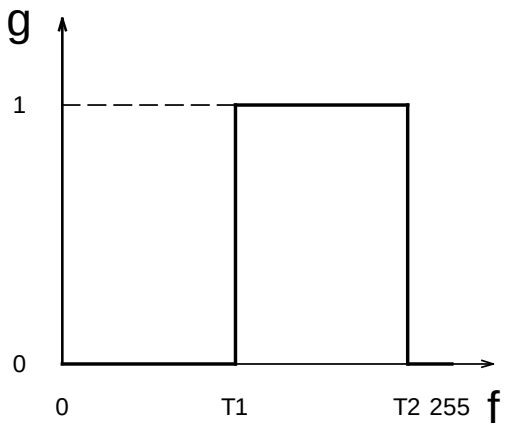
$$g(i, j) = 0 \text{ pro } f(i, j) > T \tag{2.4}$$

kde $g(i, j) = 1$ je obrazový bod popředí (text) v i-tém sloupci a j-tém řádku počítáno z levého horního rohu ve výstupním prahovaném obrázku a $g(i, j) = 0$ je obrazový bod pozadí, $f(i, j)$ je odpovídající bod původního obrazu, který je ve stupni šedi a T (z angličtiny threshold) je prahová hodnota intenzity daného obrazového bodu. Tento proces může být znázorněn pomocí obrázku (Obr. 4), který je uveden dále [7].



Obr. 4 – Grafické znázornění prahování [7]

V některých případech je používáno prahování se dvěma prahovými úrovněmi. Hodnota 1 je přiřazena intenzitám spadající do určitého intervalu a hodnota 0 je přiřazena zbylým. Toto prahování je znázorněno graficky průběhem intenzity na obrázku níže (Obr. 5).

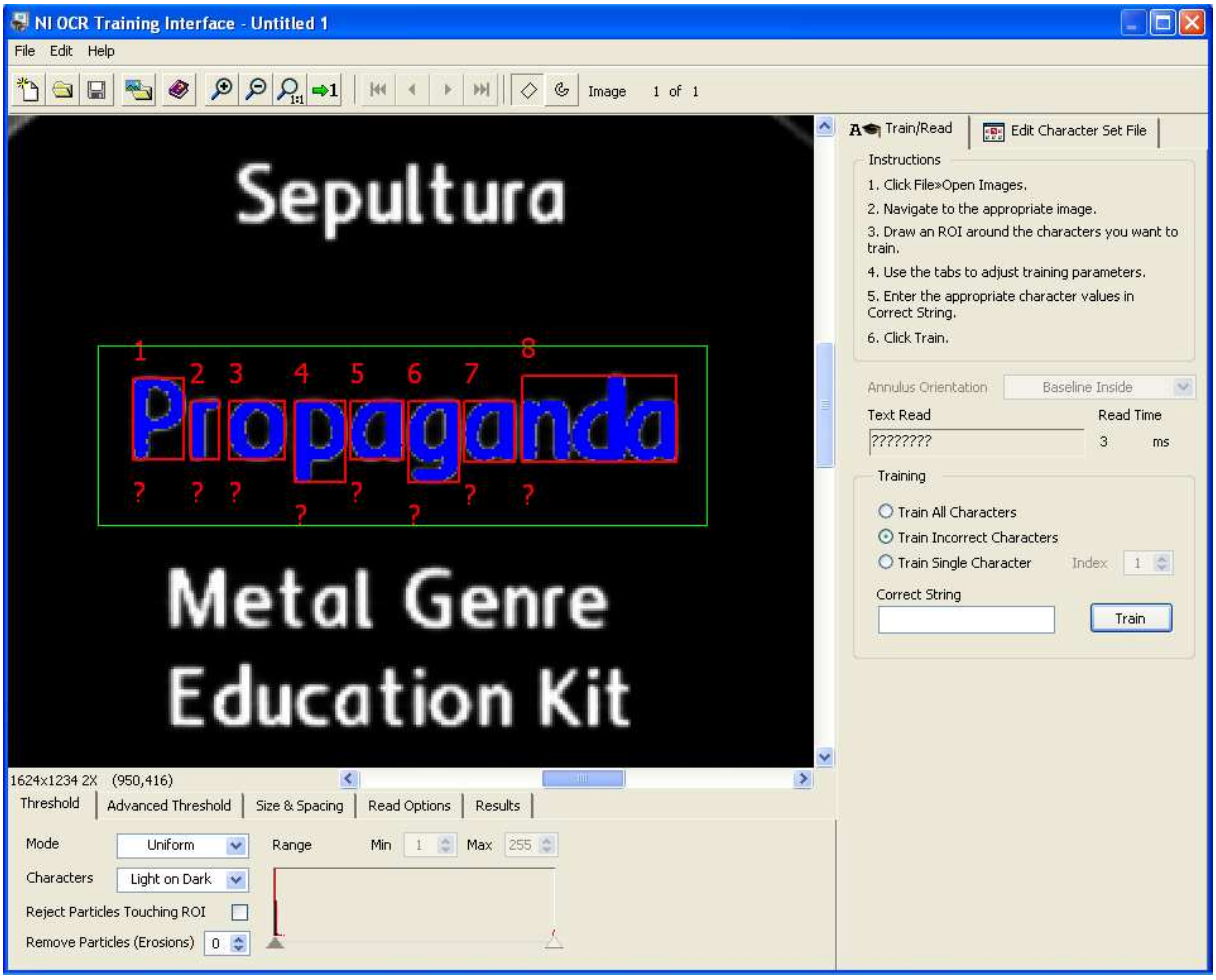


Obr. 5 - Grafické znázornění prahování [7]

Pokud jsou jednotlivé části v obrazu vzájemně odděleny a nedotýkají se, je prahování relativně jednoduchá záležitost. Klíčovým faktorem při prahování obrazu je určení takzvané prahové hodnoty. Správné zvolení prahové hodnoty rozhoduje o tom, zdali jednotlivé oddělené části v segmentovaném obrazu budou dokonale oddělené a zároveň, zda nebudou porušeny jejich okraje. Dle principu existuje více metod určování prahové hodnoty [6],[7],[14].

Program Vision Builder for Automated Inspection nabízí na výběr ze čtyř typů segmentace. Jako výchozí je nastaven uniformní mód. Uniformní metoda vypočítá jednu

prahovou hodnotu a tu použije pro extrakci pixelů z celé oblasti působení. Tato metoda je rychlá a jedná se o nejlepší způsob, pokud osvětlení v obrázku je konstantní po celé ploše oblasti působení. Tato metoda je uživatelsky pohodlná, ale v některých případech, jak ukazuje obrázek (Obr. 6), má problémy a je nutné volit jinou z nabízených metod.



Obr. 6 - Ukázka uniformního prahování v programu Vision Builder

Na obrázku nahoře (Obr. 6) je vidět, že došlo k chybnému oddělení znaků na konci slova „Propaganda“ a znaky „nda“ byly vyhodnoceny jako jeden jediný znak, což znemožňuje korektní rozpoznání. Pokud bychom se podívali na uvedený obrázek blíže, viděli bychom, že mezi těmito znaky není dostatečně velká mezera, která by zaručila správné segmentování těchto znaků při použití této automatické metody. Na obrázku níže je přiblížení inkriminovaného místa a vidíme, že postižené znaky jsou prakticky spojeny v jeden znak.

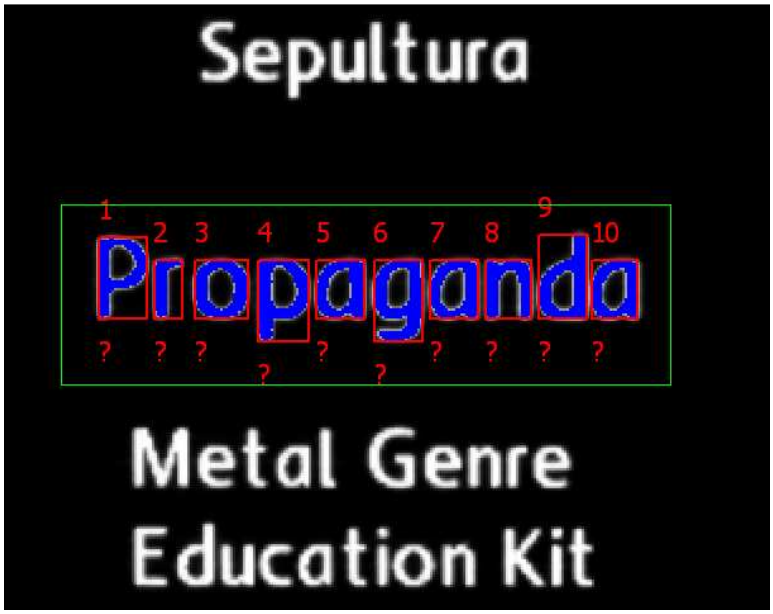
Obr. 7 - Detail textu zpracovávaného obrázku

Dalším typem segmentace je lineární metoda. Ta rozdělí oblast působení do bloků a počítá rozdílnou prahovou hodnotu pro každý vytvořený blok a lineárně interpoluje hodnotu mezi bloky. Tato metoda je vhodná pro oblasti, kde jedna strana snímku je světlejší než druhá, avšak intenzita osvětlení se mění lineárně po celé ploše působení. Při použití reálného obrázku můžeme opět vidět, že se vyskytl problém jako u předešlé metody a to, že došlo opět ke spojení některých znaků.

Obr. 8 - Ukázka uniformního prahování v programu Vision Builder

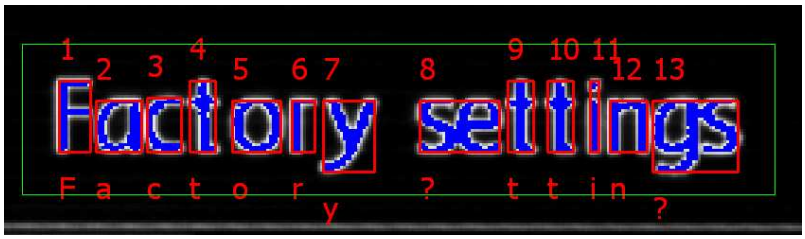
Dále je na výběr nelineární metoda, která dělí oblast působení do bloků, počítá prahovou hodnotu pro každý blok a tuto hodnotu používá k extrakci pixelů v daném bloku. I při použití této metody se vyskytuje stejný problém jako u obou výše zmíněných a to spojení znaků „da“ do jednoho jediného znaku.

Jediným možným způsobem pro úspěšnou segmentaci při použití tohoto a mnoha dalších obrázků je použití fixního módu segmentace. U fixní metody je prahová hodnota nastavena ručně uživatelem. Tato metoda zpracovává černobílé obrázky rychle, ale vyžaduje, aby osvětlení bylo jednotné po celé ploše obrazu. Tento typ segmentace je uživatelsky méně pohodlný než předešlé tři způsoby, neboť vyžaduje trošku experimentování při nastavování správné prahové hodnoty, ale jak je vidět, v některých případech je použití fixní metody nevyhnutelné. Prahová hodnota nesmí být nastavena moc malá, aby nedocházelo ke spojování znaků jako u předešlých tří automatických metod a ne příliš velká, která by sice zajistila správné oddělení všech znaků, ale také by narušila hranice jednotlivých znaků a došlo by k jejich zmenšení. Na dalším obrázku (Obr. 9), je příklad správně nastavené prahové hodnoty pro daný obrázek.

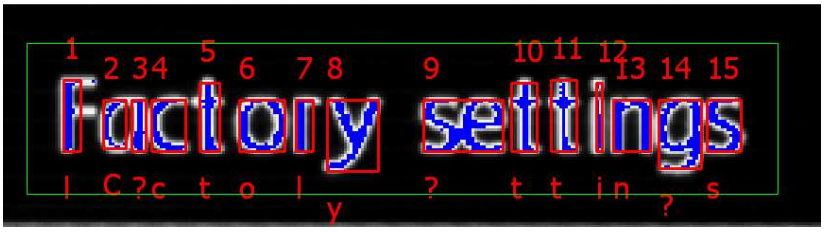


Obr. 9 - Ukázka ručního nastavení prahové hodnoty (a)

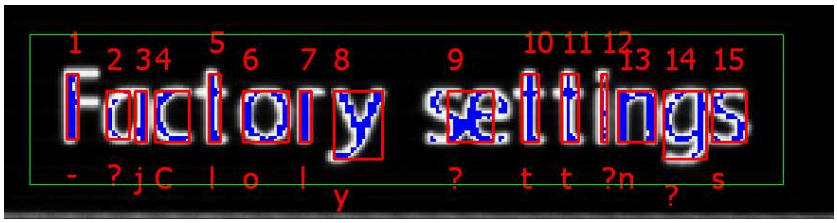
Někdy se vzájemně jednotlivé znaky natolik dotýkají, že ani ruční nastavení prahové hodnoty nezajistí správné oddělení jednotlivých znaků. Tento případ ilustruje obrázek (Obr. 10), na kterém je nastavena prahová hodnota na 234 a je vidět, že stále nedošlo k oddělení všech znaků, konkrétně znaků „se“ a „gs“ ve slově „settings“. Postupným zvyšováním prahové hodnoty dochází ke zmenšování hranic jednotlivých znaků, jako je tomu vidět na obrázku (Obr. 11) u znaku „F“, kdy je tento znak natolik poškozen, že je vyhodnocen jako „I“, avšak k separaci výše zmiňovaných znaků „se“ stále nedošlo. Dalším zvyšováním prahové hodnoty by došlo k totálnímu rozpadu dalších znaků a bylo by znemožněno rozpoznání tohoto textu (Obr. 12).



Obr. 10 - Ukázka ručního nastavení prahové hodnoty (b)



Obr. 11 - Ukázka ručního nastavení prahové hodnoty (c)



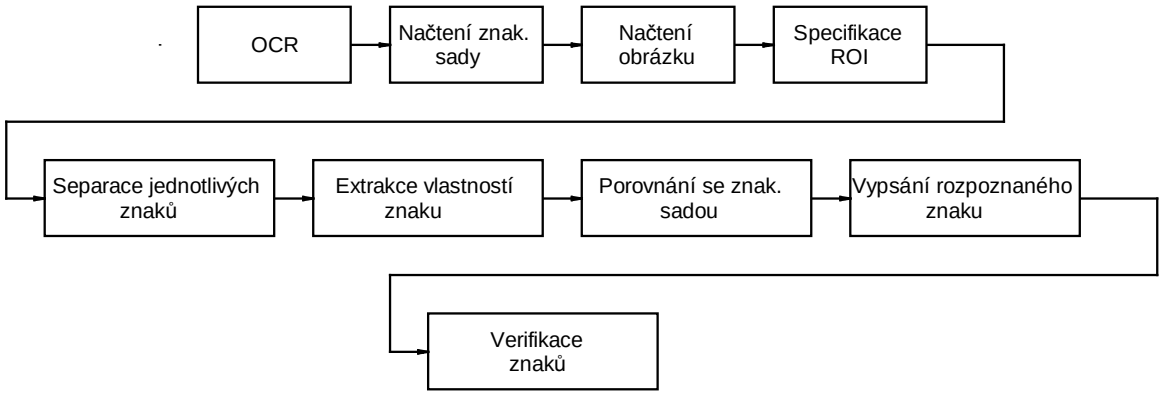
Obr. 12 – Ukázka ručního nastavení prahové hodnoty (d)

Pokud jsou již všechny znaky správně oddělené, je trénování jednoduché. Jednoduše je do položky „Correct string“ (je možné vidět na Obr. 6) napsán správný text a stisknutím tlačítka „Train“ jsou všechny znaky uloženy do databáze znaků.

OCR v programu VBAI obsahuje různé optimalizační postupy pro zlepšení účinnosti. Optimalizace umožňuje určit, zda je v daném případě požadována přesnost nebo rychlost potřebná k vypočítání prahové hodnoty.

2.5 Čtení znaků

Ve druhé fázi, při vlastním rozpoznávání naučených znaků, dochází k porovnávání předložených znaků se znaky naučenými. Uživatel musí nejprve vybrat soubor s vhodnou znakovou sadou (ta byla vytvořena během trénování), dále musí specifikovat ROI tak, jak tomu bylo i v případě učení. Uživatelské kroky odpovídají první vrstvě blokového schématu (Obr. 13). Druhá vrstva popisuje kroky, které provede program VBAI automaticky. Pokud VBAI nedokáže některý ze znaků rozpoznat, nahradí ho takzvaným substitučním znakem. Výchozí hodnota pro tento substituční znak je „?“. V nabídce je také možné nastavit takzvanou úroveň akceptovatelnosti. Jedná se o to, že každý znak je rozpoznán s určitou mírou shody. Od vývojářů programu je jako výchozí nastavena hodnota 700, ale uživatel ji může upravit v rozsahu 0 až 1000. Hodnota 0 odpovídá nulové shodě a hodnota 1000 značí, že znak je maximálně podobný znaku z předlohy. Pokud je některý znak rozpoznán s menší hodnotou shody, než jaká byla nastavena, je znak označen jako nerozpoznaný, i když třeba odpovídá správnému znaku.



Obr. 13 - Proces rozpoznávání textu v programu Vision Builder [13]

VBAI umožňuje rozpoznávat i takzvaný tečkovaný text, kdy každý znak je tvořen jednotlivými elementy. V tomto případě je tedy nutné nastavit několik dalších parametrů, z nichž hlavní dva jsou mezera mezi jednotlivými elementy a dále mezera mezi znaky. Takové znaky se však na předložených obrázcích nevyskytují a proto se toto nemusí řešit.

3 Další známé programy pro optické rozpoznávání znaků

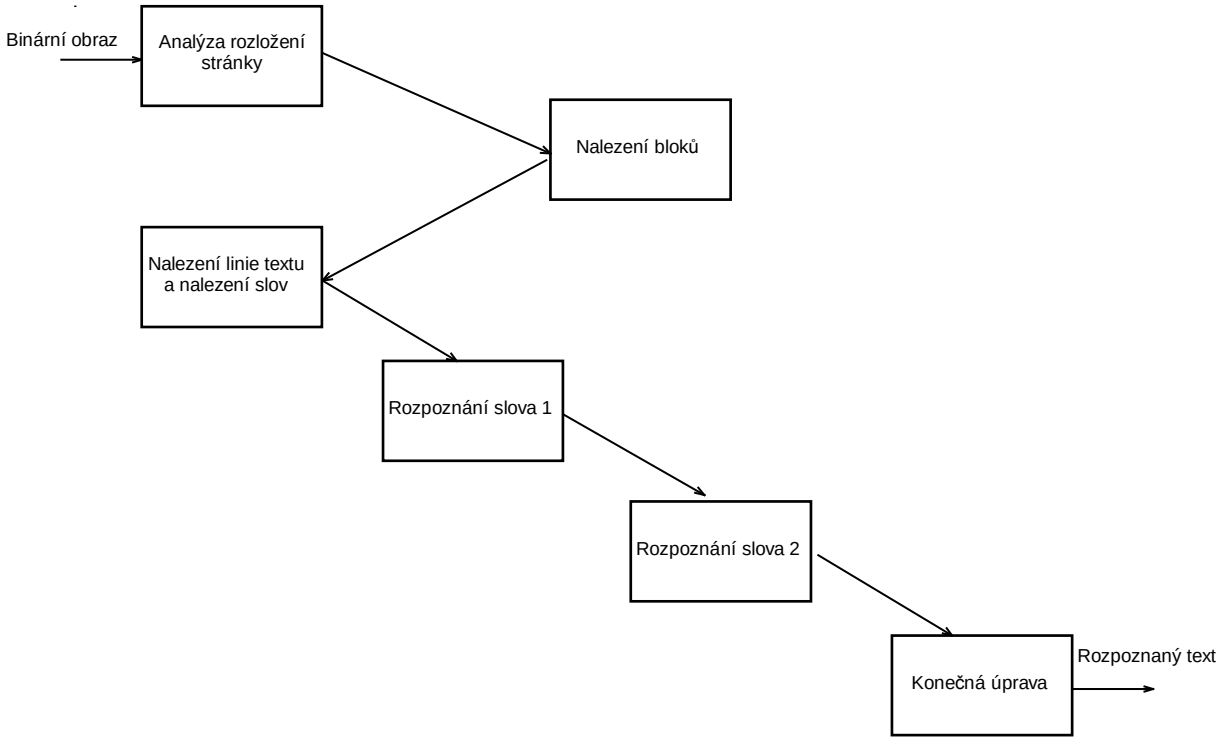
Na internetu je možné nalézt mnoho programů pro optické rozpoznávání textu. Většina těch lepších aplikací je poskytována za poplatek. Naproti tomu je možné využít programů, které jsou zdarma ke stažení a jsou tzv. opensource. Z opensource řešení jsou známé programy GOCR, FreeOCR, VietOCR. Některé z nich jsou i s grafickým uživatelským rozhraním, ale to je pro účely diplomové práce nepodstatné. Mnoho programů s grafickým uživatelským rozhraním využívají jako jádro programu Tesseract OCR. I já jsem si pro vypracování mé diplomové práce vybral program Tesseract OCR. Program Tesseract OCR jako jeden z mála vyhovoval požadavku předávání obrázku jinou cestou, než pomocí souborů a je možné jej implementovat jako DLL knihovnu.

3.1 O programu Tesseract OCR

V tomto odstavci bych se rád krátce zmínil o původu a vlastnostech programu Tesseract OCR. Program Tesseract OCR byl v letech 1985 až 1995 vyvíjen firmou Hewlett Packard. V roce 2005 byl uvolněn jako volně šiřitelný projekt. V dnešní době je vyvíjen společností Google a jak již bylo zmíněno v předešlé kapitole, na jeho základě je postaveno několik volně šiřitelných aplikací pro optické rozpoznávání textu z obrazu jako je například FreeOCR, OCRFeeder, VietOCR [15]. První verze podporovala pouze grafický formát tif a uměla rozpoznávat pouze anglicky psaný text. Pro rozpoznávání textů psaných jiným jazykem bylo nutné Tesseract OCR natrénovat. Druhá verze již podporovala několik další jazyků, mezi které patřila např. němčina, španělština, francouzština, italština. Čeština však stále i zde chyběla. Nyní je přibližně od roku 2011 uvolněna třetí verze tohoto velmi užitečného programu. Třetí verze již disponuje možností rozpoznávat česky psané texty a u této verze bylo provedeno několik dalších úprav, které jsou však pro tyto účely nepodstatné [16].

3.2 Základní princip programu Tesseract OCR

Snahou při vývoji programu bylo to, aby program Tesseract OCR byl velmi jednoduše adaptovatelný na rozpoznávání různých jazyků, mezi které patří například i čínština. Jedinou úpravou, která musí být pro rozpoznávání dalších jazyků provedena, je to, že musí být vytvořena trénovací sada znaků [17]. Program Tesseract byl navržen tak, aby bez problému dokázal rozpoznávat černý text na bílém pozadí stejně tak dobře jako bílý text na černém pozadí. Zjednodušené blokové schéma algoritmu je uvedeno na obrázku (Obr. 14).



Obr. 14 - Blokové schéma průběhu rozpoznávání textu pomocí Tesseract OCR [17]

3.3 Získání binárního obrazu

V tomto blokovém schématu je uvažován jako vstupní obraz již binární obrázek. Veškeré zpracování obrazu v programu Tesseract OCR zajišťuje volně šiřitelná knihovna leptonica. Pro binarizaci neboli prahování, kdy je z obrázku ve stupni šedi získán dvojbarevný obrázek, je používán tzv. Otsu algoritmus. Tento algoritmus přepokládá, že zpracovávaný obraz má tzv. bimodální histogram [18]. Bimodální histogram obsahuje dvě lokální maxima, kdy jedno je pro tmavou barvu a druhé je pro světlou barvu, což odpovídá tmavému textu na světlém pozadí nebo naopak světlému textu na tmavém pozadí [6]. Při binarizaci se nejprve normalizuje pozadí textu tak, že mu je nastavena bílá barva (hodnota 255). Následně je z histogramu obrázku odvozena prahová hodnota, která leží mezi dvěma lokálními maximy. Výhodou této metody binarizace je velká rychlost. Nevýhodou je to, že se musí jednat o obrázek s bimodálním histogramem, což se ale v případě rozpoznávání textu většinou jedná, neboť obraz obsahuje nějaké pozadí a popředí, kterým je rozpoznávaný text [19].

3.4 Analýza rozložení stránky pomocí konce tabulky

V dalším kroku je nutné analyzovat rozložení stránky. Pro analýzu rozložení stránky se v programu Tesseract OCR využívá detekce konce strany [17]. Tento algoritmus popsal a publikoval Ray Smith na 10. Mezinárodní konferenci analýzy dokumentů a rozpoznávání [20]. Jakýkoliv text psaný buď profesionálním publikačním systémem nebo běžným textovým editorem obsahuje tzv. netištěné znaky, kterými jsou např. mezery, tabulátory, konce řádku a odsazení. Součástí analýzy rozvržení stránky je takzvané předzpracování, kdy je provedeno oddělení sloupců s textem v případě vícesloupcového textu nebo pouze vymezení místa s textem, v případě jednosloupcového textu. Ze zpracovávaného obrázku s textem jsou také odstraněna netextová pole, jako jsou například obrázky a tabulky. Algoritmus pro detekci linie řádku umožňuje rozpoznávat šikmé řádky bez nutnosti jejich pootočení do vodorovného směru [4]. V případě rozpoznávání textu z displeje autorádií je však zajištěno, že text je nasnímán rovně, bez šikmých linií textu. Dalším krokem algoritmu, který je aplikován při rozpoznávání textu pomocí programu Tesseract OCR je nalezení bloků a horizontální linie textu. Vytvoření hranice každého znaku je provedeno tak, že středová linie mezery musí být uprostřed mezi pravou hranicí levého znaku levou hranicí pravého znaku. Hranice je vytvořena tak, aby co nejtěsněji ohraničovala každý znak, ale nesmí vzniknout rozdělení znaku tak, aby jeho část byla mimo ohraničení. V případě, že dojde ke spojení dvou nebo více znaků, Tesseract OCR se je pokusí oddělit. Předpokládejme, že pracujeme s bílým textem na černém pozadí, pak je oddělení dvou spojených znaků provedeno pomocí filtru minimum. Filtr minimum pracuje tak, že z okolí bodu vybere minimální hodnotu intenzity a nahradí jí upravovaný pixel [9],[7],[8]. V případě, že po oddělení nedojde ke zlepšení výsledku, je změna vrácena do původního stavu. Druhá možnost, jak zlepšit účinnost, je spojovat fragmentované znaky, které mohly vzniknout chybným určením prahové hodnoty. Zde je využíváno inverzní operace a je aplikován filtr maximum. Tento filtr je analogický k filtru minimum pouze s tím rozdílem, že v tomto případě je vybrána maximální hodnota z množiny sousedních bodů a tou je nahrazena hodnota upravovaného pixelu. Dojde tak k rozšíření hran a v určitém okamžiku dojde ke spojení částí znaku, které byly oddělené. V případě, že je pracováno s černým textem na bílém pozadí, je pro rozšíření hran (spojení fragmentovaných částí) použit filtr minimum a obdobně je pro oddělení dvou spojených znaků použit filtr maximum. Rozšíření hran je označováno jako dilatace a zúžení hran je označováno jako eroze. Obě tyto funkce jsou velmi užitečné, neboť to, že dojde ke spojení znaků nebo k fragmentaci jednoho znaku na dvě části, je velmi reálné a bylo možné se s tímto setkat např.

u obrázku detailu z displeje autorádia (Obr. 7). Záleží totiž na podmínkách snímání obrazu a na rozlišení snímku. Tato možnost je poměrně velkou výhodou oproti OCR ve VBAI. Jediná, ale za to relativně hodně limitující nevýhoda je ta, že tyto úpravy jsou prováděny na úkor výpočetního výkonu a zpomalují tak celý proces, což Tesseract OCR výrazně znevýhodňuje oproti programu VBAI.

3.5 Rozpoznávání

Aby mohlo dojít ke správnému rozpoznávání, je nutné provést natrénování. Program Tesseract OCR již obsahuje rozsáhlé databáze jazyků, které je schopen rozpoznávat. K natrénování dle [4] stačí pouze 20 vzorků od každého z 94 znaků pro 8 fontů a pro 4 typy písma (normální, tučné, kurzíva, tučná kurzíva). V blokovém diagramu (Obr. 14) je vidět, že rozpoznávání probíhá ve dvou částech. V první části jsou rozpoznaná slova, která jsou uvedena ve slovníku a která nejsou nejednoznačná, předána adaptivnímu klasifikátoru pro trénink. Podmínkou pro předání rozpoznaných slov adaptivnímu klasifikátoru pro trénink je to, že dané slovo musí být správně rozpoznáno, neboli každému rozpoznanému slovu je přiřazeno jakési skóre charakterizující míru shody a toto skóre musí být co největší [17]. V druhé části jsou rozpoznávána jen ta slova, která neposkytla uspokojivý výsledek z prvního rozpoznávání.

3.6 Kontextová úprava

Program Tesseract OCR disponuje možností porovnávat rozpoznaná slova se slovy, která jsou uložena ve slovníku. Porovnávání rozpoznaných slov se slovy, která jsou uložena v databázi, je založeno na procházení grafu, kde hrany spojující jednotlivé uzly jsou ohodnoceny příslušným znakem (písmenem). Požadavkem je to, aby graf vyjadřující kombinace slov byl acyklický, aby nevznikla nekonečná smyčka při procházení grafu. Pro tento termín je voleno označení DAWG (z angličtiny directed acyclic word graf) [21].

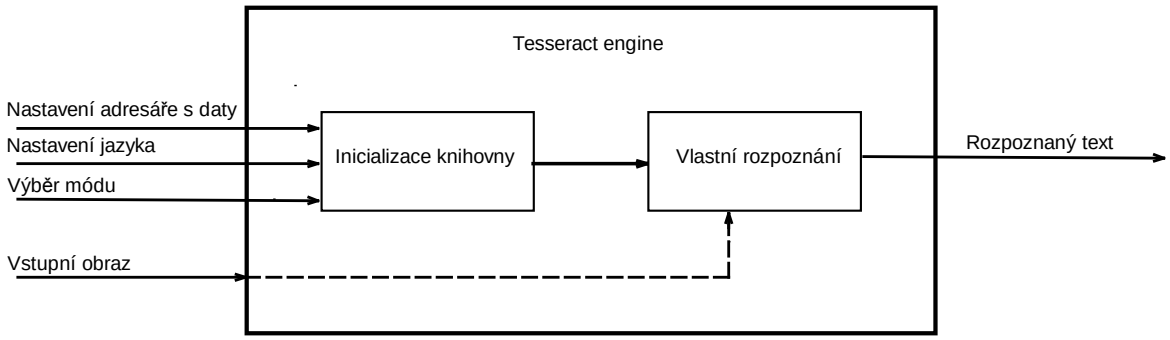
4 Vlastní implementace programu Tesseract OCR

4.1 Popis vývojového programu LabVIEW

LabVIEW je grafický programovací nástroj od firmy National Instruments. Na trhu je od roku 1986 a postupně se stal lídrem v mnoha odvětvích. Tento vývojový program je používán v mnoha oblastech vědy a výzkumu. Jeho ovládání a práce s ním je intuitivní a uživatelsky velmi komfortní. V LabVIEW mohou být vytvořeny rozsáhlé měřicí, ovládací a testovací úlohy, které vypadají jako vývojový diagram, ve kterém je zachován sled jednotlivých operací zleva doprava. Programování sofistikovaných úloh probíhá tak, že uživatel vybírá a spojuje jednotlivé funkční bloky, které vykonávají požadovanou operaci, pomocí vodičů a vytváří tak požadovanou úlohu. LabVIEW nabízí rozsáhlé možnosti v oblasti měření a řízení, je kompatibilní s tisíci nejrůznějšími zařízeními a je možné ho doplňovat o nejrůznější přídavné moduly, které jsou určeny pro určitou oblast, jako je například zpracování signálu, zpracování obrazu, získávání a zpracování dat atd. [22]

4.2 Implementace programu Tesseract OCR v programu LabVIEW

Pro implementaci knihovny Tesseract OCR bylo nutné nalézt verzi, která by splňovala všechny požadavky. Pro tyto účely byla nepoužitelná spustitelná aplikace třetí verze programu Tesseract OCR. Tato verze sice fungovala spolehlivě, ale měla jeden podstatný nedostatek, soubor se vstupním obrázkem musel být předáván jako parametr v podobě cesty k uloženému souboru. To by do značné míry omezovalo a zpomalovalo běh aplikace, neboť každý obrázek, který by byl získán pomocí kamery, by se musel uložit do souboru a výsledný rozpoznaný text by se též musel uložit do souboru a následně ze souboru načíst pro další zpracování. Tento postup předávání dat pomocí souborů nebyl tedy tím správným řešením. Po dlouhém hledání na internetu jsem způsob, jak se vyhnout předávání dat pomocí souborů, nalezl. Na domovské stránce projektu [16] jsem nalezl odkaz na .NET nadstavbu pro Tesseract OCR třetí verze. Ta je uvolněna ostatně jako celý projekt Tesseract OCR pod licencí Apache 2, která dovoluje volné šíření programu. Tento nástroj je používán jako knihovní funkce. Blokové schéma, které popisuje použití knihovny Tesseract OCR, je zobrazeno na obrázku (Obr. 15).



Obr. 15 - Implementace knihovny Tesseract OCR

Knihovna umožňuje použití různých funkcí. V mém případě stačilo využít funkci, která pracuje s obrázkem přímo přivedeným z paměti počítače, neboť každý obrázek, který je zpracováván v programu Vision Builder je uložen dočasně v paměti a je zpřístupněn pro další úpravu pomocí LabVIEW. V každém případě je nezbytně nutné nejprve provést inicializaci knihovny, v opačném případě se rozpoznání textu nezdaří, neboť následné volání dalších funkcí skončí chybou. Na obrázku (Obr. 15) je ukázán sled volání jednotlivých funkcí. Při inicializaci knihovny je nutné definovat 3 vstupní parametry. Jedná se o definování cesty k adresáři s daty programu Tesseract OCR. Tento adresář obsahuje soubory s natrénovanými daty pro každý jazyk. Dále je nutné definovat, v jakém jazyce je předložený text napsaný. To se provádí přivedením textové konstanty s předponou pro daný jazyk, např. pro angličtinu je to „eng“ a pro češtinu je to „ces“. O této předponě pro daný jazyk hovoří norma ISO 639-3 [16], ale v podstatě může být použit libovolný textový řetězec. V každém případě je ale nutné, aby předpona označující volbu jazyka korespondovala s předponou u názvů souborů obsahující znakovou sadu. V poslední řadě je definován mód, ve kterém program Tesseract OCR bude pracovat. Pokud je knihovna správně inicializována, je její výstup signalizován hodnotou pravda (T) a je možné provést samotné rozpoznávání. V případě, že funkce není nainicializována správně (vlivem nesprávné cesty k datům nebo chybně zadané předponě s jazykem) běh programu skončí chybou. K využití .NET knihovny programu Tesseract OCR slouží v prostředí LabVIEW funkce v záložce **Connectivity >> .NET**.

4.3 Implementace programu Tesseract OCR v programu Vision Builder

O programu Vision Builder již bylo pojednáno v kapitole 2.1 na straně 19. Pokud bych na tomto místě měl shrnout vlastnosti programu Vision Builder, tak se jedná o velmi užitečný nástroj pro počítačové vidění, provádění měření v obrazu atd. Mimo celou řadu funkcí, které

jsou již v programu implementovány, je možné tuto škálu rozšířit o vlastní uživatelský krok. K tomuto účelu je nutné mít nainstalovaný vývojový modul NI Vision Builder for Automated Inspection Development Toolkit. Kromě tohoto modulu je nutné mít nainstalovaný samotný Vision Builder a LabVIEW nižší verze než je verze programu Vision Builder, například Vision Builder 2011 a LabVIEW 2010. Samotná tvorba uživatelského kroku pro Vision Builder je prováděna v prostředí LabVIEW. K vytvoření uživatelského kroku slouží několik typů šablon podle funkce, která je od vlastního kroku očekávána. Může být vytvořen uživatelský krok plnící funkci získání obrazu z některého zařízení, které není ve Vision Builder standardně podporováno, dále je možné vytvořit uživatelský krok, který provádí jednoduché zpracování obrázku, jako například inverzi, filtraci atd. a dále uživatelský krok, který provádí zpracování obrazu a poskytuje určité typy výsledků pro další zpracování. Pro účely implementace programu Tesseract OCR jsem zvolil uživatelský krok, který poskytuje výsledky pro další zpracování, neboť s rozpoznaným textem je nutné dále pracovat. Tento krok však nebyl použit včetně všech detailů, ale byl do jisté míry modifikován. Po vytvoření šablony je možné ji upravovat tak, aby plnila požadovanou funkci. K tomuto účelu slouží hlavně dvě části, uživatelské rozhraní (User Interface) a uživatelské programování (User Programming), které nejsou v podstatě nic jiného, než část kódu napsaná v jazyce LabVIEW, neboli VI. V části pro uživatelské rozhraní je upravován vzhled rozhraní uživatelského kroku, který je zobrazen při vkládání uživatelského kroku při programování úloh v programu NI Vision Builder a slouží k nastavení všech potřebných parametrů vlastního kroku. Vlastní jádro uživatelského kroku je programováno v části pro uživatelské programování (User Programming). Tato část je rozdělena do tří podčástí, při čemž každá slouží ke svému specifickému účelu.

4.3.1 Součásti uživatelského kroku

Pro jednodušší orientaci při vytváření uživatelského kroku je program rozdělen do několika částí, z nichž některé jsou zpřístupněné k editaci uživatelem a některé jsou uzamčené. Pro vytvoření uživatelského kroku, po kterém chceme, aby plnil námi požadovanou funkci, slouží hlavně tyto části:

All Vis.vi slouží pro přehledné zobrazení všech podčástí.

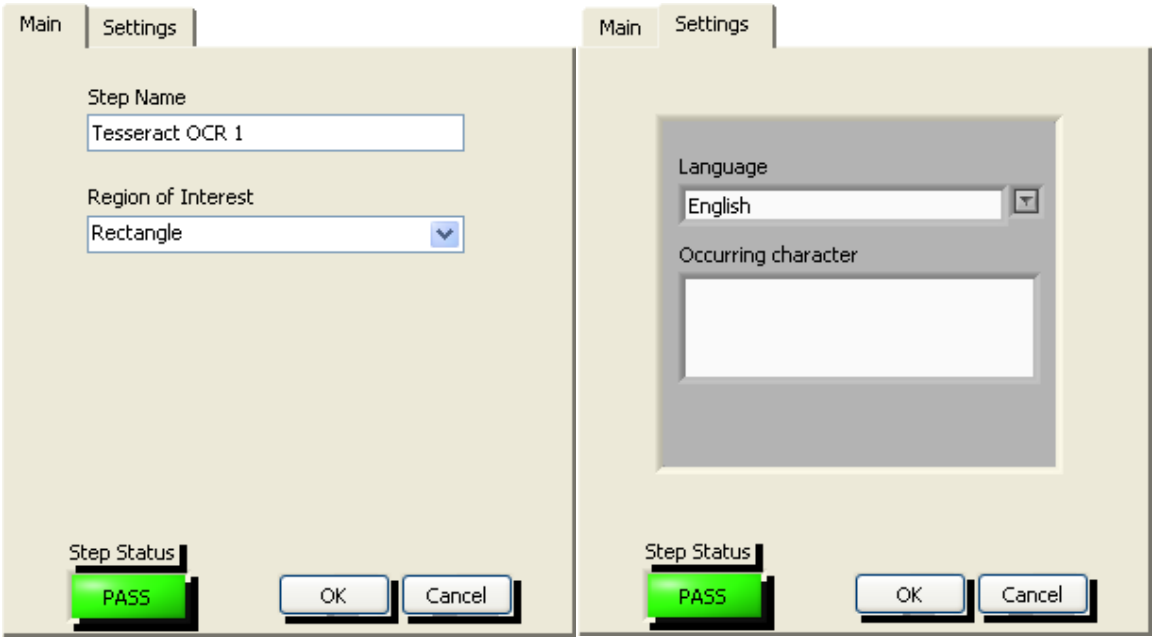
Init Globals.vi slouží pro nastavení základních vlastností. Je zde uvedeno jméno kroku a jeho popis tak, jak bude zobrazován ve Vision Builder v paletě nástrojů. Dále je zde možné

nastavit podporované typy obrázků, tvar regionu působení včetně výchozí hodnoty a také to, ve které záložce v paletě nástrojů bude tento krok umístěn.

Parameters.ctl slouží pro nadefinování obrazovky, která bude nabídnuta uživateli při vkládání uživatelského kroku. V podstatě se jedná o část uživatelského rozhraní, které je na straně 36 na obrázku uživatelského rozhraní (Obr. 17) a je možné si všimnout, že se jedná o zcela totožnou část. Zde je toto definováno proto, aby bylo vše potřebné na jednom místě, neboť s uživatelským menu, které je na obrázku (Obr. 16), je pracováno na více místech celého programu. V podstatě nadefinování všech vstupních proměnných v souboru Parameters.ctl zajistí zpřehlednění celého programu.

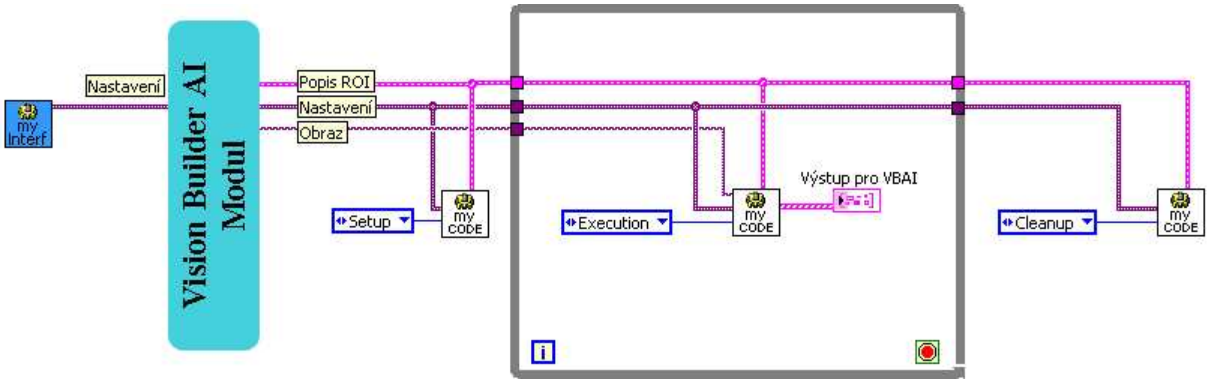
Obr. 16 - Část uživatelského rozhraní

User Interface.vi je určeno k nastavení uživatelského rozhraní, které bude zobrazováno při práci s uživatelským krokem. Veškerá nastavení v záložce „Settings“ jsou dále používána a na obrázku (Obr. 18) jsou veškeré parametry ze záložky „Settings“ vedeny silným fialovým vodičem s označením „Nastavení“.



Obr. 17 - Uživatelské rozhraní vytvořeného kroku

User Programming.vi slouží pro vytvoření celého jádra problému. Právě zde je naprogramováno to, co daný uživatelský krok bude vykonávat. Podrobnějším popisem programu se budu zabývat v následujících kapitolách 4.3.2 až 4.3.4. Na obrázku (Obr. 18) níže je možné vidět členění části programu pro uživatelské programování.

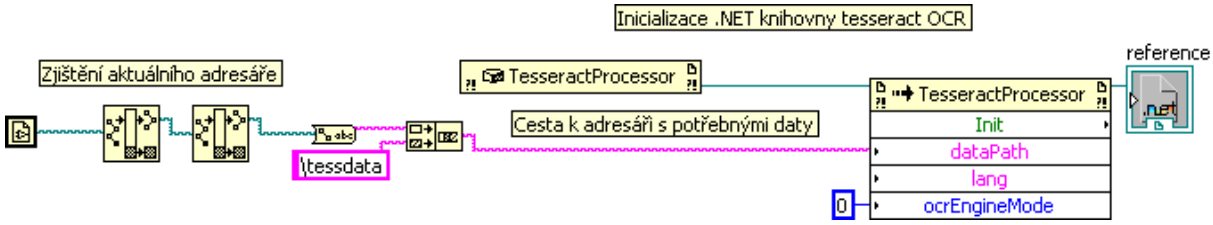


Obr. 18 - Blokové schéma programu uživatelského kroku [23]

Na obrázku (Obr. 18) výše je možné si všimnout vstupu a výstupů, které jsou označeny titulkem. Vstup do VBAI „Nastavení“ obsahuje veškeré informace, které uživatel zadá v uživatelském rozhraní (Obr. 17 vpravo). Výstup „Obraz“ tvoří odkaz na obrázek, který má být zpracováván. Ten je automaticky přiveden z programu Vision Builder z předchozího kroku tak, jak je možné vidět na obrázku s inspekčním řetězcem (Obr. 25 na straně 40). A nakonec výstup „Popis ROI“ obsahuje informace o hranici objektu, neboli vymezuje pouze tu část, která má být zpracována.

4.3.2 Inicializace knihovny

V první části je vložena část kódu, která slouží k inicializaci knihovny. Tato část se provede pouze jednou a to právě tehdy, když je v prostředí Vision Builder vložen vytvořený uživatelský krok do programovacího okna nebo když je již vložený krok měněn např. ve smyslu definování jiného ROI. Tato část je označena pomocí modré konstanty „Setup“ (Obr. 18). V této části kódu dojde k načtení knihovny do paměti. Kód, který zajišťuje tuto operaci, je zobrazen na obrázku níže (Obr. 19).



Obr. 19 – Část programu pro inicializaci knihovny

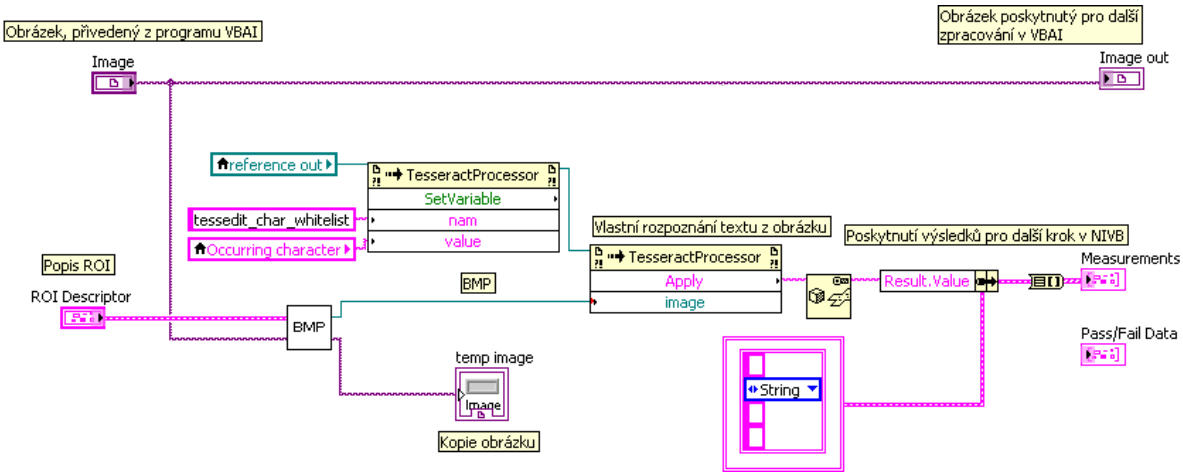
Zjištění aktuálního adresáře má za úkol zjistit, kde se daný program nachází. Ve stejném adresáři, jsou totiž data, obsahující soubory nutné pro rozpoznávání. Jedná se o data s natrénovanými znaky atd. Následně je připojena řetězcová konstanta \tessdata a je vytvořena kompletní cesta a ta je jako řetězec (v obrázku označen růžovou barvou) přivedena do bloku zajišťující inicializaci knihovny na vstup „dataPath“. Do vstupu „lang“ je přiveden textový řetězec odpovídající zvolenému jazyku. Ten se vybírá v rozhraní uživatelského kroku, které je zobrazeno na obrázku (Obr. 17). Výstupem z inicializační části je reference, která je přivedena do části kódu, která realizuje vlastní rozpoznání textu z přivedeného obrázku.

4.3.3 Vlastní rozpoznání textu

Další část je označena jako „Execution“ a je umístěna v šedě ohraničeném prostoru, který tvoří cyklus „while“. Uživatelem naprogramovaný kód, je volán vždy, když probíhá inspekce pro nový obrázek a tvoří výkonné jádro uživatelského kroku.

Uživatelský krok má standardně jeden vstup v podobě obrázku a několik výstupů. Na obrázku (Obr. 20) níže je vyobrazen celý program. Pro přehlednost a lepší orientaci jsem zde místo části programu použil blok „BMP“, jehož schéma je rozkreslené na obrázku (

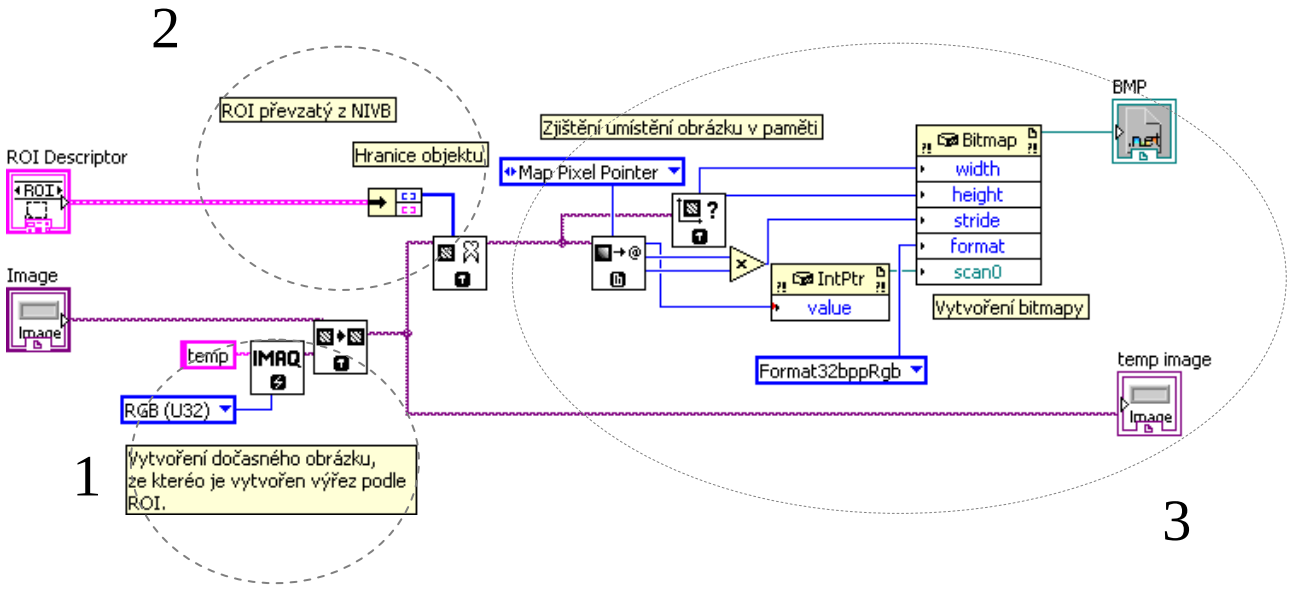
Obr. 21).



Obr. 20 - Program pro rozpoznání textu

Část programu v bloku BMP je ukázán na obrázku níže (

Obr. 21). Jednotlivé části, tak jak jdou za sebou, jsou označeny čísly 1- 3.

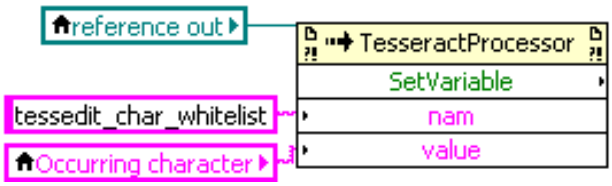


Obr. 21 - Podprogram pro úpravu obrázku

V první časti (1) je vytvořena kopie obrázku (Image), který je přiveden z předcházejícího kroku v inspekčním řetězci (Obr. 25). Kopii je nutné vytvořit proto, že pokud není požadována změna obrázku, musí být tentýž obraz beze změny přístupný pro další krok v inspekčním řetězci. Druhá část (2) vytvoří hranici obdélníku, který je dán výběrem ROI ve Vision Builder a vytvoří tak výřez z kopie přivedeného obrázku. Z tohoto je vidět, že je nutné vytvořit kopii obrázku, jinak by pro další krok ve VBAI byl přístupný pouze výřez podle ROI. Jelikož zpracovávaný obrázek je fyzicky uložen v paměti je nutné zjistit umístění

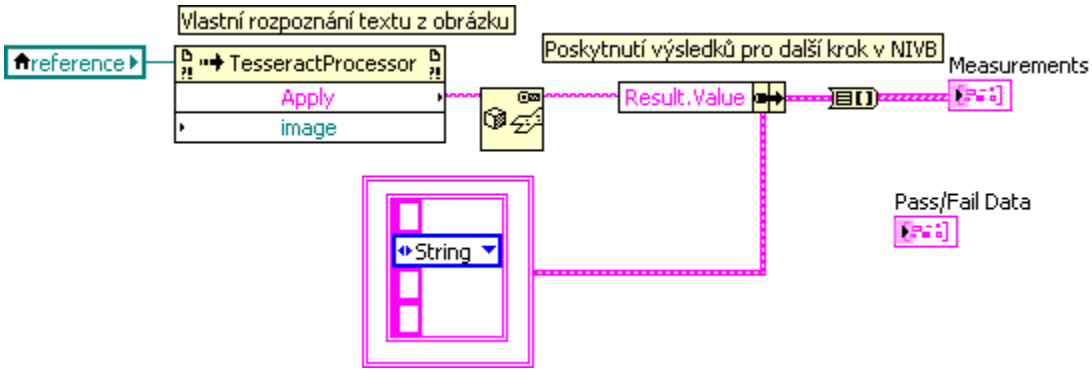
v paměti. To má za úkol třetí část (3), která zjistí ukazatel na první pixel obrázku a předá ho jako parametr knihovní funkci, která má za úkol převést obrázek z paměti na bitmapu.

Část kódu na Obr. 22 slouží pro definování všech znaků, které se mohou vyskytovat v rozpoznávaném textu a je tak možné zlepšit přesnost rozpoznávání. Tyto znaky jsou definovány v uživatelském rozhraní, které je možné vidět na obrázku na straně 36 (Obr. 17).



Obr. 22 - Definování vyskytujících se znaků

Třetí část programu, která je na obrázku (Obr. 23), obsahuje práci s knihovnou a upravuje výsledky do podoby, která umožňuje poskytnutí rozpoznaného textu pro další zpracování v programu Vision Builder.



Obr. 23 - Vlastní rozpoznání a poskytnutí výsledků

Blok reference tvoří odkaz na výstup z části, která tvořila inicializaci. Tato část byla popsána na obrázku výše (Obr. 19). Pokud by nebyl zaveden odkaz z inicializační funkce do rozpoznávací funkce, neproběhlo by správné rozpoznání, neboť by běh programu skončil chybou. Vstup označený „image“ v části, která je pojmenována „vlastní rozpoznání textu z obrázku“, slouží pro přivedení bitmapy z výstupu třetí části (3) pro úpravu obrázku (Obr. 21 na straně 38). Výstup funkce obsahuje rozpoznaný text a je tvořen řetězcem značeným růžovou barvou. Aby bylo možné rozpoznaný text použít v programu Vision Builder, je nutné provést některé úpravy. V první řadě je proveden převod řetězce do vhodného tvaru. Jelikož výsledky, ve schématu označeny ikonou „Measurements“, jsou Vision Builderu poskytována

jako pole hodnot, musí se toto pole vytvořit. K vytvoření pole dojde tak, že se nejprve vytvoří tzv. klastr, což je vlastně seskupení různých datových typů (např. řetězec a číslo) a teprve poté je tento klastr převeden na pole obsahující výsledky, které jsou přístupné k dalšímu zpracování.

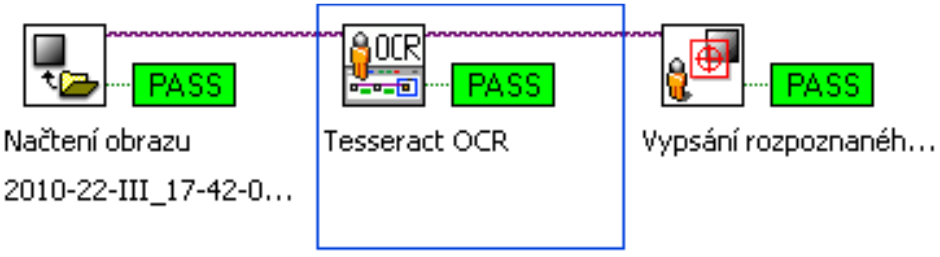
4.3.4 Odstranění dočasných souborů

V poslední části (Cleanup na Obr. 18 na straně 36) je nutné provést takzvaný „úklid“. K tomuto slouží část kódu, která vymaže všechny dočasné soubory, uvolní paměť atd. Tento kód k tomuto určený je volán jen tehdy, pokud je uživatelský krok odstraněn z řetězce kroků nebo když je zavřen program Vision Builder. V mém případě dojde k uvolnění místa v paměti, které bylo rezervováno pro načtenou knihovnu a dále dojde k odstranění kopie obrázku, která musela být vytvořena pro další zpracování. Výše uvedené kroky jsou zachyceny na obrázku níže, který ukazuje, jak je tvořena tato část.

Obr. 24 - Část kódu pro odstranění dočasných souborů

Pro smazání dočasně vytvořeného obrázku je v LabVIEW jednoduchý nástroj, který má jako vstupní parametr pouze odkaz na příslušný obrázek a uvolnění paměti je docíleno zavoláním funkce end().

Po vytvoření a uložení uživatelského kroku je možné ho vložit do inspekčního řetězce v programu Vision Builder. Jednoduchý řetězec, který provede načtení obrázku, rozpoznání pomocí Tesseract OCR a vypsání rozpoznaného textu je vyobrazen na obrázku níže (Obr. 25).

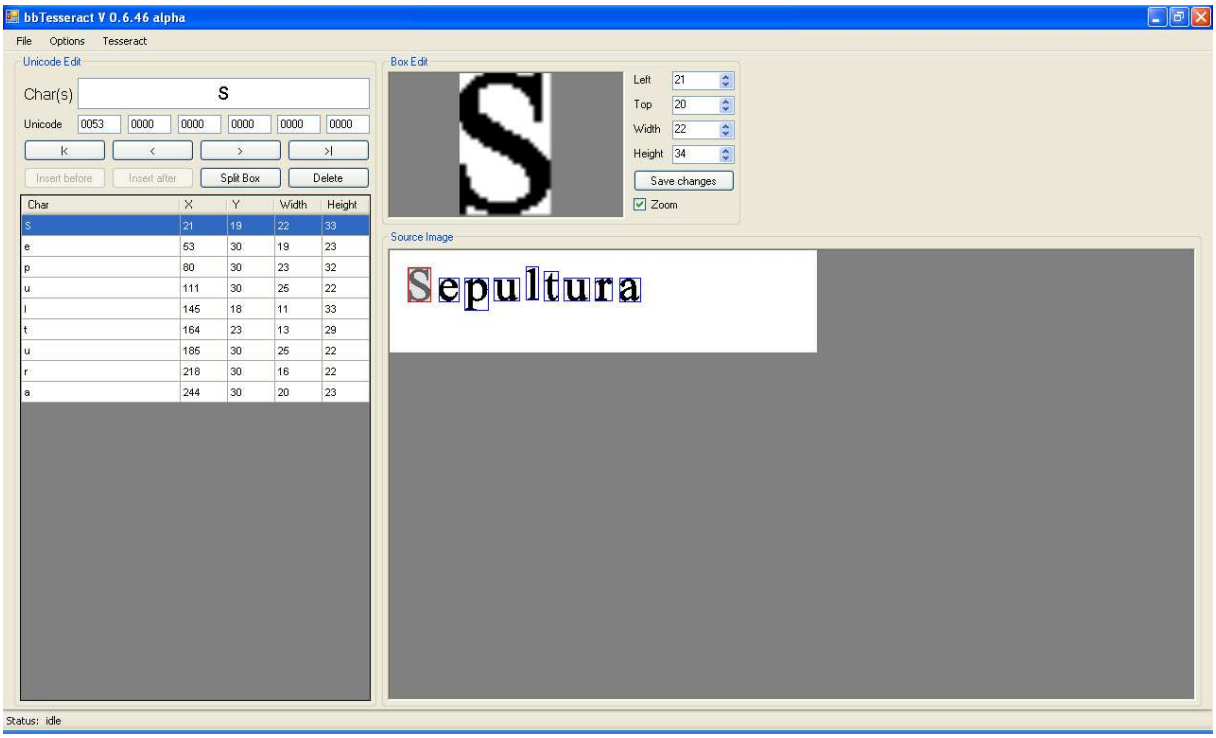


Obr. 25 - Část inspekčního řetězce s vytvořeným uživatelským krokem

4.4 Vytvoření znakové sady

V kapitole o inicializaci knihovny bylo uvedeno, že musí být definován správný adresář s daty, potřebnými k rozpoznávání textu. Těmito daty jsou myšleny soubory, které musí být vytvořeny při procesu trénování a které obsahují znakovou sadu. Postup pro trénování nových znaků se liší s verzí programu. V mé práci jsem pro implementaci použil program Tesseract OCR verze 3.01. Postup pro trénování nových znaků je velmi podrobně popsán na stránkách vývojářů [24] a i já jsem veškeré informace použité v následujícím odstavci čerpal právě z této internetové stránky.

Trénink nových znaků začíná procesem vytvoření sady trénovacích obrázků, která by obsahovala všechny možné znaky, které se budou vyskytovat na obrázcích, ze kterých bude následně text rozpoznáván. V případě, že je požadováno natrénování více typů písma, je důležité, aby na každém trénovacím obrázku byl pouze jeden jediný font. To platí i pro případ, když se jedná o např. tučné písmo téhož fontu. Pro účely rozpoznávání textu z displejů autorádií postačí natrénovat jediný font, avšak je vhodné na jednom obrázku použít několik různých velikostí písma. Pokud je již vytvořen potřebný obrázek, je nutné vytvořit tzv. boxfile, neboli textový soubor, který popisuje hranice každého znaku na obrázku. Vytvoření boxfile je nejnáročnější a nejzdlouhavější operace při trénování nových znaků. Existuje mnoho užitečných nástrojů pro snazší vytvoření tohoto souboru. Já jsem použil program bbTesseract V 0.6.46 alpha. Program je volně ke stažení pod licencí Apache 2. Tento program také umožňuje vytvořit trénovací obrázek. V podstatě je jen nutné vytvořit textový soubor a program bbTesseract z něj vytvoří obrázek, na kterém je poté prováděno trénování. Na obrázku níže je snímek programu, kde je uveden pouze příklad, který by samozřejmě pro plnohodnotné trénování nestačil.



Obr. 26 - Program pro vytvoření boxfilu

Nejzdlouhavější je korekce všech hranic znaků. V grafickém rozhraní programu je to velice snadné. Pokud jsou poupraveny hranice všech znaků, je možné vytvořit textový soubor s popsaným umístěním všech znaků. Výpis z textového souboru je uveden níže.

S 21 55 43 88
e 53 54 72 77
p 80 45 103 77
u 111 55 136 77
l 145 56 156 89
t 164 55 177 84
u 185 55 210 77
r 218 55 234 77
a 244 54 264 77

Jednotlivá čísla udávají postupně levou, spodní (počítáno zespoda), pravou a horní hranici (počítáno zespoda). Pokud je vytvořený boxfile, je nutné pokračovat s dalšími procedurami, ty jsou realizovány v příkazovém řádku. Program Tesseract OCR je spuštěn v tréninkovém režimu. K tomuto slouží příkaz

tesseract ces.VWThesis_SansRegular.tif ces.VWThesis_SansRegular.box nobatch box.train

kde ces.VWThesis_SansRegular.tif je trénovací obrázek, ces.VWThesis_SansRegular.box je vytvořený boxfile nobatch a box.train jsou soubory, které

jsou součástí instalace programu Tesseract OCR (adresář Uživatelský krok\tessdata\configs a Uživatelský krok\tessdata\tessconfigs na CD). Výstupem je soubor *.tr, který popisuje všechny znaky. Dále je nutné vytvořit soubor, který obsahuje popis znaků v Unicode. K tomuto slouží příkaz:

unicharset_extractor ces.VWThesis_SansRegular.box

Dále je nutné vytvořit soubor s prototypy písma. K tomuto slouží dvojice příkazů:

mftraining -F font_properties.txt -U ces.unicharset ces.VWThesis_SansRegular.tr
cntraining ces.VWThesis_SansRegular.tr

Font_properties.txt je soubor který obsahuje seznam všech použitých fontů, v mém případě pouze jeden. Je nutné dát pozor na to, aby jména fontů v tomto souboru korespondovala se jmény *.tr souborů, jinak běh programu skončí chybou.

Tímto je vytvořeno několik souborů:

ces.inttemp
ces.microfeat
ces.normproto
ces.pffmtable
ces.unicharset

Nyní je konečně možné vytvořit databázi znaků sloučení všech výše uvedených souborů. K tomu slouží příkaz combine_tessdata ces., kde ces. je označení pro vytvořený jazyk (čeština) dle normy ISO 639. Všechny vytvořené soubory je nyní potřeba překopírovat do adresáře …\tessdata. Všechny potřebné aplikace (tesseract.exe, unicharset_extractor.exe, mftraining.exe, cntraining.exe a combine_tessdata.exe) jsou v adresáři Uživatelský krok na CD.

5 Testování uživatelského kroku Tesseract OCR

5.1 Zhodnocení výsledků

Po vložení uživatelského kroku Tesseract OCR (Obr. 25) je možné provést rozpoznávání textu jiným způsobem, než tím, který je již implementován. Na následujících několika obrázcích (Obr. 27 na straně 44 a Obrázková příloha 1 až Obrázková příloha 6 v přílohách) je ukázáno rozpoznání textu pomocí Tesseract OCR. Zde je uveden na ukázku pouze výběr několika málo obrázků, avšak testování bylo provedeno na třiceti obrázcích, ty jsou přiloženy CD v adresáři Test\Obrázky. Výsledky testování budou shrnuty níže v tabulce (Tabulka 1). Jak je vidět na zmiňovaných obrázcích (Obr. 27 na straně 44 a Obrázková příloha 1 až Obrázková příloha 6 v přílohách), umožňuje mnou vytvořený uživatelský krok Tesseract OCR rozpoznávat i víceřádkové texty, což OCR ve Vision Builder neumožňuje. Další nespornou výhodou je to, že není nutné nastavovat parametry prahování, jako tomu bylo u již implementovaného OCR, ale program Tesseract OCR tak učiní sám. To ušetří mnoho práce a zajistí spolehlivější rozpoznání textu, neboť implementovaný OCR algoritmus od National Instruments vyžadoval u většiny použitých obrázků (Obr. 9 až Obr. 12) ruční nastavení prahové hodnoty a ta se u spousty obrázků lišila, což znemožňovalo plně automatické rozpoznávání. Záměrně jsem zde vybral tytéž obrázky (Obr. 6 až Obr. 12), na kterých bylo demonstrováno rozpoznávání pomocí Vision Builder. Rozpoznaný text je uveden u příslušného textového pole a je odlišen kurzívou.

Obr. 27 - Rozpoznávání textu pomocí uživatelského kroku (a)

Pro porovnání OCR algoritmu od NI a OCR poskytnutým programem Tesseract OCR jsem provedl testování úspěšnosti rozpoznaných znaků pro dvě metody segmentace u Vision Builder a dále jsem provedl ověření výsledků, jaké poskytuje uživatelský krok na základě

Tesseract OCR. Výše je na obrázcích (Obr. 6 až Obr. 12 na straně 23 až 26) ukázáno rozpoznávání ve Vision Builder a na obrázcích (Obr. 27 na straně 44 a Obrázková příloha 1 až Obrázková příloha 6 v přílohách) rozpoznávání textu pomocí Tesseract OCR. Jedná se pouze o několik obrázků, avšak testování jsem provedl na třiceti obrázcích.

Testovací obrázky byly pořízeny kamerou Basler scA1600-14fc. Jedná se o kameru připojitelnou přes rozhraní FireWire, rozlišení kamery je 1628 x 1236 a dosahuje rychlosti snímání 14 obrázků/s [25]. Rozlišení obrázků, na kterých jsem provedl testování, je 96 DPI, což se velmi nepříznivě projevilo na úspěšnosti rozpoznání, kdy, jak již bylo popsáno v kapitole 2.4 na straně 21, měl na obrázcích (Obr. 6 až Obr. 12) program Vision Builder problémy se správnou segmentací jednotlivých znaků. Implementovaný uživatelský krok Tesseract OCR takové problémy se segmentací znaků neměl, neboť jak již bylo uvedeno v kapitole 3.4, programu Tesseract OCR tolik nevadí vzájemně dotýkající se znaky. Výsledky jsou shrnuty v tabulce níže (Tabulka 1), která vychází z počtu chybně rozpoznaných znaků (Tabulková příloha 1).

Počet obrázků	**30**
Celkový počet znaků	**1911**
Úspěšnost při uniformním módu (VBAI)	**48,09 %**
Úspěšnost při fixním módu (VBAI)	**86,24 %**
Úspěšnost Tessarct OCR	**91,16 %**
Úspěšnost Tessarct OCR při nastavení vyskytujících se znaků	**91,78 %**

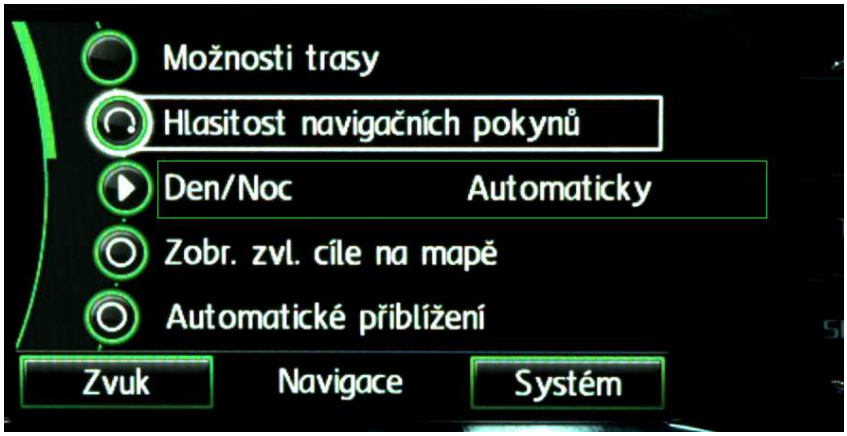
Tabulka 1- Porovnání úspěšnosti rozpoznávání

Úspěšnost při uniformním typu segmentace byla velmi neuspokojivá. Úspěšnost rozpoznávání dosahovala pouze 48,09 %. Hlavní problémy pramenily z toho, že program nedokázal automaticky oddělit jednotlivé znaky (Obr. 7), což přikládám hlavně malému rozlišení obrázků. Bohužel jsem neměl k dispozici obrázky ve vyšší kvalitě, abych potvrdil, případně vyvrátil toto tvrzení, ale dá se předpokládat, že to bylo způsobeno právě tím. Mnohem lepší úspěšnosti bylo dosaženo při nastavení prahové hodnoty ručně, tzv. fixní mód. Tato hodnota byla zvolena experimentálně 225. Větší hodnota by způsobila rozpad jednotlivých znaků na více částí. S tímto nastavením bylo dosaženo úspěšnosti 86,24 %, což ale stále nebylo dostačující. I zde byla naprostá většina chybně rozpoznaných znaků špatně segmentována.

Největší úspěšnosti na testovacích obrázcích bylo dosaženo právě s vytvořeným krokem na základě programu Tesseract OCR. Přestože dle [26] je doporučené rozlišení obrázků

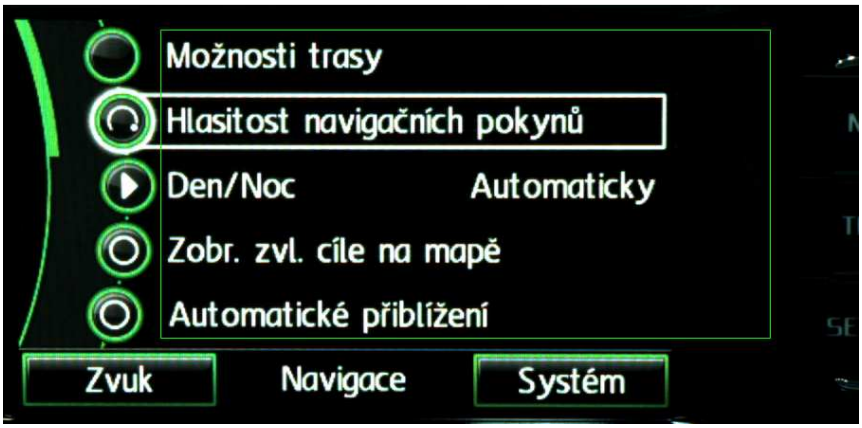
300 DPI, bylo dosaženo i při rozlišení 96 DPI úspěšnosti 91,16 % a s filtrem pro definování vyskytujících se znaků ještě o malinko lepší, tedy 91,78 %.

Další kritériem hodnocení výsledků poskytovaných vytvořeným krokem Tesseract OCR je rychlost zpracování. V tomto ohledu nemůže Tesseract OCR konkurovat OCR algoritmu od NI, neboť využívá externího souboru. Rychlost zpracování totožných obrázků a na totožných částech textu (Obr. 28) při použití Tesseract OCR byla menší a dosahovala přibližně necelých 3 sn./s a při použití OCR od NI byla rychlost asi dvakrát vyšší a dosahovala rychlosti až kolem 7 sn./s.



Obr. 28 - Snímek pro testování rychlosti zpracování- jeden řádek

Při aplikaci regionu zaměření na celou hlavní část displeje (Obr. 29) byla rychlost při použití Tesseract OCR (pozn.- OCR od NI toto neumožňuje) pouze asi 1 sn./s. Při pěti řádcích by tedy celková rychlost zpracování při použití OCR od NI byla asi 1,4 sn./s, což je téměř shodné s rychlostí 1 sn./s při použití Tesseract OCR.



Obr. 29 - Snímek pro testování rychlosti zpracování- celé pole

6 Závěr a další možnosti

V diplomové práci bylo prozkoumáno téma rozpoznávání textu v programu Vision Builder. Program Vision Builder disponuje svým vlastním algoritmem pro rozpoznávání textu, avšak jelikož se jedná o patentovanou technologii firmy National Instruments, nebylo možné do něj velmi podrobně nahlédnout. Přesto však bylo možné toto OCR odzkoušet na reálných obrázcích displejů autorádií a bylo tak možné zjistit určité omezující faktory. K dispozici byly obrázky s rozlišením 96 DPI. Úspěšnost rozpoznávání je velmi závislá na podmínkách, za kterých je snímek pořizován. Pro větší účinnost je žádoucí, aby rozlišení snímaných obrázků bylo výrazně větší než 96 DPI, neboť jak se ukázalo, toto rozlišení nedává předpoklady pro vysokou účinnost.

Algoritmus OCR implementovaný ve Vision Builder se podařilo rozšířit o další algoritmus, jež využívá program Tesseract OCR. Za použití nově vytvořeného kroku Tesseract OCR se podařilo dosáhnout vyšší účinnosti i při relativně nízkém rozlišení použitých obrázků, které způsobovalo dotyk jednotlivých znaků. Zatímco největším problémem při rozpoznávání pomocí OCR od NI bylo nízké rozlišení obrazu a tedy i problém se segmentací jednotlivých znaků, největším problémem při rozpoznávání pomocí kroku Tesseract OCR, byla česká slova a to i přes to, že byla vytvořena znaková sada pro české znaky. Program si tedy zatím nedokáže poradit s diakritikou a z toho pramenilo výrazné snížení účinnosti rozpoznávání. Tato účinnost se pohybovala těsně pod hranicí 92 % a byla tak asi o 4 % vyšší než účinnost při použití OCR od NI. Pokud budeme brát v úvahu jisté limitující faktory, především rychlost zpracování snímků, jeví se použití Tesseract OCR jako slibné řešení do budoucna. Pro lepší celkovou účinnost algoritmu však bude nutné zvýšit účinnost rozpoznávání českých znaků. Testování proběhlo za použití již vytvořené rozsáhlé znakové sady, která zajišťuje plnohodnotné rozpoznávání mnoha fontů, v budoucnu by však bylo vhodné učení nových znaků zakomponovat přímo do kroku Tesseract OCR. V době odevzdávání diplomové práce byl uživatelský krok Tesseract OCR zprovozněn na testovacím počítači. Problém nefunkčnosti na ostatních počítačích je řešen i s technickou podporou NI. Na zprovoznění uživatelského kroku pro další zařízení stále usilovně pracuji.

Bibliografie

1. **Zain, Jasni Mohamad.** Optical Character Recognition By Using Template Matching (Alphabet) (Jasni Mohamad Zain). *Academia.edu.* [Online] [Citace: 13. únor 2012.] http://ump.academia.edu/JasniMohamadZain/Papers/730698/Optical_Character_Recognition_By_Using_Template_Matching_Alphabet_.

2. **Kamil Kopecký, David Nocar, Roman Kopecký.** OCR technologie v pedagogických disciplínách. *OCR technologie v pedagogických disciplínách.* [Online] [Citace: 13. únor 2012.] http://epedagog.upol.cz/eped3.2003/clanek03.htm.

3. **Budhiraja, Vicky.** Image pre-processing for good OCR results - tesseract-ocr | Skupiny Google. *tesseract-ocr | Skupiny Google.* [Online] Google Inc., 21. únor 2011. [Citace: 3. březen 2012.] http://groups.google.com/group/tesseract-ocr/browse_thread/thread/839684fcebe0c935.

4. **Smith, Ray.** Documentation - tesseract-ocr - Technical papers describing various aspects of Tesseract. . *tesseract-ocr | Skupiny Google.* [Online] 2007. [Citace: 3. březen 2012.] http://tesseract-ocr.googlecode.com/svn/trunk/doc/tesseracticdar2007.pdf.

5. **Chen, C. H. a Wang, P. S. P.** *Pattern Recognition and Computer Vision.* London : World Scientific, 2005. 981-256-105-6.

6. **Sonka, Milan, Hlaváč, Václav a Boyle, Roger.** *Image Processing, Analysis an Machine Vision.* Toronto : Thomson Learning, 2008. 0-495-08252-X.

7. **Efford, Nick.** *Digital Image Processing.* Harlow : Pearson Education Limited, 2000. ISBN 0-201-59623-7.

8. 7.2. Konvoluční matice. *GIMP Documentation.* [Online] [Citace: 4. březen 2012.] http://docs.gimp.org/2.2/cs/plug-in-convmatrix.html.

9. **Pavelek, Milan.** 15.6 Filtrace obrazů. [Online] [Citace: 3. březen 2012.] http://ottp.fme.vutbr.cz/~pavelek/optika/1506.htm.

10. Neuronové sítě. *Department of Cybernetics.* [Online] ČVUT, 3. září 2012. [Citace: 9. březen 2012.]

11. **Klán, Petr.** Home Page of Petr Klán. *Institute of Computer Science Academy of Sciences of the Czech Republic.* [Online] [Citace: 9. březen 2012.] http://www.uivt.cas.cz/~pklan/CHI7.pdf.

12. Izomorfismus - Wikipedie. *Wikipedie, otevřená encyklopedie.* [Online] 4. listopad 2011. [Citace: 9. březen 2012.] http://cs.wikipedia.org/wiki/Izomorfismus#Definice_pro_grafy.

13. **National Instruments.** NI Vision Concepts Manual - June 2008 - National Instruments. *National Instruments: Test, Measurement, and Embedded Systems.* [Online] červen 2008. [Citace: 3. březen 2012.] http://www.ni.com/pdf/manuals/372916g.pdf.

14. Segmentation (image processing) - Wikipedia, the free encyclopedia. *en.wikipedie.org.* [Online] únor 2012. [Citace: 15. únor 2012.] http://en.wikipedia.org/wiki/Image_segmentation.

15. Tesseract (software) - Wikipedia, the free encyclopedia. *Wikipedia, the free encyclopedia.* [Online] 26. únor 2012. [Citace: 31. březen 2012.] http://en.wikipedia.org/wiki/Tesseract_(software).

16. ReleaseNotes - tesseract-ocr. *tesseract-ocr | Skupiny Google.* [Online] Google Inc., 2011. [Citace: 31. březen 2012.] http://code.google.com/p/tesseract-ocr/wiki/ReleaseNotes.

17. **Smith, Ray, Antonova, Daria a Lee, Dar-Shyang.** Documentation - tesseract-ocr. *tesseract-ocr | Skupiny Google.* [Online] 25. červen 2009. [Citace: 31. březen 2012.] http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/cs//pubs/archive/35248.pdf.

18. Otsu's method - Wikipedia, the free encyclopedia. *Wikipedia, the free encyclopedia.* [Online] březen. 26 2012. [Citace: 31. březen 2012.] http://en.wikipedia.org/wiki/Otsu's_method.

19. **Birdal, Tolga.** Famous Otsu Thresholding in C# - CodeProject®. *CodeProject® - Your Development Resource.* [Online] 20. červenec 2009. [Citace: 31. březen 2012.] http://www.codeproject.com/Articles/38319/Famous-Otsu-Thresholding-in-C.

20. **Smith, Ray.** *CVC - Centre de Visió per Computador.* [Online] 2009. [Citace: 31. březen 2012.] http://www.cvc.uab.es/icdar2009/papers/3725a241.pdf.

21. Directed acyclic word graph - Wikipedia, the free encyclopedia. *Wikipedia, the free encyclopedia.* [Online] 20. listopad 2011. [Citace: 15. duben 2012.] http://en.wikipedia.org/wiki/Directed_acyclic_word_graph.

22. What Is NI LabVIEW? - National Instruments. *Ntional Instruments.* [Online] [Citace: 13. únor 2012.] http://www.ni.com/labview/whatis/.

23. **National Instruments.** NI Vision Builder for Automated Inspection Development Toolkit User Guide - September 2006. *National Instruments: Test, Measurement, and Embedded Systems.* [Online] září 2006. [Citace: 7.. březen 2012.] http://www.ni.com/pdf/manuals/371424b.pdf.

24. TrainingTesseract3 - tesseract-ocr. *tesseract-ocr | Skupiny Google.* [Online] Google Inc., 6. březen 2012. [Citace: 3. únor 2012.] http://code.google.com/p/tesseract-ocr/wiki/TrainingTesseract3.

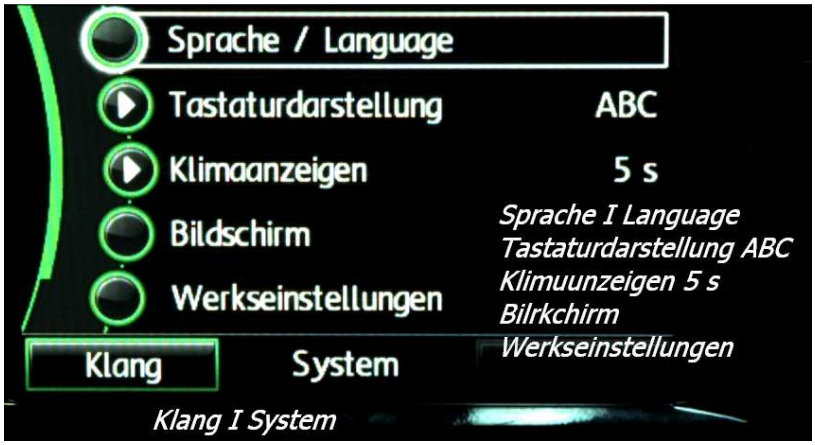
25. scA1600-14fc. *Průmyslové-kamery.cz | ESHOP*. [Online] [Citace: 30. duben 2012.] http://eshop.prumyslove-kamery.cz/IEEE1394-15990/161-scA1600-14fc.

26. FAQ - tesseract-ocr - Frequently Asked Questions. *tesseract-ocr | Skupiny Google.* [Online] Google Inc., 26. duben 2012. [Citace: 30. duben 2012.]

27. Diskrétní 2D konvoluce. *BruXyho home page.* [Online] 1. prosinec 2006. [Citace: 5. březen 2012.] http://bruxy.regnet.cz/fel/36ACS/konvoluce.pdf.

28. **Smith, Ray.** HP Labs : Technical Reports. *HP Labs - Advanced Research at HP*. [Online] prosinec 1994. [Citace: 14. duben 2012.] http://www.hpl.hp.com/techreports/94/HPL-94-113.pdf.

Přílohy

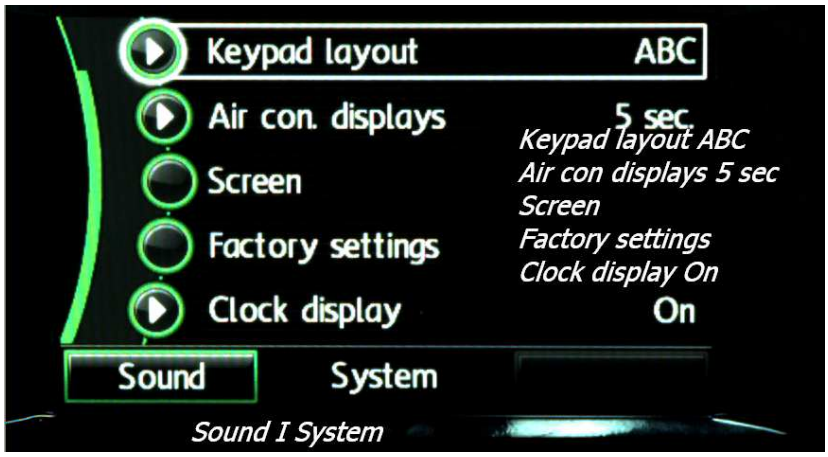


Obrázková příloha 1 - Rozpoznávání textu pomocí uživatelského kroku (b)

Obrázková příloha 2 - Rozpoznávání textu pomocí uživatelského kroku (c)

Obrázková příloha 3 - Rozpoznávání textu pomocí uživatelského kroku (d)

Obrázková příloha 4 - Rozpoznávání textu pomocí uživatelského kroku (e)



Obrázková příloha 5 - Rozpoznávání textu pomocí uživatelského kroku (f)

Obrázková příloha 6 - Rozpoznávání textu pomocí uživatelského kroku (g)

Obrázek	Počet znaků	Počet chybně rozpoznaných znaků			
		Fixní mód	Uniformní mód	Tesseract OCR	Po upřesnění znaků
test (1)	47	5	23	4	4
test (2)	15	2	4	3	1
test (3)	4	0	0	0	0
test (4)	41	6	13	3	3
test (5)	15	1	4	3	1
test (6)	56	3	25	5	5
test (7)	53	2	18	6	5
test (8)	54	8	20	8	4
test (9)	40	3	17	6	5
test (10)	61	9	15	4	4
test (11)	74	7	16	2	2
test (12)	85	7	49	15	15
test (13)	97	15	46	7	7
test (14)	115	15	63	15	15
test (15)	117	8	80	11	11
test (16)	87	6	55	7	7
test (17)	85	7	53	7	7
test (18)	48	5	26	4	4
test (19)	89	7	45	8	8
test (20)	92	29	62	4	4
test (21)	92	24	52	4	4
test (22)	83	18	51	11	11
test (23)	75	22	56	5	5
test (24)	25	1	12	0	0
test (25)	81	27	56	1	1
test (26)	86	6	47	10	10
test (27)	37	4	13	5	3
test (28)	45	7	10	0	0
test (29)	71	3	38	6	6
test (30)	41	6	23	5	5
	1911	**263**	**992**	**169**	**157**

Tabulková příloha 1 - Seznam obrázků s počtem chybně rozpoznaných znaků