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

DIGITALIZACE HISTORICKÝCH MAP

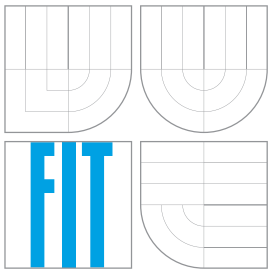
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JOSEF POSPÍŠIL

BRNO 2008

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

DIGITALIZACE HISTORICKÝCH MAP

ANCIENT MAPS DIGITIZING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JOSEF POSPÍŠIL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠPANĚL

BRNO 2008

Abstrakt

Tato práce se zabývá zpracováním historických map, konkrétně jejich digitalizací a vektorizací. Zaměřuje se na mapy z druhého vojenského mapování a pokouší se najít metody, které by mohli být užitečné pro odstranění textu z těchto map.

Klíčová slova

historické mapy, druhé vojenské mapování, digitalizace, AdaBoost

Abstract

This work is about processing of historical maps, especially their digitizing and vectorization. The main focuses of this project are maps from the second historical military mapping and finding methods useful for removing texts from these maps.

Keywords

historical maps, the second historical military mapping, digitize, AdaBoost

Citace

Josef Pospíšil: Digitalizace historických map, diplomová práce, Brno, FIT VUT v Brně, 2008

Digitalizace historických map

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Michala Španěla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal, v seznamu literatury.

.....
Josef Pospíšil
18. května 2008

Poděkování

Rád bych poděkoval Ing. Michalovi Španělovi za čas, který mi věnoval. Dále bych rád poděkoval vedení této fakulty za pružnost a ochotu reagovat na náměty studentů. Za pomoc také děkuji Doc. Dr. Ing. Pavlovi Zemčíkovi, neboť z jeho předmětů o zpracování obrazu vychází tato práce a v neposlední řadě Ing. Petrovi Přidalovi za pomoc při návrhu této práce.

Obsah

Kapitola 1

Úvod

Úkolem vědy a vědění není jen usnadňovat lidem život, ale také rozšiřovat jejich zájem o okolní svět. S rostoucím množstvím informací o tomto světě a s rostoucí hloubkou, do jaké ve vědě pronikáme, již není možné uchovávat informace pouze ústně. Je potřeba nacházet stále nová média, kde lze vědění zaznamenávat a uchovávat. Již od počátku byla v lidech touha něco po sobě zanechat, něco zaznamenat, něco předat dalším generacím, aby se poučili a ve výzkumu začali tam, kde minulá generace skončila.

Prvotní formu předávání informací by se dalo spatřit v obrázcích na zdech jeskyní,(k rozvoji řeči a komunikace mezi lidmi už zabíhat nebudu) přes všechny napodobeniny a sochy z různých materiálů až k rozvoji písma a matematiky. Písmo pak mělo za následek rozvoj písemného uchovávání informací. O písmu by se vlastně dalo říci, že jde o první záznam s digitálními parametry. Oproti obrazům bylo možné knihy reprodukovat (opisovat) bez ztráty obsahu a tedy kvality, naopak při replikaci obrazu mohlo a stále docházelo ke změnám. Dá se to naznačit na příkladě. Představme si, že neznámý malíř vytvoří mapu starověkého Říma. Vojevůdce Maximus by rád Řím dobyl a proto si nechá tajně mapu přemalovat, při tom vznikne v replikaci několik, možná bezvýznamných chyb. Ještě dřívě, než stihne Maximus Řím dobýt, nechají se z jeho repliky mapy vytvořit další kopie pro jeho velitele a na jejich mapách už budou jednak chyby vzniklé při první replikaci plus chyby vzniklé při dalších replikacích. Z map také stářím zmizí spousta detailů a konečně tím dojde k neopravitelné degeneraci informací na těchto mapách. Neopravitelné jsou i proto, že Řím před několika lety, kdy se tato mapa vytvářela, vypadal jinak. Časem vždy přestanou u historických materiálů existovat zdroje těchto materiálů.

Přiblížím se více tématu historických map. Nabízí se otázka jak uchovávat staré mapy. Prakticky na každém médiu, ať už jde o papír, plátno, zeď jeskyně či cokoli jiného, dochází časem k degradaci. Zatím nebylo nalezeno médium s „nekonečnou“ trvanlivostí. Existuje možnost data bez ztráty replikovat a uchovat na médiu, které nemá nekonečnou trvanlivost, ale s možností data beze ztráty replikovat tak aby i replika repliky byla shodná se vzorem. Poté by bylo možné data uchovávat tak, že by se vždy překopírovala na novější médium.

Proč ale uchovávat staré mapy, když novější jsou přesnější a o to přece u map jde? Důvodů je několik, jedním z nich je i to, že historická mapa je většinou malovaná a stejně tak jako obrazy a jiná díla z minulosti, má historickou a uměleckou hodnotu. Dalším důvodem proč spravovat historické mapy je možnost čerpat z nich informace. Například srovnání rozlohy lesů, polí vod a dalších před 500 lety bývá zajímavým údajem o tendenci vzestupu nebo sestupu těchto rozloh. Staré mapy obsahují cesty, které mohly vejít v zapomnění a přitom právě tyto cesty mohou pomoci naplánovat kudy vést novou silnici nebo turistickou trasu. Právě mnoho cest je založeno na historických podkladech. Všechny tyto informace

je zajímavé z map získavat, což lze nejlépe po digitálním zpracováním.

Cílem diplomové práce je hledání způsobů jak zpracovávat digitální mapy z druhého vojenského mapování. Zaměřuji se na nalezení metod k identifikaci a odstranění nápisů na mapách. Je to krok, který se dělá skoro při každé digitalizaci historických map. Samotné nápisy totiž na mapě zabírají hodně místa a při rozkonponování na určité oblasti (lesy, louky, zástavba) by negativně ovlivňovaly výsledky. Taktéž nápisy brání při hledání cest na historických mapách. Cílem mojí diplomové práce tedy je text rozlišit od ostatních prvků mapy, zvláště pak od cest, které se často mohou s textem splést. Po provedení nalezení a odstranění textu se budu snažit vhodnými metodami nahradit vzniklá místa po odstraněných nápisech.

Dalším cílem této práce je lépe se seznámit s knihovnou OpenCV pro zpracování obrazu, se způsoby digitalizace historických map a s metodami na hledání vzorů v obraze.

Výsledkem by měla být přenositelná knihovna funkcí pro detekci a odstranění nápisů z map z druhého vojenského mapování a s funkcemi, které by měli odhadnout původní povrch mapy, před tím, než na ni byl natisknut nápis.

Text je rozdělen do několika částí, první se zabývá historickými mapami, kterým se tato práce věnuje, poté se zaměřuji na základní seznamení se způsoby jakým se mapy digitalizují a vektorizují. Postupně se práce dostane přes teorii zpracování obrazu k návrhu možností, jak odstranit text ze starých map. V návrhu je popsána většina nápadů, která doprovázela tento projekt. Nakonec se tato práce bude věnovat metodám, které byly aplikovány a posléze také zhodnocením celé práce a diskuzi nad možným rozšířením.

Tato diplomová práce vychází ze semestrálního projektu, který rozšiřuje o celou teorii detektorů a některých úprav obrazu, o celý návrh všech použitých metod, kromě detekce hran. Dále přibyla implementace a zhodnocení výsledků.

Kapitola 2

Druhé vojenské mapování

Kapitola nás seznamuje se základními informacemi o druhém vojenském mapování. Mapy z něj jsou výchozími body pro tuto práci. Právě na nich se budu pokoušet odstranit nápisy a překrýt je odhadnutým povrchem, který zastírají.

Obrázek 2.1: Ukázka mapy z druhého vojenského mapování z let 1842–1852

Historická mapa 2.1 zobrazuje část dnešního území České republiky (okolí Havlíčkova Brodu) před více jak 150-ti lety. Druhé vojenské mapování též nazývané Františkovo podle císaře Františka II., na jehož popud bylo provedeno, proběhlo v letech 1806 až 1866 na území Rakouska-Uherska.

Jeho vzniku předcházela vojenská triangulace, která sloužila jako geodetický základ tohoto díla, oproti I. vojenskému mapování můžeme tedy sledovat zvýšenou míru přesnosti. Podkladem byly mapy Stabilního katastru v měřítku 1 : 2 880, což mělo také pozitivní vliv na přesnost map. Z výsledků tohoto mapování byly odvozeny mapy generální (1: 28 800) a speciální (1: 144 000).

Mapování Čech proběhlo v letech 1842–1852. Morava a Slezsko byly mapovány v letech 1836–1840.

Měřítko mapování bylo 1:28 800.

Obsahem map jsou cesty, zděné budovy a kamenné mosty. Z přírodních prvkůn to jsou pole, louky, pastviny, lesy, rybníky a toky. Pro znázornění terénu byly využity Lehmannovy šrafy, které znázorňovaly směr největšího spádu terénu a jeho velikost. Pomocí grafického protínání byla zjišťována poloha vrcholových tvarů a průběh čar terénní kostry. Měřením nebo odhadováním úhlů sklonu byly získávány potřebné údaje k půdorysnému vyjádření terénních šraf. Z [3].

Lehmannovo šrafování vytváří obraz reliéfu vypovídající o směru jeho největšího spádu i o jeho velikosti. Se vzrůstem sklonu se šrafy zkracovaly. Kreslily se ve směru spádnic a jejich síla rostla v tomto odstupňování: při nulovém sklonu byl poměr stínu (šrafy) ke světlu (bílé mezeře) 0:9, při pětistupňovém 1:8 a při sklonu svahu 45° 9:0. Vodorovné a mírně skloněné plochy zůstaly v mapě bílé, naopak strmé svahy byly zobrazeny silnými a krátkými šrafami s úzkými bílými mezerami. Z [12].

Obsah mapy je v podstatě totožný s I. vojenským mapováním, přidány byly pouze výšky trigonometrických bodů, avšak zobrazovaná situace se velmi liší. Mapy II. vojenského mapování vznikaly v době nástupu průmyslové revoluce a rozvoje intenzivních forem zemědělství, kdy vzrostla výměra orné půdy za 100 let o 50% a lesní plochy dosáhly u nás historicky nejmenšího rozsahu. Převzato z [9] a z [7].

Jednotlivé části celé mapy mají jemné odlišnosti jasu v obraze, ale drží jednotný styl, proto by aplikace měla mít určitý interval automatického nastavení, aby byla schopna zpracovávat všechny mapy z tohoto mapovaní.

Kapitola 3

Digitalizace a vektorizace map

Kapitola o digitalizaci okrajově popisuje proces převádění existujících map do digitální podoby. Jelikož se v diplomové práci zabývám spíše až samotnými úpravami digitalních map, není potřeba do této části hlouběji pronikat. Některé z informací jsem získal z [8].

Zde lze nalézt odpověd na otázky proč digitalizaci map provádět, jak ji provádět a jak výsledná digitální data ukládat.

3.1 Snímání map

Tato kapitola se věnuje přenosu map do digitální podoby, jejich vektorizací a několika základními úpravami, aby mapy odpovídali skutečnosti a zachovávaly poměry vzdáleností mezi jednotlivými objekty.

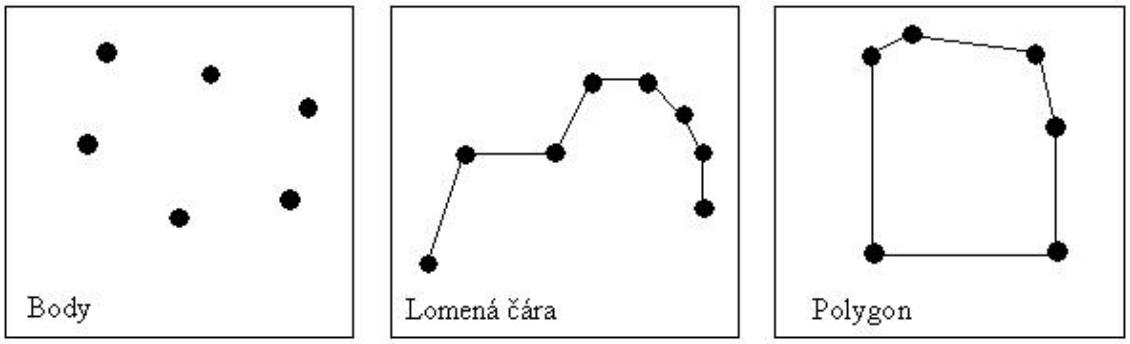
Na začátek bych položil otázku proč vůbec mapy digitalizovat? Je pravdou že při každé digitalizaci z mapových podkladů jako je papír, plátno nebo něco podobného, dochází ke ztrátě dat. Je to součástí převodu ze spojitého signálu, za který lze obraz považovat, do diskrétních hodnot, jenž potřebujeme pro uložení v počítači. Toto je ovšem poslední ztráta, která historický materiál potká. Poté lze tyto data libovolně kopírovat bez nárustu chyb. Taktéž je lze mnohem lépe distribuovat širší veřejnosti. Paradoxně se pak možná originál zachová v lepším stavu, jelikož bádání historiků bude probíhat většinou na digitální kopii tohoto dokumentu.

Existuje více metod přenosu dat do počítače, nyní seznámíme se s těmi základními.

3.1.1 Ruční digitalizace

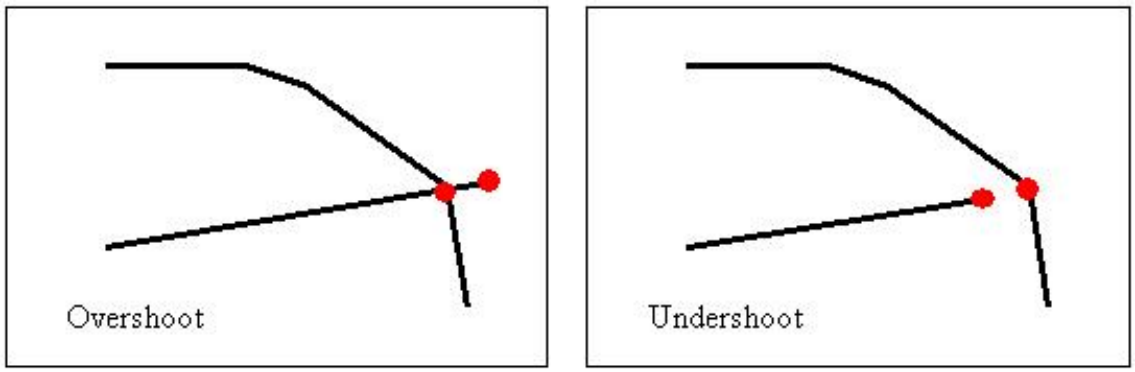
Základní metoda digitalizace klasických map na papíře probíhá tak, že se stará mapa položí na digitalizační zařízení a ručně se označují všechny významné body na mapě. Digitalizační zařízení v tomto případě funguje podobně jako touchpad. Při digitalizaci se pouze dotekem přes papír mapy zadávají významné body v mapě. Existuje několik základních typů bodů jenž se zadávají. Bod může být jenom bod nebo je součástí vetších objektů třeba lomené čáry nebo polygonu. Tyto objekty jsou poté definovány sadou bodů jenž se ručně zadají a poté se spojí vektory. Pochopitelně čím více bodů se zadá, tím více se výsledek podobá originálu a lze dosáhnout méně hranatějších tvarů. (viz. obrázek 3.1).

Většinou lze body zadávat s velkou přesností, což samozřejmě závisí take na obsluze tohoto zařízení, která body zadává. Přesto je obvykle nutné po zadání všech bodů mapy opět všechny body projít a odstranit několik chyb v jejich zadání. Existují dva základní typy chyb - tzv. undershoots a overshoots což by se dalo přeložit jako nedostřelení a přestřelení.



Obrázek 3.1: Typy jednotlivých bodů při digitalizaci

Tyto chyby jsou většinou dány tím, že cesty na mapě jsou obvykle zaznačeny větší šířkou čáry než v mapě ve vektorové podobě, takže při zádávání leží některé body na okraji cesty a jiné na středu cesty. Lépe to popisuje obrázek 3.2.



Obrázek 3.2: Obvyklé chyby při ruční digitalizaci

Výsledek tohoto způsobu digitalizace je také závislý na přesnosti staré mapy. Většinou je nutné tyto mapy poté tzv. georeferencovat tedy upravit je tak aby správně vystihovaly poměry vzdáleností mezi jednotlivými body. K této problematice se vrátím později.

Výhody této metody:

- Možnost okamžitě opravovat chyby a poškozené části staré mapy
- Člověk jenž zařízení obsluhuje většinou velmi spolehlivě rozpoznává objekty mapy.
- Věci, které na starých mapách chybí, lze okamžitě doplnit např. z různých jiných zdrojů. Zde záleží na zkušenosti obsluhy.
- Na konci této metody vznikne vektorizovaná mapa.

Nevýhody této metody:

- Tento způsob digitalizace je pracný a velmi časově náročný, což je příčinou jeho vysoké ceny.

- Výsledek často velmi záleží na zkušenosti obsluhy.
- I od stejného člověka, který zařízení obsluhuje, mohou vyjít různé výsledeky ovlivněné jeho momentálním stavem (únava, stres, ospalost).

3.1.2 Skenování

Druhou možností jak digitalizovat staré mapy je jejich skenování. Zařízení pro tuto činnost je několik. Od těch nejjednoduších skenerů přes fotoaparáty ke složitějším zařízením. Vždy však závisí na stavu daného skenovaného dokumentu, podle něho se kupříkladu můžeme rozhodnout pro focení, jelikož je k němu nejšetrnější.

Obrazy (mapy) které se mají skenovat by měli být v co nejlepším stavu s minimálním počtem nápisů. Skener pak pouze rozlišuje intenzitu a barvu jednotlivých pixelů a ukládá jejich hodnoty do rastru, což je tabulka o velikosti obrazu, kde se jedna buňka nazýva pixel.

Tento rastr zatím neumožňuje šikovně mapu využívat, jde totiž o pouhý obrázek v počítači. Abychom mohli z map lépe získávat informace, je lepší převést tuto rastrovou podobu mapy do podoby vektorové, kde všechny objekty mapy jsou popsány matematicky přes orientované úsečky a další analytická primitiva. Vektorizace se provádí dvěma metodami - automatická vektorizace a manuální vektorizace. V závislosti na použité metodě vektorizace se vybere odpovídající rozlišení při skenování. Mnoho programů pro vektorizaci potřebuje aby veškeré čáry na mapě byli minimálně 3 pixely široké aby se daly převést na vektory. V důsledku to pak znamená, že by se měli mapy skenovat alespoň s rozlišením 200 dpi – 300 dpi. Toto rozlišení se týká automatické vektorizace, při použití manuální vektorizaci postačí rozlišení od 75 dpi do 150 dpi

Automatická vektorizace

Tuto metodu je vhodné použít pokud skenujeme velké množství map v dobrém stavu. Také je dobré aby se na celé skenované mapě používala shodná označení pro stejné objekty, aby každý typ cesty v mapě byl označen čárou určité šířky a tato šířka se držela v celé mapě. Dále je užitečné mít všechny průsečíky cest a jiných hraničních oblastí vyznačeny velmi zřetelně a pro počítač rozpoznatelně. Cestou jsem v tomto popisu myslel i například hranici mezi lesem, loukou apod. Při tomto automatickém zpracování je taktéž potřeba, aby mapy obsahovaly minimum popisků a jiných podobných objektů. V případě, že budou všechny objekty na mapě od sebe rozeznatelné, oddělitelné a aplikce pro vektorizaci správně rozpozná všechny typy cest, místa odkud a kam vedou a kde se protínají, bude nutný pouze malý počet oprav po vektorizaci. Naopak při nekvalitním vstupu bude potřeba hodně mapu doopravovat. Výhody této metody:

- Může být velmi rychlá.
- V porovnání s ruční vektorizací je levná.
- Její výsledky jsou velmi přesné a správné.

Nevýhody této metody:

- Špatně rozeznává text a málo obvyklé značky v mapě.
- Vyžaduje dobrou vstupní mapu, která správně a konzistentně popisuje objekty v krajině.

- Při špatném výsledku je potřeba hodně oprav, které jsou časově a tedy i finančně náročné.

Manuální vektorizace

Tato metoda se používá ve dvou základních případech. V prvním případě nelze vstupní mapu vektorizovat automaticky a to kvůli kvalitě nevhodné pro automatickou digitalizaci. Ve druhém případě, pokud nemáme zařízení pro ruční digitalizaci, tak tuto digitalizaci, jejíž výstupem je vektorový obraz, provádíme na počítači. Jde vlastně o tu samou metodu ruční digitalizace, ovšem nyní je vytvářena na počítači. I zde dochází ke stejným chybám, jak bylo popsáno v podkapitole o ruční digitalizaci.

Výhodou této metody je to, že s ní může zároveň pracovat více lidí. Každý z nich zpracovává na počítači určitou část mapy. Taktéž je tato vektorizace snažší než ruční digitalizace, protože práce za počítačem je méně namáhavější než práce u digitalizačního zařízení pro ruční digitalizaci.

Nevýhodou této metody je opět jako u ostatních manuálních metod časová náročnost a pracnost.

3.1.3 Shrnutí

Jelikož nemám k dispozici zařízení pro ruční digitalizaci ani originální mapy, pouze jejich už naskenované podoby, budu postupovat jako bych mapy získal skenováním. Ruční digitalizaci jsem zde uvedl pro úplnost metod pro digitalizaci map. V dalších kapitolách se nebudu zabývat formáty pro uložení vektorových dat. Mojí snahou je zatím předpříprava map pro automatickou vektorizaci. Tedy zaměření se na odstranění některých specifických objektů z mapy. Pravděpodobně tedy vůbec nebudu s vektorovými formáty pracovat.

3.2 Georeferencování

Georeferencování map je transformace, při které jsou mapy upravovány tak, aby v určitém měřítku odpovídaly přesně skutečnosti. U georeferencování nynějších map je potřeba vypořádat se se zakřivením země. Je prakticky nemožné přenést na mapu skutečný obraz krajiny z podstaty toho, že jde o funkci z trojrozměrného prostoru do prostoru mapy, tedy prostoru dvojrozměrného. Pak tedy čím je mapa větší tím více tam dochází k nepoměrům.

Při georeferenci se obecně postupuje takto: nejdříve se vybere několik skutečných a známých referenčních bodu (nejméně tři) o kterých víme, kde přesně leží a jak jsou od sebe vzdáleny, taktéž tyto body známe na mapě. Tyto body umístíme v měřítku mapy pod tuto mapu a podle nich mapu morfujeme tak, aby tyto referenční body odpovídaly bodům na mapě. Čím více bodů použijeme tím by měla býti mapa přesnější.

Pro zjišťovaní referenčních bodů se používá více metod, dnes je velmi oblíbené GPS. Někdy je potřeba i vyšší přesnost než má GPS (odchylka až 7 metrů), pak je potřeba používat hodnoty z přesnějších měření.

Georeferencování u historických map je nutné proto, aby data zjištěné z historických map, např. rozloha lesů v roce 1850, odpovídala skutečnosti. Při georeferenciaci historických map se většinou používá nová, už georeferencovaná mapa, na kterou se historická mapa pasuje. Nejlépe tak, aby co nejvíce bodů v historické mapě odpovídalo správným bodům v mapě nové. To, jak se často historické mapy liší od map nynějších- přesnějších, je vidět na obrázku 3.3, kde je přes sebe položená historická a nynější mapa Kypru. Zde se prakticky

bez složitější georeference nelze obejít, neboť se mapy hodně liší a to nejenom v lineárním měřítku.

Obrázek 3.3: Stará a nová mapa ostrova Kypr na sobě

Kapitola 4

Zpracování obrazu

Kapitola popisuje poměrně obecně zpracování obrazu, věnuje se základnímu postupu při zpracování obrazu. Popisuje možnosti jak obraz uložit a jak lze který formát vhodně použít pro historické mapy. Taktéž jsou zde popsány barevné modely pro reprezentaci barev v obraze. Přestože se to týká této práce pouze okrajově, mohou být právě barvy a tedy použití barev některého barevného modelu důležitou součástí řešení některého problému při analýze obrazu.

4.1 Obecný postup při zpracování obrazu

Zde je velmi zkráceně popsaný postup zpracování obrazu. Tyto informace jsem čerpal především ze zdrojů [18], [15] a [11].

4.1.1 Snímání

Snímání obrazu je obecně převod optické veličiny na elektrický signál, přičemž optické veličiný nemusí být jen jas z kamery, ale mohou zde být i jiné veličiny např. ultrazvuk nebo elektromagnetické záření. V případě této práce nás však bude ze všeho nejvíce zajímat jasová složka obrazu.

4.1.2 Digitalizace

Jak jsem již uvedl výše, u zpracování map se jedná o převod analogového signálu do signálu digitálního. Digitální obraz, jak už bylo řečeno, lze popsat funkcí $f(x, y)$ kde x a y jsou souřadnice v prostoru obrazu. Samotný obraz je pak získán vzorkováním obrazu do matice s MxN body a kvantováním do K úrovní (určující jas a barevný odstín).

Velikost obrazu se obvykle udává v pixelech a rozlišení odpovídá poměru počtu pixelů na palec. Při nízkém rozlišení dochází k velkým ztrátám informací a naopak při vysokém rozlišení stoupá výpočetní náročnost úprav na obraze. Vzorkování by se mělo zvolit podle Shanonovy věty, která v tomto případě říká, že vzorkovací interval (vzdálenost mezi dvěma nejbližšími vzorky) by měla být maximálně polovina vzdálenosti dvou bodů, jenž chceme v obraze od sebe rozpoznat.

Jednotlivé vzorky lze ukládat do mřížky, která může být např. čtvercová nebo hexagonální. Každá má své výhody, hexagonální není vhodná pro Fouriérovu transformaci, ale naopak při řešení spojitosti objektů v obraze je výhodnější než mřížka čtvercová.

Popis barev v obraze

Jak přesně vypadá každý pixel určuje jeho odstín. Ten se dá zaznamenat pomocí barevných modelů. Nejpoužívanější je model RGB.

Model RGB Popis barvy v tomto modelu je velmi běžný, popisuje pixel pomocí tří barevných složek, které se aditivně skládají. Barva se složkami $0, 0, 0$ odpovídá barvě černé, naopak barvě bílé odpovídá barva se složkami $1, 1, 1$. Jednotlivé složky označují barvy red - červená, green - zelená a blue - modrá, které se skládají. V počítačové grafice se spíše používají hodnoty celých čísel v rozmezí 0–255, což je možné zapsat na 8 bitech čili na bajtu. Jelikož má tento model tři složky používá se tzv. 24-bitová hloubka, odpovídající třem složkám po osmi bitech. Samozřejmě je možné použít větší či menší rozsah barev. Při vytváření odstínu mluvíme o kvantování právě do složek mezi 0–255 na jedné složce. K popisu jedné barvy i v úrovni šedi by mělo být použito minimálně 50 úrovní jasu.

Model CMY, CMYK Jde o opačný model modelu RGB. Barvy se nepřičítají, ale odečítají od bílé. Rozdíl mezi CMYK a CMY je v tom, že CMYK obsahuje navíc černou barvu. Je sice pravda, že černá by se měla dát složit ze složek CMY, ale v praxi, např. v tisku, se používá černá zvlášť. Modelu CMYK odpovídají složky cyan-azurová, magenta-purpurová, yellow-žlutá a black černá.

Model HSI Tento model se liší oproti předchozím modelům v tom, že jeho složky neodpovídají základním barvám, ale popisují tři vlastnosti jednotlivých barev hue-odstín, saturation-sytost, intensity-jas. Odstín určuje barvu pixelu, sytost určuje množství bílé složky v barvě a jas určuje kolik světla daná barva odrazí, tedy její zářivost. paragraphModel YUV Tento model se v počítačové grafice příliš nepoužívá. paragraphStupně šedi Často se obraz zpracovává pouze ve stupních šedi, jelikož se takto dají barvy mezi sebou porovnávat. Stupeň šedi se vypočítá z barevných složek modelu RGB, podle vzorce: $I = 0,299 * R + 0,587 * G + 0,144 * B$.

4.1.3 Předzpracování obrazu

Po digitalizaci obrazu je někdy nutné obraz předzpracovat. Záměrem předzpracování je odstranění známých chyb z digitalizace a jeho příprava pro snažší identifikaci objektů v obraze. Existuje velmi mnoho metod na předzpracování obrazu, většinou záleží na tom, jak dále se má obraz zpracovávat. Obecně lze tyto metody rozdělit do tří skupin:

- Jasové transformace
- Geometrické transformace
- Filtrace a ostření obrazu

4.1.4 Segmentace

Jedním z nejtěžších kroků zpracování obsahu obrazu je jeho segmentace. Jde o analýzu obrazu, která má vést k nalezení objektů v obraze. Za objekty se zde považují části obrazu, jež jsou dále bodem zájmu v dalším zpracování. Výsledkem segmentace by měl být soubor oblastí odpovídající objektům ve vstupním obraze. Jedná se pak o tzv. kompletní segmentaci. Pokud ale oblasti neodpovídají přesně objektům, pak tuto segmentaci nazýváme částečnou.

4.1.5 Popis obrazu

Čtvrtým krokem je popis obrazu nebo též popis nalezených objektů z předešlé segmentace. Existují dva základní způsoby popisu. Jeden je založený na kvantitativním přístupu, což znamená popis objektů pomocí souboru číselných charakteristik. Mohou jimi být např. velikost objektu, kompaktnost apod. Druhou možností je kvalitativní přístup, ve kterém jsou popisovány vztahy mezi objekty a jejich tvarové vlastnosti. Způsob popisu je zvolen podle způsobu dalšího využití. Ve většině případů je tento popis vstupní informací pro rozpoznávání objektů. Výběr popisu je pak závislý na použitém rozpoznávacím algoritmu.

4.1.6 Klasifikace

Posledním krokem při zpracovávání obrazu je klasifikace objektů. Ve většině případů se jedná o zařazení objektů nalezených v obraze do skupin předem známých tříd. Metody klasifikace objektů se dělí do dvou základních skupin, které jsou úzce spjaty se způsobem popisu objektů. Jedná se o příznakové (statistické) rozpoznání a strukturální rozpoznání. Příznakové metody jsou založeny na principu využití příznaků, což je skupina číselných charakteristik objektu. Trénování vlastního klasifikátoru zde může být s trénovací sadou i bez ní na principu shlukové analýzy. Strukturální rozpoznávání využívá jako vstupu kvalitativní popis objektů. Objekty jsou zde popsány primitivy. Dále je definována abeceda, jazyk popisu a gramatika jednotlivých tříd. Vlastní rozpoznávání je pak založeno na principu rozboru slova a kontroly správnosti syntaxe pro všechny třídy. Celá podkapitola částečně citována z [1]. Klasifikace probíhá, jak už bylo částečně řečeno, pomocí detektorů a detekce, více o ni pojednávám v části o detekci.

4.2 Formáty pro uložení map

Jak už jsem ve shrnutí v podkapitole o snímání map naznačil, nebudu se zabývat vektorovými formáty pro ukládání obrazu. Tato práce se spíše zaměřuje na práci s rastry a tedy hlavně rastrovými formáty.

Soupis formátů určitě nebude kompletní, spíš bych jenom rád představil některé ze základních a nejběžnějších formátů pro uložení obrazu map a formáty se kterými budu v rámci této práce spolupracovat.

Na začátek bych rozdělil formáty do dvou skupin na ztrátové a bezztrátové. Každá skupina má svůj význam. Po naskenování je vždy vhodné uložit naskenovanou mapu bezztrátově, vyžaduje to sice větší prostorové nároky na uložení, ale jedná se o nejkvalitnější digitalizovanou podobu dokumentu. Ostatní kopie tohoto dokumentu už mohou být v různých formátech s různým rozlišením, ale tento základ se vždy uchovává.

4.2.1 Bezztrátové formáty

TIFF

První verze formátu TIFF (Tagged Image File Format) byla uvedena v roce 1987, šestá a zatím poslední specifikace pak v roce 1992. Vlastníkem formátu TIFF je nyní firma Adobe, která umožňuje využití TIFFu zdarma.

TIFF je typickým představitelem bitmapového formátu, tj. grafická informace je v něm vyjádřena formou matice obrazových bodů - pixelů, přičemž u každého pixelu je udána informace o jeho barvě. Největší výhodou uvedeného typu formátů je schopnost věrné repre-

zentace "přirozeného" obrazu (sejmutého například skenerem či digitálním fotoaparátem). Dalšími výhodami jsou robustnost (nehrozí ztráta informací při přenosu do jiného prostředí) a poměrně snadné zpracování při výstupu. Jednou z hlavních nevýhod je velký objem souborů, rostoucí úměrně s rozměry a rozlišením (redukci objemu nicméně napomáhají různé kompresní bezztrátové algoritmy).

Na rozdíl od většiny ostatních bitmapových formátů mohou být dokumenty v TIFFu i vícestránkové a díky tomu v nich lze uložit i poměrně velké bitmapy. Jak již označení "tagged" naznačuje, v TIFF souborech je možno použít různé tagy, tedy klíčová slova popisující vlastnosti obrázku - toho je využito k tvorbě různých rozšíření a modifikací. Obrázky ve formátu TIFF jsou schopny nést nejširší spektrum barevných informací (černobílá grafika, odstíny šedi, RGB, CMYK, CIELab, indexované barvy aj.). TIFF také podporuje využití řady bezztrátových kompresních algoritmů (PackBits, LZW, Huffman RLE)a barevné profily ICC. Oficiálně je v TIFFu také možno využít i ztrátovou JPEG kompresi, specifikace je však v tomto ohledu nepříliš povedená.

Formát TIFF je zatím nejčastější způsob ukládání prvních naskenovaných map. Časem možná bude vytlačen progresivnějšími formáty, jako jsou PNG, PDF či JPEG2000. Z části citováno z [2].

BMP

BMP je další bitmapový formát, který je ovšem kompatibilní hlavně v prostředí Windows. To ho odsunuje až za formát TIFF či PNG. Jelikož však lze bez ztráty kvality tyto formáty převádět mezi tiffem a bmp, budu v předpokládané knihovně, kterou bych chtěl vyvíjet ve Windows, tento formát používat pro testování.

4.2.2 Ztrátové formáty

Ztrátové formáty dosahují obrovského zmenšení velikosti souboru s digitalizovanou mapou. Mapy, které jsou takto zkomprimovány se hodí pro lepší přenositelnost např. pro použití na webu nebo při prezentacích. Při vhodně zvolené míře komprese dochází, jak už bylo řečeno, k obrovskému snížení velikosti dat obrazu při minimální, lidsky rozeznatelné ztrátě kvality.

Monopost v těchto formátech nyní drží formát jpeg, který také v této práci používám k prezentaci výsledků.

JPEG

Formát jpeg je založen na Fourierově transformaci (diskrétní kosinová transformace) - využívající při komprimaci obrázku sinusových funkcí - avšak bere obrázek po malých čtvercových blocích. Popisy těchto bloků jsou pak v komprimovaném souboru uloženy v pořadí, odpovídajícím rozkladu obrázku směrem shora dolů. To vede k riziku narušení vizuální věrnosti po dekomprimaci. Přechody jednotlivých bloků mohou být však viditelné. Jeho hlavní plánované použití je na webu pro komprimování fotografií. Při komprimaci obrázku s textem dochází k dosti výrazné ztrátě kvality na vysoce kontrastních hranách. Pro použití s mapovým materiálem se nehodí, využitelný je pravděpodobně pouze při prezentaci map nebo při jejich přenosu jako takovém, kde je třeba přenést co největší množství map na omezeném datovém prostoru.

Tento formát je volně šířitelný a velice rozšířeně používaný. Je snaha nahradit formát JFIF formátem JPEG 2000 založeném na vlnkové transformaci s lepšími kompresními výsledky. Nahrazení však bude ještě trvat a to díky popularitě, kterou JPEG má.

4.3 Úpravy obrazu

4.3.1 Prahování

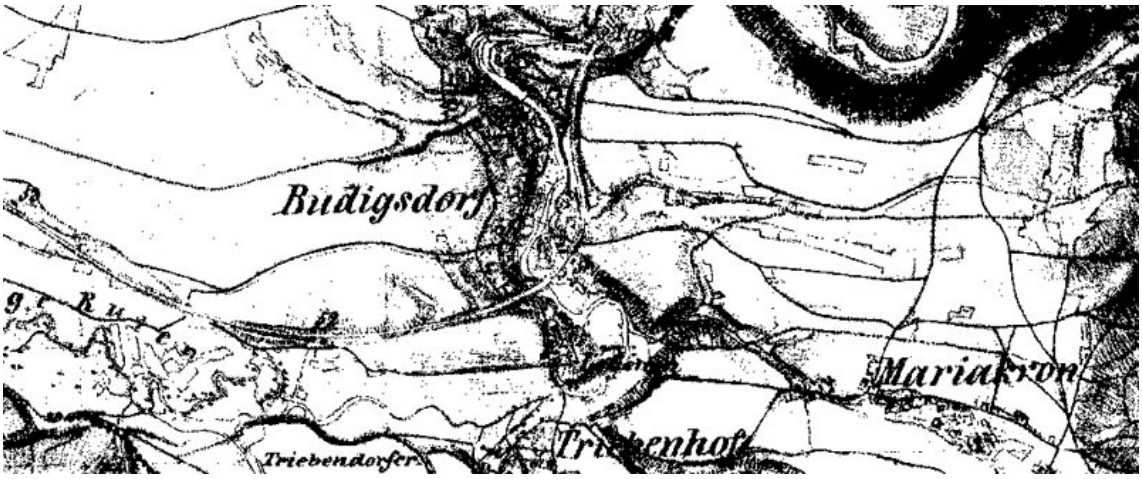
Prahování je nejjednodušší a nejstarší metoda segmentace. Patří mezi nejpoužívanější metody díky své jednoduchosti a rychlosti. Pracuje obvykle s jasovou složkou obrazu tedy s šedotónovým obrazem. U každého pixelu obrazu porovnává jas pixelu a daný práh a podle nastavení funkce změní hodnotu jasu pixelu. Lze použít více prahů a rozdělit pixely do určitých skupin (segmentace) podle velikosti jejich jasu. Obvyklé použití je ovšem pouze s jedním pevně daným prahem, které popisuje následující rovnice.

$$g(x,y) = \begin{cases} 1 & f(x,y) \geq T \\ 0 & f(x,y) < T \end{cases} \tag{4.1}$$

Kde funkce $g(x,y)$ je výsledný obraz odpovídající masce po prahovaní podle hodnot jeho pixelů. Funkce $f(x,y)$ obsahuje hodnotu jasu daného pixelu. Hodnota T odpovídá hodnotě prahu.

Jsou další možnosti jak upravit prahování hodnoty masky. Výsledkem nemusí být pouze 1 nebo 0, ale v případě první možnosti může obrázek použít původní hodnoty pixelu místo jedničky nebo obráceně. Variant je zde opravdu mnoho, ale kterou jsem ještě nezmínil je adaptivní prahování. Zatím zvažuji použití jednoho stejného prahu pro celý obraz. Lze ovšem práh interaktivně upravovat podle průměrné velikosti jasu pouze v určité oblasti. Takto lze obsáhnout místa obrazu, která jsou například různě osvětlená.

Ukázka výsledku po prahování je na obrázku 4.1.



Obrázek 4.1: Výsledek po prahování s pevně daným prahem. Odfiltrovala se všechna místa s jasem vyšším než práh

4.3.2 Dilatace a eroze

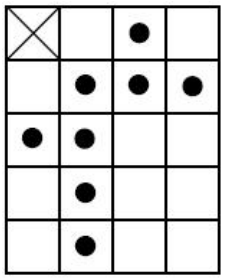
Dilatace a eroze patří do oblastí matematické morfologie opírající se o teorii bodových množin. Každý obraz lze totiž považovat za množinu bodů - pixelů. Pro názornost celé části a v rámci předpokládaného použití těchto operací budu zvažovat pouze binární (dvousložkový

- černá/bílá) obraz. kapitola je inspirována částí matematická morfologie z článku [17], odkud jsou i doprovodné obrázky.

Předpokládám binární obraz takový, že černý pixel je označen hodnotou jedna a bílý hodnotou nula. Vezměm si např. objekt, který je popsán množinou černých pixelů. Ve středu pozornosti dilatace a eroze je především tvar takového objektu. Pomocí těchto dvou operací je možno rekonstrovat porušený tvar objektu nebo obraz postižený drobným šumem. Taktéž lze tyto operace použít na zpracování objektu např. pro zjednodušení tvaru objektu a také jejich pomocí zdůraznit strukturu objektu - ztenčování, zesilování apod.

Binární obraz

Nechť množina X je množina objektů a odpovídá černým bodům na množině bodů obrazu. Každý prvek množiny X je popsán dvojicí (x, y) označující polohu prvku - bodu. Zbylé prvky obrazu, jež nepovažujeme za objekty, nazýváme pozadí. Binární obraz značím E^2 a lze si ho představit jako následující obrázek 4.2.



X = { (0, 2), (1, 1), (1, 2), (1,3), (2, 0), (2, 1), (3, 1), (4, 1) }

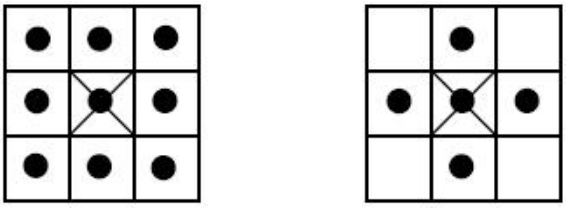
počátek (0, 0)

Obrázek 4.2: Vlevo je binární obraz a vpravo množina X obsahující objekty binárního obrazu.

Pixel obrazu označený křížkem z rohu do rohu je myšlený počátek. Na obrázku je nastaven na bod s označením $(0, 0)$, což jsou jeho souřadnice v počátku obrazu.

Realizace morfologické operace

Touto realizací je myšlena relace mezi množinou objektů z obrazu a menší bodovou množinou B, která se nazývá strukturní element. Strukturní element má definovaný střed, který ovšem nemusí ležet uprostřed. Taktéž nemusí být na tomto středu elementu černý bod - objekt. Příklady takových elementů jsou na obr. 4.3.



Obrázek 4.3: Typické strukturní elementy. Křížkem je označený myšlený střed elementu

Morfologickou operaci si představíme, jako bychom pohybovali strukturním elementem B systematicky po celém obraze. Bod obrazu, který se shoduje s počátkem souřadnic struk-

turního elementu, nazýváme okamžitý bod. Výsledek relace mezi obrazem a strukturním elementem zapíšeme do okamžitého bodu obrazu.

Dilatace

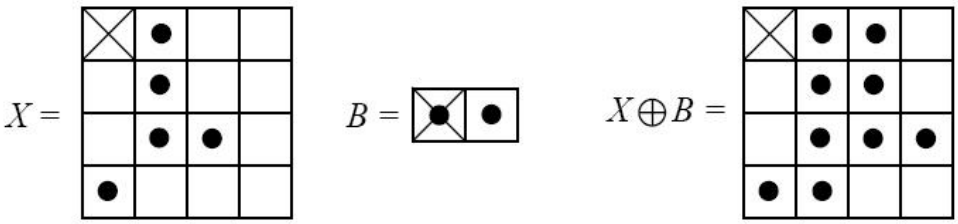
Operaci dilatace označím znakem $\oplus$. Tato operace skládá body dvou množin pomocí součtu souřadnic jejich prvků.

$$X \oplus B = \{d \in E^2, d = x + b, x \in X, b \in B\} \tag{4.2}$$

Příklad dilatace:

$$\begin{aligned} X &= \{(0,1),(1,1),(2,1),(2,2),(3,0)\} && (4.3)\\ B &= \{(0,0),(0,1)\} && (4.4)\\ X \oplus B &= \{(0,1),(0,2),(1,1),(1,2),(2,1),(2,2),(2,3),(3,0),(3,1)\} && (4.5) \end{aligned}$$

Příklad popisuje obrázek 4.4.



Obrázek 4.4: Příklad dilatace

Při dilataci se nečastěji používají strukturní elementy s rozměry 3x3, obsahující všech 9 bodů osmiokolí. Při takové dilataci se okraje objektů zvětší o jeden bod. Díry a zálivy do maximální velikosti 2 bodů se zaplní.

Eroze

Operaci eroze označím znakem $\ominus$. Tato operace skládá body dvou množin pomocí rozdílů souřadnic jejich prvků.

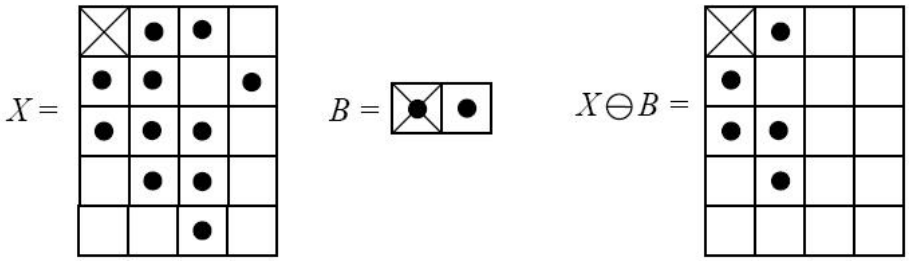
$$X \ominus B = \{d \in E^2, d + b \in X, \forall b \in B\} \tag{4.6}$$

Příklad eroze:

$$\begin{aligned} X &= \{(0,1),(0,2),(1,0),(1,1),(1,3),(2,0),(2,1),(2,2),(3,1),(3,2),(4,2)\} && (4.7)\\ B &= \{(0,0),(0,1)\} && (4.8)\\ X \ominus B &= \{(0,1),(1,0),(2,0),(2,1),(3,1)\} && (4.9) \end{aligned}$$

Příklad popisuje obrázek 4.5.

Při erozi se nejčastěji používají strukturní elementy s rozměry 3x3, obsahující všech 9 bodů osmiokolí. Při takové erozi se okraje objektů zmenší o jeden bod. Osamělé body do maximální velikosti 2 bodů nebo čáry o tloušťce 2 body zmizí. Možno použít na odstranění šumu.



Obrázek 4.5: Příklad eroze

4.3.3 Metody „zamalování“

Metody zamalování je můj volný překlad z angličtiny tzv. „Inpaint Methods“. Tyto metody slouží obecně k restauraci obrazu. Jejich úkolem je odstranit díry, škrábance a podobné vady z obrazu odhadnutím pixelů, které byli poškozeny. V případě mého použití se použijí na odhad pixelů mapy, které byly přepsány textem.

Metoda zamalování od A. Telea

Více o této metodě je v [13]. Pro úplnost zde uvádím princip této metody. Jedná se o poměrně rychlou metodu, která svými výsledky dosahuje kvalit podobných metod zamalování, které jsou ovšem pomalejší. Pracuje na základě této funkce:

$$I_q(p) = I(q) + \Delta I(q)(p - q) \tag{4.10}$$

Kde $I(p)$ je hodnota jasu pixelu na bodě p, $\Delta I(q)$ je hodnota určující postup změny jasu v okolí bodu q. Bod p v rovnici je bod na okraji oblasti, kterou zamalovávám a bod q bodem v oblasti blízké bodu p ovšem takové, kde známe hodnoty pixelů.

Jelikož takových bodů je více, spočítá se hodnota výsledného pixelu na bodě p takto:

$$I(p) = \frac{\sum_{q \in O(p)} w(p,q)[I(q) + \Delta I(q)(p - q)}{\sum_{q \in O(p)} w(p,q)]} \tag{4.11}$$

Kde $O(p)$ je známá oblast bodu p a $w(p,q)$ je váha vztahu mezi p a q. Je pochopitelné, že větší váhu budou mít body blíže bodu p. Postupně se směrem od hranic celé oblasti zaplní. Při tom dochází k propagaci jasových změn ve známém okolí do zaplňované oblasti.

Filtrace mediánem

Tato filtrace není přímo metodou pro zamalovávání oblastí, nicméně na jednotlivé pixely použitelná je. Taktéž je použitelná pro předzpracování, což vysvětlím v návrhu. Medián se často používá k odstranění šumu a oblíbený je především díky své jednoduchosti. Algoritmus výpočtu mediánu je popsán následujícími třemi kroky jež se aplikují na všechny pixeli $[i, j]$ obrazu A:

1. Načti body (pixely) z intervalu $[i-k, j-k][i+k, j+k]$ do pole M délky $l = (2k+1)^2$
2. Seřaď pole M
3. Výstupní obraz $B[i, j] = M[(l - 1)/2]$ tedy vzniká tak, že se pro každý jeho pixel použije střední hodnota z oblasti o poloměru k okolo pixelu.

Oblasti okolo pixelu mohou mít různé tvary. Obecně medián ničí ostrý obraz, např. naprostým odstraněním tenkých kontrastních linií.

4.4 Detekce v obraze

Tato část je teoretická a obsahuje vybrané metody detekce objektů v obraze. Rozděluji ji na tři hlavní části:

- Metody založené na hledání podle barvy nápisu.
- Metody na detekci hran.
- Detektory objektů.

Postupně popisuji všechny tyto způsoby detekce, vybírám metody, jejichž možné použití popisuji v návrhu.

4.4.1 Metody založené na hledání podle barvy

Na daném obraze se najdou všechny pixely jejichž barevný odstín leží v určitém rozmezí. V grafickém programu Photoshop se dá tato metoda simulovat všem známou kouzelnou hůlkou. Přesněji by šla tato metoda popsat následujícím postupem:

1. Zadá se referenční odstín pomocí diskretizovaného barevného modelu RGB.
2. Pixel po pixelu se prochází celý obraz a hledají se podobně barevné pixely. Podoba může být upravitelná pomocí větší či menší tolerance barevného odstínu. Hledání lze omezit např. hledáním pouze sousedních pixelů.
3. Pomocí všech označených a nalezených pixelů lze detekovat objekt barevně odlišný od okolí.

Více podobných metod lze získat odvozením z předešlého postupu. Varianty se mohou lišit např. jiným způsobem použití tolerance. (tolerance pro každý barevný odstín, pevná tolerance pro všechny odstíny atd.)

4.4.2 Metody na detekci hran

V této podkapitole se po popisu konvoluce budu zabývat nejznámějšímy hranovými detektory. Konvolucí se zabývám, protože je základem dalších popsaných filtrů na detekci hran.

Konvoluce

Jelikož lze obraz považovat za dvou-rozměrný signál, je možné na něj použít skládání signálů neboli konvoluce. V praxi se toho využívá např. k filtraci obrazu. Nechť signál $I_{i,j}$ odpovídá signálu vstupního obrazu, $h_{i,j}$ odpovídá signálu filtru, což je většinou matice čísel o rozměrech $2k+1$ a $V_{i,j}$ popisuje výstupní obraz po filtrování pomocí konvoluce. Znak $\bullet$ odpovídá operaci konvoluce. Pak platí:

$$V_{i,j} = I_{i,j} \bullet h_{i,j} = \sum_{x=-k}^{k} \sum_{y=-k}^{k} I_{i-x,j-y} h_{i,j} \tag{4.12}$$

Je-li konvoluce uplatněna pouze jednou, lze mluvit o lineární transformaci obrazu. Často se však konvoluce použije dvakrát po sobě, např. při hledání hran. Existuje více možností jaký druh konvolučních jáder pro hledání hran v obraze použít.

Sobelův filtr - konvoluční jádra

Sobelův filtr je jeden z nejznámějších detektorů hran. Používá dvou po sobě jdoucích konvolucí s jádry:

$$h_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad h_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Robertsův filtr - konvoluční jádra

Zde je filtr podobný tomu Sobelovu. Využívá tyto konvuluční jádra:

$$h_1 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad h_2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Prewittové filtr - konvoluční jádra

Prewittův filtr používá dvou po sobě jdoucích konvolucí s jádry:

$$h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

Kirschův filtr - konvoluční jádra

Kirshův filtr používá dvou po sobě jdoucích konvolucí s jádry:

$$h_1 = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 3 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & -5 & 3 \end{bmatrix}$$

Laplaciánův filtr - konvoluční jádro

Laplacián využívá pouze jedno konvoluční jádro, toto jádro se liší podle toho, jaké okolí vyhodnocovaného bodu budeme brát v potaz. Jedno z těchto jader pro 4-okolí má tvar:

$$h_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

4.4.3 Detektory objektů

Tato část obsahuje teoretický úvod k detekci, vysvětlení základních používaných pojmů a vztahů. Popisuje především detektor navrhnutý Paulem Violem a Michaelem Jonesem.

Detektory objektů lze použít na rozpoznání objektů, majících určité vlastnosti nebo příznaky. Existuje více základních typů metod na rozpoznání objektů v obraze např.:

- Strukturální rozpoznávání
- Příznakové (statistické) rozpoznávání

Detekce objektů - základní pojmy

V detekci objektů se používají pro detektory tyto pojmy:

- Míra správně detekovaných objektů (DR)- udává se v procentech a popisuje, kolik procent z hledaných objektů na obraze detektor nalezl podle následujícího vzorce:

$$DR = \frac{n}{p} \tag{4.13}$$

  Kde n je počet správně detekovaných objektů na obraze, p je celkový počet hledaných objektů na obraze.

- Míra chybného přijetí (FAR)- Používá se pro popis detektorů a popisuje kolik z nalezených objektů, nebylo hledanými objekty. Podle vzorce:

$$FAR = \frac{fn}{n} \tag{4.14}$$

  Kde n je počet nalezených objektů a fn je počet chybných nálezů. V této práci používám tento pojem také k obecnému označení detekcí, na které nebyl detektor určen.

- Chybné nepřijetí (FRR)- popisuje kolik objektů mělo být detekováno, ale nebylo. Udává se v procentech a počítá podle vzorce:

$$FRR = \frac{p - n}{p} \tag{4.15}$$

  Kde n je počet nalezených a správně detekovaných objektů a p je celkový počet hledaných objektů na obraze.

Strukturální rozpoznávání - obecný popis

Strukturální metody pracují na základě přesně definovaného klasifikačního stromu. Objekty lze popsat řetězcemi z abecedy a pomocí gramatiky je zpracovávat. Laicky popsáno obraz se přesně nasegmentuje a podle přesně daných pravidel se jednotlivé objekty zařadí do správné třídy. Často se tato metoda používá pro rozpoznání dvou-rozměrných známých útvarů. Viz také [10].

Kromě parametrů jednotlivých objektů v obraze se pracuje i s relacemi mezi objekty. Taková relace může být např. dotek, překrytí, ohraničení atd.. Tyto relace je pak možno hiearchicky uspořádávat do rozhodovacích grafů popsatelných gramatikami.

Mezi strukturální metody patří např. Houghova transformace, která kromě jiného hledá a rozpoznává geometrická primitiva v obraze. Viz dále.

Houghova transformce Houghova transformace je metoda pro nalezení parametrického popisu objektů v obraze. Při implementaci je třeba znát analytický popis tvaru hledaného objektu. Proto je tato metoda používána pro detekci jednoduchých objektů v obraze jakou jsou přímky, kružnice, elipsy, atd. Houghova transformace je používána především pro segmentaci objektů, jejichž hranice lze popsat jednoduchými křivkami. Hlavní výhodou této metody je robustnost vůči nepravidelnostem a porušení hledané křivky. Citováno z [4].

Příznakové (statistické) rozpoznávání - obecný popis

V těchto metodách je objekt popsán příznaky a podle pravděpodobnosti, jaký příznak mají objekty v určité třídě (skupině shodných objektů) se tyto objekty klasifikují a rozdělují do tříd. Klasifikaci provádí klasifikátor, který by měl správně detekovat a rozpoznat hledaný objekt.

Pro správnější extrakci příznaků lze užít učení.

Detektor Paula Viola a Michaela Jonese

Tento detektor objektů patří do skupiny příznakových metod. Původně byl vytvořen pro detekci obličejů v reálném čase. Text je inspirován článkem [14], odkud pochází i některé obrázky.

Tento detektor pracuje pouze s obrazem v šedých odstínech, tedy pouze na základě jasu jednotlivých pixelů. Barevný pixel se na převede na tzv. „stupně šedi“. Převod je popsán v kapitole o úpravách obrazu.

Jelikož tento detektor patří do tzv. „příznakových“ metod, pracuje s příznaky. Původní anglický název příznaku je „Feature“, což by se dalo přeložit i jako „rys“ ve významu vlastnosti (např. povahový rys). Víc se těmto příznakům budu věnovat později. Taktéž tento detektor pracuje na speciálně předpřipravených obrazech, čemuž se věnuje následující část.

Předpříprava obrázku mapy V anglickém článku ([14]) tento předpřipravený obraz nazývají „Integral Image“. Do češtiny název nepřekládám. Před použitím tohoto detektoru je potřeba šedotonový obraz(převod z RGB popsán výše) převést právě na tento předpřipravený obraz.

Jelikož příznaky, kterým bych se věnoval později, jsou určovány z obdelníkových výřezů původního obrazu, je tato následující reprezentace pro jejich určování velmi výhodná.

Každá buňka v umístění odpovídající umístění pixelu obrazu na souřadnicích x, y obsahuje součet hodnot jasu pixelů zároveň nad ní a nalevo od ní. Matematicky to lze popsat takto:

$$b(x,y) = \sum_{u<=x, v<=y} p(u,v) \tag{4.16}$$

Funkce $b(x,y)$ odpovídá hodnotě v předpřipraveném obraze na souřadnici $[x,y]$, funkce $p(u,v)$ odpovídá hodnotě pixelu na souřadnici $[u,v]$.

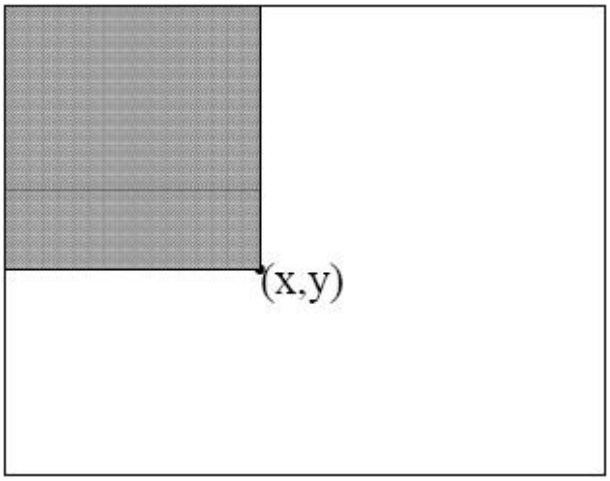
Při výpočtu v počítači je použit rekurentní vzorec, který umožňuje spočítat požadovaný obraz jedním průchodem.

$$s(x,y) = s(x,y-1) + p(x,y) \tag{4.17}$$

$$b(x,y) = b(x-1,y) + s(x,y) \tag{4.18}$$

Funkce $s(x,y)$ odpovídá součtu pixelů v řadě. Celý vzorec říká, že další buňku spočítáme, když vezmeme pixel odpovídající buňce a k němu přičteme hodnotu buňky nad ním a vlevo od něho. Tímto způsobem se vytvoří celý předpřipravený obraz.

Příznaky Detekce probíhá na základě poměrně jednoduchých rysů - příznaků. Použití těchto rysů má oproti porovnávání v rámci jednotlivých pixelů dvě výhody. První a důležitější je, možnost lepšího vytvoření obecného klasifikátoru. Jinými slovy lze mnohem lépe postihnout požadovanou podobnost, narozdíl od porovnávání pixelů, kde se spíše pracuje



Obrázek 4.6: Hodnota buňky v předpřipraveném obraze na souřadnicích $[x, y]$ je součet všech hodnot pixelů nalevo a nad touto souřadnicí

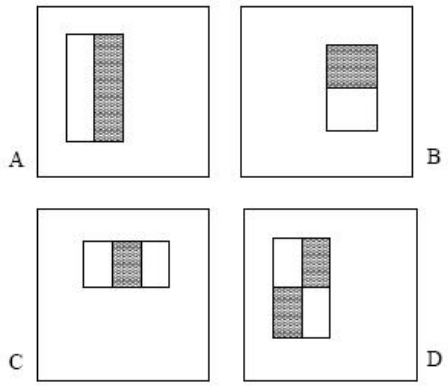
se shodností. Klasifikátor pak v konečné fázi dokáže lépe rozpoznávat objekty, které jsou pouze podobné objektům, na které byl natrénován (trénování klasifikátoru se budu věnovat později). Druhou výhodou je, rychlost testování. Pokud by se měli objekty se vzory porovnávat pixel po pixelu zabralo by to mnohem více času. Takto leze poměrně velké plochy obrazu vyloučit z vyhledávání mnohem dříve. V příznacích se využívá rozdílu v součtech jasů v určitých oblastech. Existuje více základních rozložení oblastí, podle nichž se počítají rozdíly. Na obrázku 4.7 je několik typů příznaků. Ve vylepšené verzi existuje i více příznaků např. natočených nejen do svislých a vodorovných poloh.

Vezmeme-li v potaz vzorové objekty, které jsou zpravidla 24x24 což je rozlišení detektoru, poté několik typů příznaků a spoustu variant jejich umístění je možných příznaků velmi mnoho (přes 45 tisíc).

Právě zde v počítání příznaků se vyplatí naše pokročilá reprezentace obrazu, které jsme dosáhli v předpřípravě. Předpokládejme, že chceme vypočítat hodnotu pole D na obrázku 4.8. Popis výpočtu je pod obrázkem a je vidět, že ho bylo dosaženo pouze čtyřmi operacemi. Tedy s velmi malou výpočetní složitostí.

Průběh skenování Výběrem správných rysů určující objekt se budu zabývat v části o učení. Předpokládejme, že máme několik příznaků - rysů, které popisují objekt. Vyhledávacím okýnkem se prohledá celý obraz a zjišťuje se podobnost příznaků popisujících hledaný objekt s odpovídajícími příznaky zjištěných z vyhledávajícího okna.

Hledané objekty se na obraze vyskytují v různém měřítku tedy v různých velikostech. Většina klasických vyhledávačů toto řeší tak, že postupně mění velikost obrazu, na kterém se objekty hledají. Vypočítat ovšem několik desítek zmenšenin obrazu je dosti výpočetně náročná operace, jedná se o podvzorkování, které lze počítat velmi dlouho. Přístup popisovaného vyhledávače je jiný. Nevytváří celou piramidu velikostí obrazu, ale zvětšuje vyhledávací okénko a mění pouze velikosti příznaků a jejich rozdílových hodnot. Tento výpočet je pak už docela rychlý oproti zmenšování skutečného obrazu.



Obrázek 4.7: Příklad obdélníkových příznaků zobrazených na tzv. vyhledávajícím okénku. Součet pixelů ležících uvnitř bílých obdélníkových částí v rámci příznaku ve výhledávacím okénku, je odečten od součtu pixelů v šedých obdélnících. Zde jsou 4 typy příznaků. Na obrázcích (A) a (B) jsou tzv. dvou-obdélníkové příznaky, zaměřující se na hrany, na obrázku (C) je troj-obdélníkový příznak zaměřující se na čáry a na obrázku (D) je čtyř-obdélníkový příznak sledující diagonální změny součtu.

AdaBoost Ještě než se začnu věnovat učení tohoto detektoru, je potřeba vysvětlit pojem AdaBoost. Citováno z [16]. AdaBoost při klasifikaci lineárně kombinuje rozhodnutí několika „jednodušších“ klasifikátorů a potencionálně tak dosahuje lepších výsledku, než by bylo možno dosáhnout použitím pouze jednoho klasifikátoru samostatně.

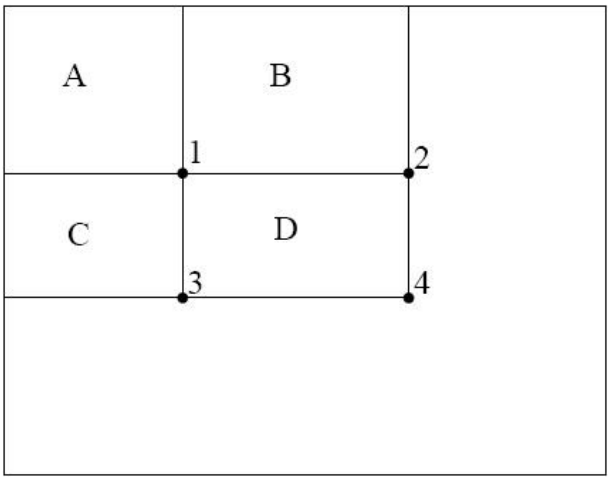
V tomto popisovaném detektoru se k natrénování klasifikátoru využívá trénování.

Učení klasifikátoru Jelikož možných příznaků lze získat mnohem více než pixelů v obraze (příznaky se týkají vztahů mezi pixely a tedy jich může být mnoho násobně víc), je potřeba vybrat několik důležitých příznaků, které co nejlépe postihují hledaný objekt. Z těchto několika příznaků se vytvoří klasifikátor. Podle předpokladu ve článku [14] opravdu stačí popsat objekt několika málo příznaky, problémem je, jak vybrat ty správné. Výběr nejlepších příznaků se provádí dohromady s vytvářením klasifikátoru pomocí metody AdaBoost.

Na začátku trénování klasifikátoru je potřeba mít poměrně velkou sadu pozitivních obrázků, jsou to ty, na kterých se vyskytuje hledaný objekt, a sadu negativních obrázků, kde se objekt nevyskytuje. Při trénování se zkoumají jednotlivé možné příznaky a hledá se ten, který nejlépe rozděluje pozitivní obrázky od negativních. Použitím jednoho příznaku vytváříme tzv. slabý klasifikátor, lineární kombinací těchto slabých klasifikátorů vzniká výsledný klasifikátor. Nechť $h_j(x)$ je funkce popisující slabý klasifikátor. Má pouze dvě možné hodnoty 0 a 1, které buď znamenají, že slabý klasifikátor objekt na obrázku x zamítl (v případě hodnoty 0) nebo přijal (v případě hodnoty 1). Pro upřesnění, přijetím myslím tu variantu, kdy se posuzovaný objekt porovnává s hledáným objektem a je určen jako podobný.

$$h_j(x) = \begin{cases} 1 & if\ p_j f_j(x) < p_j t_j \\ 0 & jinak \end{cases} \tag{4.19}$$

Funkce popisuje fungování slabého klasifikátoru. Hodnota f_j je hodnota příznaku. V našem



Obrázek 4.8: Příklad počítání součtu pixelů v poli D. Hodnota v místě označeném 1 na předpřipraveném obraze je součet pixelů v poli A, hodnota s označením 2 je součet pixelů v polích A a B, hodnota 3 je obdobně součtem polí A a C a hodnota 4 součtem A, B, C a D. Součet pixelů v poli D lze vypočítat z hodnot na souřadnicích označených 1, 2, 3 a 4 pomocí vzorce (4)+(1)-((2)+(3)), kde hodnota (x) je hodnotou na souřadnici označenou x

případě se jedná právě o rozdíl součtu jasu pixelů mezi dvěma či více oblastmi, tak jak jsem to již dříve popisoval. Index j označuje o který klasifikátor a příznak - o který rys se jedná. Hodnota t_j je tzv. „prahová hodnota“. Tato hodnota se taktéž získá učením. Jde o to ji správně nastavit tak, aby klasifikátor co nejlépe odděloval pozitivní vzory od negativních. Příznak (f_j) se porovnává s prahem (t_j) a podle toho, jak je to pro úspěšnost klasifikátoru výhodné musí být buď větší či menší. Hodnoat p_j označuje paritu, nabývá hodnot -1 a 1 a rozhoduje o směru porovnávacího operátoru. V případě, že klasifikátor přijímá objekt pokud jeho příznak je větší než daný práh a pokud použijeme vzorec 4.19, je parita pochopitelně záporná a v důsledku jen otáčí znaménko nerovnosti.

Celý algoritmus Ada-boost, tedy jeho varianta pro tento detektor, je popsán dále:

Algoritmus Ada-boost

- Vstupem algoritmu jsou dvě sady obrázků, jedna pozitivní a druhá negativní. Každý obrázek je popsán dvojící (x, y). Všechny obrázky pak posloupností $(x_1, y_1), \ldots, (x_n, y_n)$, kde n je celkový počet obrázků použitých k učení, x je daný obrázek a y je buď 1 resp. 0 pokud se na obrázku hledaný objekt vyskytuje a je tedy pozitivní, resp. pokud je negativní.

- Každý obrázek z testovací sady má tzv. „váhu“ $w_{1,i}$, kde první index značí pořadí t popisující, kolikátý slabý klasifikátor se používá.

- Inicializace vah - každá váha se nastaví na hodnotu $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$, kde m resp. l je počet negativních obrázků resp. pozitivních obrázků.

- For $t = 1, \ldots, T$:

  1. Normalizace velikosti vah:

  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

  2. Každému příznaku j, se natrénuje klasifikátor h_j, který používá pouze tento příznak a vypočítá se jeho chyba ϵ_j v závislostech na vahách w_t podle vzorce:

  $$\epsilon_j = \sum_{i=1}^{n} w_i |h_j(x_i) - y_i|$$

  3. Vybere se klasifikátor h_t s nejnižší chybou ϵ_t.
  4. Přepočítají se váhy jednotlivých obrázků podle vzorce:

  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

  Kde $e_i = 0$ pokud je obrázek x_i pozitivní a $e_i = 1$ pokud je obrázek negativní. Hodnota β je podle vzorce:

  $$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$$

- Konečný silný klasifikátor $H(x)$ je získán ze slabých klasifikátorů:

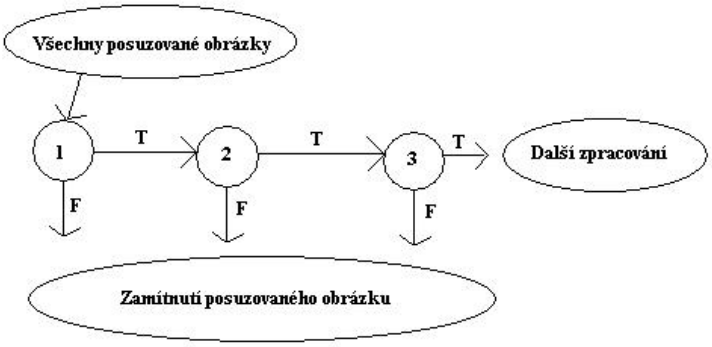
$$H(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & jinak \end{cases}$$

Celý popsaný algoritmus může být nejasný, proto ho zde ještě popisuji. V podstatě postupuje tak, že najde pokud možno co nejlepší klasifikátor pro ohodnocené obrázky. Potom vyhodnotí celou testovací sadu a změní váhy obrázků tak, aby se další klasifikátor především vybral podle toho, jak umí klasifikovat obrázky nesprávně klasifikované předešlými klasifikátory.

Vytváření rozhodovacího stromu Anglický originál článku [14] popisuje tzv. „Cascade“, kterou jsem přeložil jako rozhodovací strom. Rozhodovací strom použitý v tomto případě, je struktura, která vezme určitý objekt a postupně ho hodnotí a klasifikuje.

Rozhodovací strom se používá k urychlení detekce, vychází ze tří úvah, první je, že

na obrázku, kde je potřeba vyhledat určité objekty, jsou tyto objekty v menšině. Tedy existují obrovské plochy obrazu, kde se hledaný objekt nevyskytuje, čili je možné poměrně velké oblasti vyřadit z testování dřív a ušetřit tím výpočetní výkon. Druhá úvaha se týká použitého prahu v algoritmu AdaBoost. Výsledný prah silného výsledného klasifikátoru je $\sum_(t = 1)^T \alpha_t$, pokud by byl prah o něco nižší zvýšil by se výrazně počet správně detekovaných objektů (DR), naopak by se ale také zvýšil počet chybně detekovaných objektů (FAR). Poslední úvaha se věnuje slabým klasifikátorům. S využitím již popsaného snížení prahu, lze u prvních nalezených klasifikátorů dosáhnout velmi vysoké DR blízké 100% za cenu poměrně vysoké FAR např. kolem 40% což v důsledku znamená, že se zamítne kolem 60% klasifikovaných obrázků (v tomto případě klasifikovaným obrázkem myslím výřez v hlavním obraze, kde právě probíhá detekce objektu). Strom popisuje obrázek 4.9.



Obrázek 4.9: Obrázek popisuje rozhodovací strom při zpracovávání obrázku. Posuzovaným obrázkem se myslí oblast - vyhledávací okénko na hlavním obrazu, kde právě probíhá detekce, stavy 1, 2 a 3 odpovídají skupinám slabých klasifikátorů.

Princip stromu je následující. Vezme se okénko, ve kterém probíhá detekce, a minimální skupina slabých klasifikátorů, které však dohromady dosahují velmi velké DR i za cenu velké FAR. Pokud okénko - obrázek projde, na obrázku 4.9 tomu odpovídá hrana T, je posuzován další skupinou slabých klasifikátorů, ovšem nižší FAR a bohužel i DR. Takto se detekce zpřesňuje až po poslední, nejpřísnější skupinu klasifikátorů, která rozhodne, zda se jedná o hledaný objekt. Pokud některý stav (čili jeho skupina klasifikátorů) daný obrázek zamítne odchází se ze stromu hranou označenou F a posuzovaný obrázek se zavrhne a dále nezpracovává. Zde je právě princip zrychlení pomocí stromu, protože většina obrázku hledaný objekt neobsahuje.

Nalezené pozitivní objekty, které byly detekovány tak museli projít celý strom. Pokud by většinu obrazu tvořily hledané objekty, rozhodovací strom by se nemusel časově vyplatit, protože každý nalezený objekt by byl testován celým stromem, tedy pradĕpodobně i vícekrát stejným klasifikátorem ovšem s jiným prahem, což by mohlo výsledek zpomalit.

Dle mého názoru by zajímavým testem tohoto detektoru bylo, hledat určitý fraktál v jeho fraktálové grafice. Toto je ovšem mimo obor této práce, takže tuto myšlenku nebudu dále rozvíjet.

Až do teď jsem považoval slabý klasifikátor za klasifikátor, který používá pouze jeden příznak, což tak nemusí být. Slabý klasifikátor má pouze horší výsledky než konečný silný klasifikátor a příznaků může obsahovat více.

Jak už jsem naznačil celý strom se skládá z takzvaných „stavů“ (přeloženo z anglického

„stages“), které mohou obsahovat určitý počet klasifikátorů, postavených na určitých příznacích, spojených do jednoho silnějšího klasifikátoru s určitým prahem. FAR tedy chybějící nálezy některých hledaných objektů, lze teoreticky spočítat z následujícího vzorce:

$$FAR = \prod_{i=1}^{K} f_i \tag{4.20}$$

Kde K je počet klasifikátorů a f_i je FAR jednotlivých klasifikátorů. DR celého rozhodovacího stromu se získá analogicky, tedy ze vzorce:

$$DR = \prod_{i=1}^{K} d_i \tag{4.21}$$

Kde K je počet klasifikátorů a d_i je DR jednotlivých klasifikátorů. Zde dochází ke komplikaci, neboť při větším počtu klasifikátorů může být DR velmi nízká, proto je nutné udržovat DR ve stavech co nejvyšší i za cenu vyšší FAR.

Natrénování rozhodovacího stromu je složitý těžko optimalizovatelný problém, ve kterém je nutné volit mezi rychlostí a kvalitou, které zde jdou proti sobě. Je potřeba optimálně zvolit počet stavů stromu, počet použitých příznaků n_i a práh v každém ze stavu stromu. Při optimálním nastavení by se měl použít minimální počet příznaků N, ovšem při zachování hodnot FAR a DR. Minimální počet příznaků N, který se použije, ovlivňuje rychlost detekce. Taktéž je nutné minimalizování použití stejného typu příznaku dvakrát, ovšem s jiným prahem. Tento optimalizační problém se řeší tak, že pro každý stav rozhodovacího stromu se zadá jeho přijatelná f_i, což je chybovost ve stejném smyslu jako FAR a d_i což má stejný význam jako DR. Každý stav se poté trenuje pomocí AdaBoostu a zvyšuje se jeho f_i a d_i přidáváním nových příznaků tak dlouho, dokud nedosáhnou požadovaných hodnot. Pokud celková teoretická chyba rozhodovacího stromu je větší než minimální požadovaná chyba na začátku, vytvoří se další stav, který má chybu více zmenšit.

Algoritmus vytváření rozhodovacího stromu

- Na začátku programu je nutné stanovit hodnoty f_i udávající maximální chybu každého stavu a d_i udávající minimální kvalitu detekce v každém stavu (odpovídá DR).
- Zvolí se F_{kon} což odpovídá maximální FAR celého rozhodovacího stromu.
- Množina P resp. N odpovídá množině pozitivních obrázků resp. negativních obrázků. Poté je nutná oddělená sada pro validaci.
- $F_0 = 1.0$ - počáteční nastavení celkové chyby stromu.
- $D_0 = 1.0$ - počáteční nastavení kvality detekce (DR).
- $i = 0$ poslední i odpovídá počtu stavů stromu.
- while $F_i > F_{kon}$
  1. $i \leftarrow i + 1$
  2. $n_i = 0$ vynulování počtu použitých příznaků použitých v jednom stavu
  3. $F_i = F_{i-1k}$
  4. while $F_i > f \times F_{i-1}$ (dokud se nedosáhne požadované minimální chyby stavu)
     (a) $n_i \leftarrow n_i + 1$
     (b) Pomocí množin obrázků P a N algoritmem AdaBoost se natrénuje klasifikátor využívající právě n_i příznaků.
     (c) Vyhodnotí se natrénovaný klasifikátor na validační sadě a zjistí se jeho F_i a D_i.
     (d) Sníží se $i$tého klasifikátoru tak, aby jeho DR byla minimálně $d \times D_{i-1}$. Změna prahu vyvolá změnu F_i.
  5. $N \leftarrow \{\}$
  6. Pokud je $F_i > F_{kon}$ vyhodnotí se celý strom na sadě negativních obrázků a do množiny N se přidají všechny obrázky, kde došlo k FAR tedy k označení negativního obrázku za pozitivní.

Použití detektoru Shrnul bych zde celý průběh, ačkoliv je většina etap už podrobně popsána výše. Skládá se ze dvou hlavních částí. První část je trénovací, vytvoří se a natrénuje rozhodovací strom pro hledání daných objektů. Poté nastává druhá část - použití detektoru. Ta probíhá následovně:

1. Nejdříve se obraz předpřipraví, tak aby šlo lépe počítat příznaky.
2. Skenuje se celý obraz vyhledávacím okénkem různých velikostí a pro každé okénko se vyhodnocuje rozhodovací strom. Vypočítávají se požadované příznaky v obraze a klasifikují se tak jednotlivá okénka.
3. Vyhodnotí se vícenásobné detekce což popisuji dále.

Vícenásobná detekce Poslední věcí, kterou bych na tomto navrhovaném detektoru před zhodnocením popsal, je řešení vícenásobné detekce. Může se stát, že se daný objekt detekuje vícekrát. Způsobeno to může být dvěma příčinami, první je, že objekt je detekován ve

vyhledávacích okéncích ležících velmi blízko sebe. Druhou příčinou mohou být malé změny měřítka hledaného objektu, kdy je jeden objekt určený dvakrát jakoby v různých velikostech. Filtrací pomocí vícenásobných detekcí, kdy označíme za pozitivní ty objekty, které se detekují např. víc jak dvakrát, lze snížit hodnotu FAR a bohužel i DR.

Závěr o detektoru Paula Violy a Michaela Jonese

Detektor dosahuje velkých rychlostí detekce. Existuje více variant tohoto detektoru, které pracují na podobném principu, ale liší se množinou příznaků. Existují vylepšení, kde se používají další typy příznaků.

Kapitola 5

Návrh

Zásadní a nejrozsáhlejší kapitola popisuje návrh možností jak odstranit nápisy z map. Kapitola popisuje možné využití metod popsaných v minulé kapitole na požadované zpracování nápisů na mapě.

Některé metody zde byly přidány až v průběhu implementace, protože průběžné výsledky vyžadovaly navrhovat další a další doplnění pro zkvalitnění výsledků. Kapitola je rozdělena na tři základní části. První se zabývá samotnou detekcí nápisů na mapě, druhá popisuje odstranění nápisů a třetí popisuje metody na znovuzaplnění míst po odstraněných nápisech. Snažím se zde popsat všechny varianty použití metod, které mě nebo někoho s kým jsem svoji práci diskutoval, napadly. V další kapitole o implementaci pak uvádím, které metody jsem vybral a implementoval.

Některé metody jsem testoval (s použitím Adobe Photoshop) už v průběhu návrhu a zamítl je předem, přesto je zde zmiňuji spíš jako nevhodné varianty.

Metodám, kterým se zde budu věnovat nejvíce, tedy především detekci objektů, kterou budu nakonec používat, se budu věnovat hlouběji než ostatním metodám.

Jak už jsem předestřel zpracování nápisů se skladá z těchto tří po sobě jdoucích částí:

1. Detekce nápisů na mapě - základní problém celé této práce je vůbec nápisy identifikovat.
2. Odstranění nápisů - jakým způsobem a jakými metodami nápisy odstranit.
3. Zaplnění místa po nápisech - návrh jak zaplnit místo po odstraněných nápisech.

Výsledkem by měla být knihovna funkcí pro zpracování textu na starých mapách Druhého vojenského mapování.

5.1 Návrh použití metod na nalezení nápisů na mapě

Tato část popisuje metody pro nalezení nápisů na mapě. Pro lepší přehlednost bych metody rozdělil do několika skupin. Na konci každé skupiny metod zhodnotím jejich použitelnost podle pokusů zpravidla z testovaní v Adobe Photoshop. Stejně tak u každé metody na konci popisu a ukázky výstupu této metody krátce zhodnotím, zda by mohla být pro cíl jakkoliv prospěšná.

5.1.1 Využití metody založené na hledání podle barvy nápisu

Metodu jsem si pro tento návrh otestoval pomocí programu Adobe Photoshop, kde ji simuluji kouzelnou hůlkou(viz obrázek 5.1). Výsledek je označen černobílou hranicí.

Obrázek 5.1: Výsledek po použití kouzelné hůlky ve Photoshopu

Zhodnocení Tato metoda je pro nalezení nápisů na mapě samostatně nepoužitelná, protože ve stejném barevném odstínu je na mapě více rozdílných objektů, které neodpovídají textu. Přesto by šlo tuto metodu využít buď pro přípravu obrazu, aby se zbytečně nehledaly nápisy tam, kde nejsou, nebo po vyhledání nápisu k jeho odstranění v rámečku ohraničující nápis. To už ovšem předbíhám, odtranění nápisu se budu věnovat až v další části návrhu.

Navíc v obrázku kde je více Lehmannových šraf dochází k naprosté degradaci výsledků, neboť tyto šrafy mají shodnou barvu s nápisy.

5.1.2 Návrh použití metod na detekci hran

Existuje více možností řešení problému nápisů na starých mapách, jedna možnost je najít písmo pomocí některého z hranových detektorů a pak ho na způsob OCR rozpoznat a odstranit. V této podkapitole se budu zabývat hranovými detektory a sledovat výsledky jejich použití na staré mapě.

Pokud mluvíme o OCR (obecný název pro programy rozpoznávající text)existují metody, které rozpoznávají text v obrázku např. některé spamové filtry. Pokud bude volně dostupný některý z těchto vyhledávačů bylo by zajímavé otestovat jeho schopnosti třeba i po jemné modifikaci jeho kódu na staré mapě.

V této části se zabývám možností detekovat text jako určitý shluk hran, který se jinde na mapě nevyskytuje. Zpravidla je totiž na mapě kolem písma mnohem více výrazných hran než kdekoliv jinde. Popisuji účinky nejznámějších filtrů popsaných v předešlé teoretické kapitole. Vždy výsledný pokus vytvořený s pomocí Photoshopu zhodnotím a navrhnu možnost jeho použití či nepoužití v rámci detekce nápisů.

Sobelův filtr

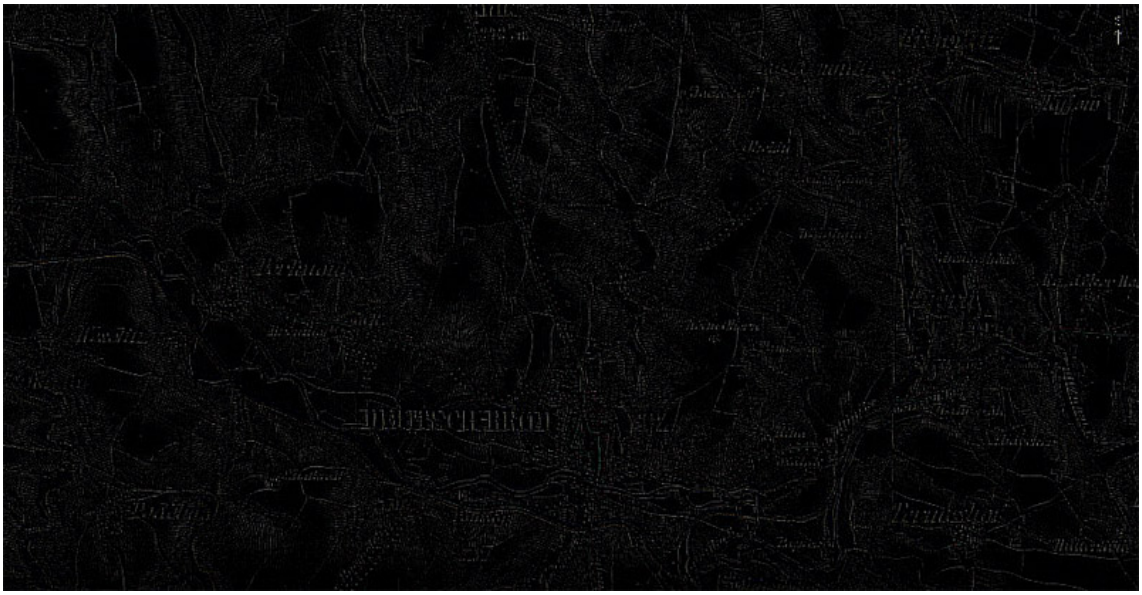
Výsledek Sobelovy filtrace je na obrázku: 5.2.

Obrázek 5.2: Hledání hran pomocí Sobelovy filtrace

Zhodnocení Sobelovy filtrace Nezdá se, že by použitím Sobelovy filtrace hrany nápisů zvýrazněly víc než než na okolních objektech mapy. Pro tento projekt je výsledek skoro nepoužitelný.

Robertsův filtr

Výsledek po filtraci na obrázku: 5.3.



Obrázek 5.3: Hledání hran pomocí Robertsova filtru

Zhodnocení Robertsova filtru Výsledky tohoto filtru nevypadají pro tento projekt použitelně. Robertsův filtr má spíše horší výsledky než filtr Sobelův, navíc je i více náchylnější k šumu v obraze. U starých map je šumu většinou více, čili pravděpodobně tento filtr nebude mít význam.

Prewittové filtr

Výsledek po filtraci na obrázku: 5.4.

Obrázek 5.4: Hledání hran pomocí Prewittové filtru

Zhodnocení Prewittové filtru Výsledky tohoto filtru nevypadají pro tento projekt použitelně.

Kirschův filtr

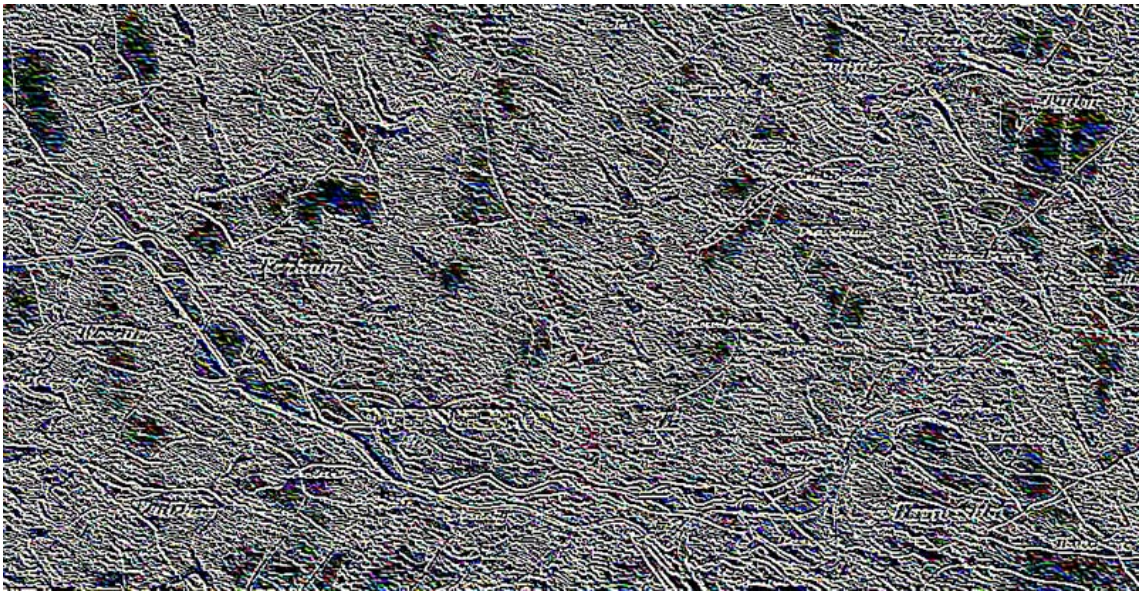
Výsledek po filtraci na obrázku: 5.5.

Zhodnocení Kirschova filtru Výsledky tohoto filtru nevypadají pro tento projekt použitelně.

Laplaciánův filtr

Výsledek po filtraci na obrázku: 5.6.

Zhodnocení Laplacianova filtru Nápisy jemně vystouply, při použití kouzelné hůlky lze částečně omezit množství nesprávných objektů. Spíše ale nepoužitelné.



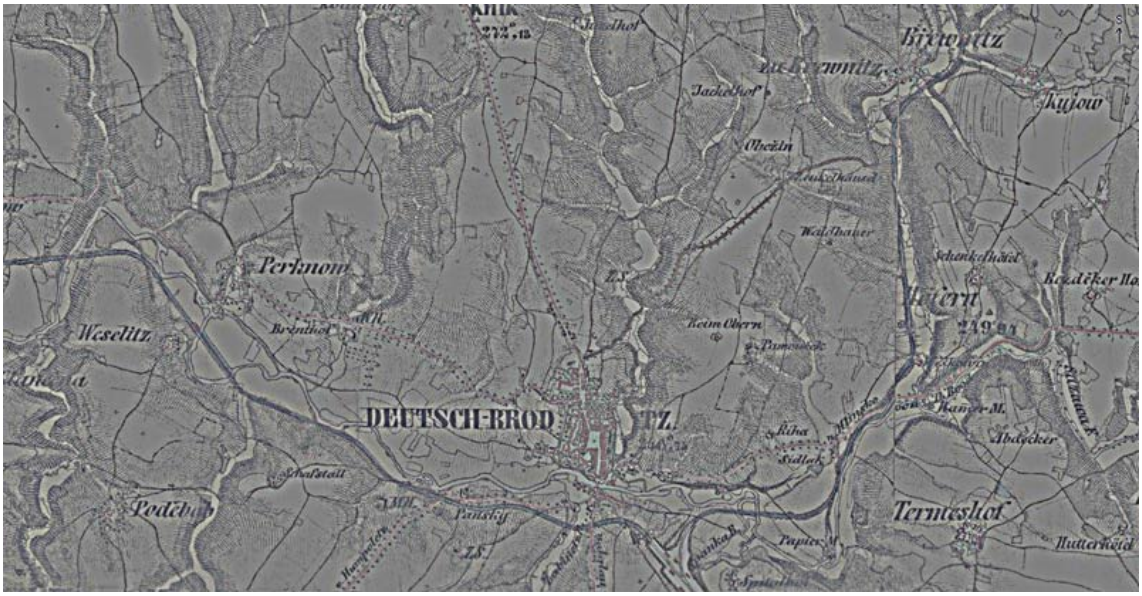
Obrázek 5.5: Hledání hran pomocí Kirschova filtru

Obrázek 5.6: Hledání hran pomocí Laplaciánova filtru

Horní propusť

Horní propusť lze opět simulovat ve Photoshopu, pokoušel jsem se najít konvoluční jádro jenž by simulovalo učínky filtru horní propusti ve Photoshopu, ale zatím bez úspěchu.

Tento filtr dává velmi zajímavé výsledky - viz obrázek 5.7, proto ho zde uvádím, bohužel bez další teorie.



Obrázek 5.7: Filtrace obrazu horní propustí

Zhodnocení horní propusti Jedná se opět o hledání hran, protože se filtrují vysoké frekvence obrazu, které se nacházejí právě na hranách. Tento filtr by bylo možné opět použít jako předpřípravu mapového materiálu pro pozdější zpracování. Výsledek je zajímavý také proto, že se na obrázku nevyskytují Lehmannovy šrafy, které jsou ovšem na mapě velmi často.

Složitější metody a zhodnocení předešlých metod

Cannyho detektor a Marr a Hildreth hranový filtr, které taktéž hledají hrany v obraze nepopisuji ani v předešlé teoretické kapitole. Po neúspěchu s předešlými metodami nevidím důvod testovat a zkoumat další detektory. Houghově transformaci, použitelné jako detektor hran, se budu věnovat v další podkapitole.

Po předešlých výsledcích se obecně zdá býti reálnější použití spíše detektorů objektů než rozpoznávače písma. Mapy totiž obsahují velké množství hran, které nejsou písmem a hlédání se tím dosti znesnadňuje. S detektory hran by spíše šlo pracovat při hledání cest, hranic lesů, polí atd..

Výsledky hledání nápisů pomocí hran mě už zezačátku zklamaly, a proto se jimi jako metodami na hledání nápisů dále nevěnuji a žádnou z nich používat nebudu, uvádím je zde spíše jako neúspěšný pokus.

5.1.3 Návrh na použití detektoru objektů

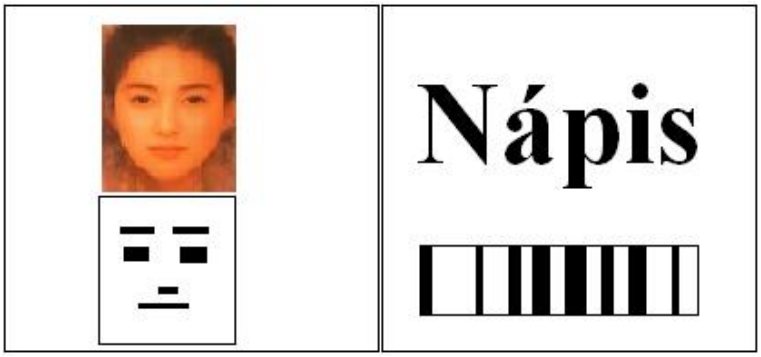
V této části analyzuji návrh na použití detektoru popsaného v minulé kapitole. Zaměřuji se na možnosti jeho fungování a na možné způsoby přípravy a tvorby takového detektoru.

Jak už jsem uvedl v teoretické části, detektory objektů lze použít na rozpoznání objektů, např. pomocí přínaků. Je potřeba se zaměřit na příznaky typické pro písmo na mapě. Jedním z nich je určitě šířka čáry, neboť je zpravidla silnější než u čar popisujících cesty. Taktéž určité pravidelné změny jasu jsou pro nápisy charakteristické a snad vyhledatelné.

Detektor Paula Violy a Michaela Jonese

V této části popisuji úvahu, proč by tento detektor mohl být úspěšný a jakými způsoby jej lze trénovat a připravit k použití.

V nápise se zpravidla, s určitou pravidelností, střídají místa s nižším a vyšším jasem. Je tedy možné upravit tento detektor na vyhledávání nápisů na mapě. Předešlou úvahu vysvětluje obrázek 5.8. Možná bude potřeba hledat každé písmeno samostatně, potom by se hledali speciální rozdíly jasu na každém písmeně zvlášť.



Obrázek 5.8: Obrázek vlevo ukazuje, jak tento detektor pracuje při hledání obličeje, zaměřuje se na rozdíly jasu zvláště v oční části obličeje. Levý obrázek zhruba nastiňuje funkčnost detektrou u textu.

Rád bych využil některou z implementací tohoto detektoru např. v rámci knihovny OpenCV (popsáno v kapitole Implementace). Viděl jsem výsledky tohoto detektoru při detekci obličejů, kde má výborné výsledky. Princip celého detektoru je vlastně velmi jednoduchý, proto asi také dosahuje velkých rychlostí detekce. Nevýhodou může být absence detekce barevných odstínu, se kterými detektor vůbec nepracuje. Některé plochy, kde se text nevyskytuje, by šly vyloučit pomocí detekce jejich barev.

Návrh testovací sady pro vytvoření jednoho/více detektorů Rozsáhlou částí by mohl být návrh trénovací sady pro vytvoření tohoto detektoru. Popisuji několik nápadů, které by se při implementaci daly použít.

- Pro každé písmenko by se vytvořil detektor. Trénovací sada pro každý detektor by byla vytvořena z dostatečně velkého počtu stejných písmen, vyříznutých z různých míst map. Negativní vzory by se vzali z co největšího počtu míst, kde se text nevyskytuje. Do negativních vzorů by také šly přidat všechny výřezy ostatních písmenek než je to hledané. Poté by detektory mohly číst i názvy ze starých map.

  Nevýhodou tohoto přístupu je náročnost vytvoření trénovací sady. Počet písmenek v sadě by měl, dle mých počátečních pokusů, mít alespoň 200 poz. obrázků. Tuto sadu je nutné vytvořit pro všechna písmena v abecedě. Poté je nezbytné vytvořit tuto sadu i pro velká písmena a pro písmena, které mají více variant tvarů. Vytvořit sadu 200-krát malé „e“ je jen otázkou času, ale vytvořit takovou sadu pro písmeno „q“ nebo „x“ či jiné málo obvyklé písmeno je velmi zdlouhavé.

- Vytvoří se trénovací sady, které postihují pouze vodorovné nápisy. Postihnout nápisy, které jsou v jiném směru nebo se např. vlní kolem řeky, je mimo možný záběr této

práce.

- Další možností vylepšení trénovací sady by bylo ji vytvářet výřezy, tedy tak jako v předešlém případě, ale z map, kde bylo provedeno prahování, takže by se snížila variabilita pozadí za nápisem. Prahování by šlo spojit i s ostatními navrhovanými metodami.

  Nevýhodou nebo spíše problémem je zvolení správného prahu. Navíc by se odstranila jasová informace obrazu, což by nakonec nemuselo výsledky ovlivnit správným směrem.

- Zdrojovou mapu by šlo pomocí např. eroze nebo konoluce „rozmáznout“ ve vodorovném směru. Z vodorovných nápisů by se vytvořili jakési tmavé obdelníky a testovací sada by obsahovala pouze obrázky s typickým tmavším místem uprostřed. Šlo by ji kombinovat i s prahováním z předešlého bodu. Úvahu popisuje obrázek 5.9.

Obrázek 5.9: Obrázek popisuje návrh úpravy testovací sady. V horní části obrázku je erozí upravený obraz a ve spodní jsou pravděpodobné typy příznaků použité pro detekci. Červené obdelníky na obrázku označují pozici textu na mapě.

  Nevýhodou této metody jsou opět Lehmannovy šrafy, které mohou takto upravený obraz zcela zmást. Metoda by byla úspěšná pokud by se text vyskytoval na výrazně světlejším pozadí.

- Trenovací sada by se opět vyřezávala z originálních map, ale jako hledané objekty by se se definovaly mezery mezi písmeny. Je to oblast, kde je uprostřed vyšší jas tak jak to popisuje obrázek 5.10. Vytvořila by se buď jedna trénovací sada nebo více sad pro vytvoření více detektorů s několika základními typy krajů mezi písmeny. Pro lepší pochopení opět odkazuji na obrázek 5.10.

  Nevýhodou detektoru této trénovací sady je, že nebude detekovat dlouhé nápisy, kde šířka mezi dvěma písmeny je i dvakrát širší než samotná písmena.

Obrázek 5.10: Obrázek popisuje návrh úpravy testovací sady vytvořené mezerami mezi písmeny. Popisuji několik možných typů mezer. Buď lze použít více variant a na každé vystavět detektor, nebo vše smíchat dohromady a vystavět detektor společný. Na obrázku není pozadí mapy, které by zde normálně bylo. Ve spodní části obrázku jsou některé vztahy mezi dvěma sousedícími písmeny. Podle nich by se vytvářely příznaky. Typický příznak pro toto vyhledávání není těžké odhadnout.

- Spojením prvního a předchozího nápadu by vznikla trénovací sada, která by vytvářela jeden detektor. Tato trénovací sada by obsahovala náhodný výběr z úseků textu. Pokud by byla tato sada velmi rozsáhlá, mohl by se detektor takové sadě přizpůsobit a detekovat text pouze na základě jakési pravidelné změny jasu v textu. Zčásti popsáno již dříve. Taktéž bych jako podvariantu uvedl sadu, kde budou všechna písmena dohromady. I ta má stejné nevýhody a dále se jí nemíním zabývat.

  Nevýhodou je těžká odhadovatelnost funkčnosti. Nasbírat několik tisíc vzorků a poté zjistit, že to správně nedetekuje by bylo nepříjemné.

- Poslední způsob vytvoření trénovací sady a z ní nacvičeného detektoru je vytvoření několika trénovacích sad a potažmo detektorů ze skupinek písmen. Skupinka písmen by vždy obsahovala alespoň trochu tvarově podobná písmena (např. malé „e“ a „o“). Dala by se tím ušetřit námaha při sbírání velkého počtu písmenek. Vytvořilo by se několik skupin, do kterých by se písmena podle tvaru rozdělila. Tento návrh bych asi použil nejraději ze všech.

  Nevýhodou této metody může být větší nespolehlivost oproti detektorům pro každé písmeno zvlášť.

Filtrace výsledků detektoru

Detektor zravidla oznámí souřadnice všech hledaných a nalezených objektů na obraze. Některé nálezy jsou ale chybné a proto je nutné je co nejlépe odfiltrovat. Při detekci textu bych chtěl aplikovat dvě takové filtrace. První se zabývá velikostí nalezených objektů - písmen, druhá pak jejich vzájemnou polohou.

Filtrace nálezů podle velikosti Protože se na mapě vyskytuje text, který má určitou maximální a minimální velikost, je možné všechny nálezy otestovat a zjistit, zda nalezený objekt je v rozmezí normální velikosti písma na mapě. Jde o dodatečnou klasifikaci pomocí velikosti nálezu.

Filtrace nálezů podle jejich vzájemné polohy Jelikož se každý nápis na mapě skládá z více písmen a detekované objekty jsou jednotlivá písmena, pomocí jejich vzájemných poloh by mělo jít odfiltrovat některé chybně nalezené objekty, které leží zpravidla osamoceně (viz příklad na obrázku 5.11).

Obrázek 5.11: Příklad chybné detekce, kterou lze odstranit pomocí testování vzájemných poloh nalezených objektů. Červené kolečko označuje nalezený objekt, modrá šipka ukazuje na chybně detekovaný objekt (domeček). Spodní část obrázku popisuje žlutou barvou oblasti okolo písmene, kde se hledá vedlejší písmeno.

Řešením je vyhledat skupiny písmen (svým způsobem jde o strukturální detekci)tvořících název ve vodorovném uspořádání. Tyto skupiny ponechat a odstranit všechny detekované objekty ležící samostatně nebo ve velmi malé skupině.

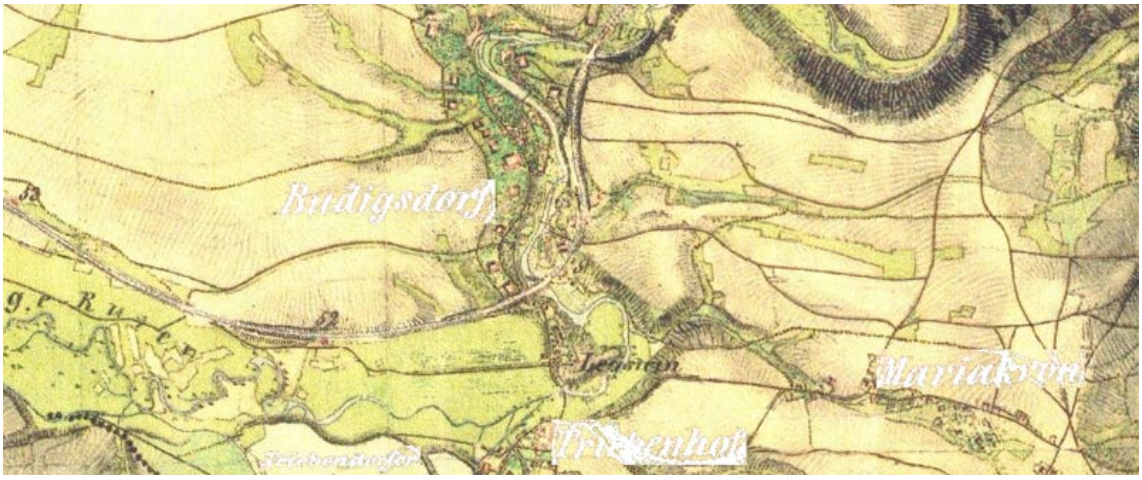
Vylepšením této metody by bylo také testovat malé skupiny písmen na délku. Například skupina se třemi nálezy by měla být minimálně dvakrát delší než vyšší. Poměr bude určen až při testování.

5.2 Návrh metod použitelných pro odstranění vyhledaných nápisů

Z detektoru vyjde pouze seznam souřadnic a velikostí objektů. Ideální by bylo, kdyby z detektoru vyšel přímo seznam pixelů obrázku mapy, kde se text nachází. Místo toho

detektor pouze ohraničí písmena čtverci. První variantou jak nápisy odstranit je, vymazat celé tyto čtverce. Jelikož písmeno zabírá v průměru tak třetinu čtverce ztratí se tímto způsobem dvě třetiny možná užitečné informace o nejbližším okolí písmene.

Druhou, lepší variantou je odstranit text ze čtverců pomocí prahování. Prahovaní je popsáno v kapitole o úpravách obrazu. Jelikož jsou všechny nápisy vytvořeny pixely s nižším jasem, stačí odfiltrovat všehny pixely čtverce s jasem nižším než pevně daný práh. Práh je pevný, protože jas nápisů na mapě je vesměs stejný. Velikost tohoto prahu se určí až při implementaci podle výsledků. Po odstranění písma vznikne na jeho ploše prázdná na obrázku 5.12 bílá oblast. Výsledek prahu by mohl být zhruba podle obrázku 5.12.



Obrázek 5.12: Takto by mohl vypadat výsledek po prahování - předběžný test návrhu ve Photoshopu

Pokud se pozorně zaměříme na okraje odfiltrovaných písmen, uvidíme tmavší okraje nápisu, který ovšem nebyl prahováním odstraněn. Předpokládám, že místo pod nápisem budu nahrazovat především odhadem z nejbližších okolních pixelů. Tyto tmavší pixely, které jsou pravděpodbně pozůstatkem rozpitého inkoustu při tisku nápisů, by poměrně hodně změnily výsledek zaplňování, proto je nutné je odstranit. To by se dalo udělat nižším prahem, čímž by se ale odstranily všechny pixely s jasem podobným těmto okrajům a zbytečně by se přišlo o další informace. Jelikož se jedná o okraje, je možnost využít na tyto bíle oblasti dilataci a roztáhnout bílé oblasti o jednu vrstvu pixelů. Tím by se měly tyto okraje nadobro odstranit. Výsledky dilatace jsou na obrázku až v rámci kapitoly výsledků a testování.

V rámci návrhu bych zmínil také úpravu předešlého postupu. Na mapách se často vyskytují některá tmavá místa hor nebo místa s velkým množstvím šraf. Jelikož prahování by tyto šrafy odstranilo a navíc by byla na pozůstatky aplikována dilatace, byly by odstraněné oblasti zbytečně velké. Proto bych rád použil ještě před dilatací určitý typ eroze. Vypočítal by se podíl oblasti odstraněné pouze prahováním a celé oblasti čtverce a pokud by byl větší než určitá mez, prošel by se pixel po pixelu celý obraz a zjišťoval by se vždy počet prahovaných pixelů v osmiokolí. Pokud by byl větší než daný počet, pixel by zůstal prahovaný tedy bílý, naopak pokud by nebyl, vrátila by se pixelu jeho původní barva. Tímto by se měli odfiltrovat některé zbytečně prahováním odstraněné oblasti, především tenké čáry cest a šrafy. Obrázek bude opět až v kapitole s výsledky.

5.3 Návrh metody na zaplnění místa po odstraněných nápisech

K zaplnění oblastí bych navrhoval použít metodu od A. Telea popsanou v části o úpravách obrazu. Poměrně zajímavě totiž propaguje jasové změny ve svém okolí do zaplňované oblasti. Její rychlost je dalším důvodem proč tuto metodu použít.

Jelikož některé odstraněné nápisy leží v místě s Lehmannovými šrafami dochází ke zmatení předešlé metody. Metoda totiž sleduje změny jasu v okolí odstraněné oblasti. Pokud jsou šrafy směrem kolmým ke směru zaplňování dochází k chybnému výpočtu pokračování průběhu jasu. Pro příklad postupujme ze známé oblasti přes obnovovaný bod po směru zaplňování oblasti. První známý pixel, který projdeme, je součástí Lehmannovy šrafy a je tedy tmavý, druhý pixel je už světlý a odhaduje se pixel třetí. Jelikož jas na tomto postupu se jen zvyšoval bude mít tento třetí pixel ještě větší jas než prošlé známé pixely. S dalším odhadnutým pixelem to bude podobné. Přitom ve skutečnosti by se tam místo světlejšího pixelu měl objevit tmavší pixel pravděpodobné Lehmannovy šrafy.

Tento problém odstraním velmi jednoduše. Na obraz, ze kterého se budou počítat zaplňované oblasti, aplikuji mediánový filtr s čtvercovými rozměry 3x3. Tím se šrafy rozmáznou a nebude v tak velké míře docházet k nevhodné propagaci zvyšování jasu do zaplňované oblasti. Obrázek se zaplněním, kde došlo k negativní propagaci barev zobrazím v kapitole implementace.

Kapitola 6

Implementace

V této to kapitole popisuji implementaci knihovny na práci s nápisy. Kapitola se dělí na čtyři následující části:

1. Nástroje použité při implementaci - popisuji zde, v čem knihovnu naprogramuji a pomocnou knihovnu pro práci s obrazy.
2. Hledání nápisů - popisuje již kapitolu z návrhu ovšem z pohledu konkrétní implementace. Součástí této části je i popis trénování detektoru.
3. Odstranění nápisů - postup odstranění a metody na doladění.
4. Zaplnění místa po nápisech.

6.1 Nástroje použité při implementaci

6.1.1 Knihovna OpenCV

Pro práci s obrazem plánuji použít knihovnu OpenCV, kterou dále popisuji v této kapitole, čerpal jsem i z [6].

Knihovna OpenCV (Open Source Computer Vision Library) od firmy Intel je volně použitelná knihovna s funkcemi pro práci s obrazem a videem. Zaměřuje se na real-time počítačové vidění, což v praxi znamená zaměření se na funkce jež se vykonávají v reálném čase. Zde je výčet základních druhů funkcí této knihovny:

- Funkce pro vytvoření obrazu a pro přístup do obrazů z externích zdrojů (třeba souboru).
- Lineární transformace obrazu.
- Filtrace obrazu.
- Funkce pro konverzi barevného prostoru obrazu.
- Zjištění histogramu a prahování obrazu.
- Filtrace obrazu.
- Transformace obrazu (rotace, posun, zoom, zkosení, převrácení....).

- Logické a aritmetické operace nad obrazem.
- Funkce pro identifikaci a rozpoznání objektů v obraze.
- Funkce na hledání hran v obraze.
- Funkce pro segmentaci obrazu.

Knihovna je volně dostupná na internetu (http://opencvlibrary.sourceforge.net/). Především používám detektor Paula Violy a Michaela Jonese implmentovaný v rámci této knihovny.

6.1.2 Implementační prostředí

Knihovnu vytvářím v prostředí Visual Studia 2005 od firmy Microsoft v jazyce C++. I přes toto prostředí by kódy měly být přenositelné do Linuxu, neboť nevyužívám žádné speciální funkce z .NET Frameworku. Taktéž OpenCV pracuje i pod Linuxem.

6.2 Hledání nápisů

V této části se zabývám hledáním nápisů na mapě. Zahrnuje dvě podčásti - jednak samotnou detekci pomocí detektoru, a také trénování tohoto detektoru. Využívám detektoru poskytovaného s knihovnou OpenCV, který je postaven na detektoru od Paula Violy a Michaela Jonese. Oproti původnímu návrhu však přidává další typy příznaků, které jsou ovšem také velmi dobře počitatelné z předpřipraveného obrazu.

6.2.1 Vytvoření detektoru

Vytvoření detektoru se skládá z několika dílčích úkolů, nejdřív je potřeba vytvořit pozitivní a negativní sadu obrazů pro natrénování detektoru. Poté je nutné zvolit parametry, které má mít rozhodovací strom, a se kterým bude detektor pracovat. Při vytváření jsem postupoval podle článku [5].

Vytvoření trénovacích sad

Jelikož mám čtyři poměrně velké mapové listy ve formátu tiff, zvolil jsem jeden z nich, označený GW_I_11. Označení dodávám, protože celý originální mapový list s rozměry 3840x3840 pixelů nemá smysl tisknout ani do příloh, je na přiloženém CD. Tento mapový list jsem použil k vytvoření trénovacích sad. Ostatní jsem využil k testování výsledků.

Obrázky z trénovacích sad jsou v příloze. Nyní uvedu výčet písmen, která jsem vyřízl z uvedeného mapového listu. Prováděl jsem čtvercové výřezy poměrně těsně okolo každého písmene. Násbíral jsem:

- Vzory malé a - 128 výřezů.
- Vzory malé b - 13 výřezů.
- Vzory malé c - 0 výřezů - tvarem bylo podobné písmenu „e“ a výsledky ukázaly, že to nemá smysl vyřezávat.
- Vzory malé d - 24 výřezů.
- Vzory malé e - 121 výřezů.

- Vzory malé f - 2 výřezy - víc jich na mapovém listu nebylo.
- Vzory malé g - 10 výřezů.
- Vzory malé h - 48 výřezů.
- Vzory malé i - 70 výřezů.
- Vzory malé j - 0 výřezů - na mapovém listu skoro žádné nebyly.
- Vzory malé k - 32 výřezů.
- Vzory malé l - 38 výřezů.
- Vzory malé m - 10 výřezů.
- Vzory malé n - 70 výřezů.
- Vzory malé o - 90 výřezů.
- Vzory malé p - 7 výřezů.
- Vzory malé q - 0 výřezů - na mapovém listu taktéž skoro žádné nebyly.
- Vzory malé r - 70 výřezů.
- Vzory malé s - 58 výřezů.
- Vzory malé t - 57 výřezů.
- Vzory malé u - 32 výřezů.
- Vzory malé v - 2 výřezy.
- Vzory malé w - 37 výřezů.
- Vzory malé x - 0 výřezů - nevyskytuje se.
- Vzory malé y - 20 výřezů.
- Vzory malé z - 26 výřezů.
- Vzory velkých písmen - dohromady 210 výřezů.
- Negativní vzory - náhodně vybraných 202 výřezů.

Dohromady to je 1175 pozitivních vzorů. Všechna písmena jsem převedl na velikost 24x24 pixelů. V průběhu implementace se ukázalo výhodnější natrénovat detektor s pomocí pouze některých skupinek písmen. Vytvořil jsem 5 rozhodovacích stromů pro následujících 5 skupinek písmen. Písmena jsou do skupin rozdělena podle mnou odhadnuté vzájemné podobnosti.

1. Trénovací pozitivní sada s písmeny a, e, o. Všechna písmena jsou kulatá s prázdným prostředkem - nejúspěšnější rozhodovací strom. Najde nejvíce písmen na mapě. Také proto, že se jedná o časté samohlásky.

2. Trénovací pozitivní sada s písmeny m, n, u, v, w. Všechna písmena mají jakési skoro svislé střídající se čáry.

3. Trénovací pozitivní sada s písmeny h, k, b. Všechna písmena mají jednu svislou čáru vlevo a tmavé čáry v oblasti vpravo dole od svislé čáry.

4. Trénovací pozitivní sada s písmeny t, i, l. Všechna písmena mají jednu svislou tmavou úzkou oblast uprostřed.

5. Trénovací pozitivní sada s písmenem s. Písmenu s se skoro žádné písmeno nepodobá

Ostatní písmena do těchto trénovacích sad nezvažuji. Jejich zařazení kupodivu nezlepšilo výsledky. Velká písmena taktéž nepracovala správně a většinou je obsáhl některý z pěti detektorů.

Trénovací negativní sadu jsem vytvořil ze všech negativních obrázků.

Trénování rozhodovacího stromu

Pro trénování rozhodovacího stromu má OpenCV funkce, fungující na principu popsaném v teoretické části. První část probíhá jako přípravná. Vezmou se jednotlivé pozitivní sady a spustí se program „createsamples“ dodávaný spolu s knihovnou. Ten má za úkol simulovat různé možnosti osvětlení objektu a má hlavní trénovací program upozornit na hledaný objekt. Jde zde nastavit určitý práh použitelný v případě alespoň minimální odlišnosti pozadí od hledaného písmene. Tento práh částečně odfiltruje pozadí písmene čímž sníží potřebný počet různých variant pozadí. Jde nastavit odchylka jasu hledaného objektu. Po testování jsem pro odchylku (maxidev) zvolil číslo 5 na možné stupnici jasu od 0 do 255. Střed jasu pozadí (bgcolor) jsem po testech nastavil na 200 s poměrně velkým záběrem (bgtresh) 150. To znamená, že se odfiltruje všechno pozadí s jasem větším než 50.

Samotné trénování umožňuje program „haarcascade“. Přednastaveně pracuje tento program se symetrickými hledanými objekty. Takže testuje pouze jednu stranu objektu z pozitivní sady, protože předpokládá, že ta druhá strana je symetrická. Toto je použitelné při detekci obličejů, pro písmo jsem raději zvolil použití nesymetrických objektů (parametr -nonsym). Dále se zadává počet stavů (-stages) tvořeného stromu (vždy jsem volil co největší, aby hodnota FAR celého stromu byla co nejmenší) a taktéž maximální možnou FAR každého stavu. Celková požadovaná FAR lze spočítat vzorcem v teoretické části z počtu stavů a požadované FAR na každém stavu. Pro každou skupinu písmen jsem volil jiný počet stavů v závislosti na tom, kolik skupina obsahovala písmen. Zpravidla od 10 do 25 stavů. Ostatní parametry těchto dvou programů pro tvoření stromu neuvádím, jelikož se netýkají optimalizace vytváření stromu a jsou jasně určené. (např. velikost vzorových obrázků pozitivní sady). Více o trénování se lze dozvědět v článku [5], podle kterého jsem postupoval.

Výsledný rozhodovací strom je uložen ve struktuře xml (struktura uložení dat v souboru).

6.2.2 Detekce

Obrázek je načten do struktury „IplImage“ čímž se v práci nezabývám. Objekt třídy, kterou jsem vytvořil, se vytvoří konstruktorem „textDetect(obraz)“ kde parametr „img“ je ukazatel na strukturu „IplImage“ kde je uložen obrázek s mapou. OpenCV má funkce na načtení

obrázku (cvLoadImage) z různých formátů. Pro testování jsem použil jen obrázy typu BMP, ale dají se načíst i obrazy JPEG a možná i jiné.

Nejdřív jsem vytvořil metodu na hledání podle jednoho stromu s pomocí funkce „cvHaarDetectObject“ (pro přesné použití funkce je lepší otevřít programovou dokumentaci OpenCV dodávanou v balíku spolu s knihovnou), která mnou vytvořená funkce obsahuje. Parametry této funkce OpenCV jsou:

- Obraz ve stupních šedi tzn. do stupňů jasu, převod pomocí funkce „cvCvtColor“.
- Parametry „scale“ a „resize“ ovlivňující velikost obrazu, kde jsou hledány nápisy. Experimentálně jsem zjistil, že je pro detekci lepší mapu zhruba 1,4 krát zvětšit. Detekce je sice pomalejší, ale úspěšnější.
- Údaj „scale_factor“ je další podstatný parametr ovlivňující kvalitu detekce. Experimentálně jsem ho nastavil na 1,05, což je poměrně málo a dosti to prodlužuje detekci, ale též dojde ke zlepšení výsledků takové detekce.
- Číslo „min_neighbors“ označuje vícenásobnou detekci popsanou v teoretické části. Pro zvýšení DR jsem nastavil jedničku.
- Velikost „min_size“ je nejmenší možný detekovaný objekt. Minimální velikost jsem nastavil na 3x3 pixely.
- Parametr „cascade“ je parametr ukazující na místo s rozhodovacím stromem. Ten je načten pomocí funkce Opencv („cvLoad“) ze souboru xml. Načtení provádím až ve funkci, která provede všech pět detekcí určených skupin písmen.
- Další parametry se již nijak speciálně nenastavují, je zde odkaz na místo, kam se má ukládat struktura s výsledky.

Metoda detekce podle jednoho stromu vrací seznam nálezů ve struktuře OpenCV („cvSeq“). Je typu „private“ a slouží jako pomocná metoda „findText“, která vrací seznam všech nálezů textu. „findText“ načte všech 5 rozhodovacích stromů ze složky „samples“ a s pomocí funkce pro hledání podle jednoho stromu opakované pětkrát najde všechny výskyty textu. Tyto výsledky vrací opět v seznamu.

Další funkce jsou pro filtraci výsledků. Používám 2 metody. První odfiltruje všechny výsledky ze seznamu podle velikosti. Experimentálně jsem práh nastavil na 50 pixelů, všechny větší nalezené objekty se ze seznamu výsledků odstraní. Další filtrace je poněkud inteligentnější. Seskládává písmena k sobě tak jak jsou zhruba poskládány v nápise. Využívá rekurzivní metody „setridit“. Její algoritmus je následující:

```
setridit  (p,       //index v seznamu nalezy písmene, ke kterému se
                      vyhledávají sousední písmena
           nalezy,  //seznam výsledků detekce
           *indexy, //seznam s indexy jež už byly použity - ukazatel
           hranice) //hranice v nalezy pokud se budou vyhledávat
{                     sousední písmena
  i = 0
  vlož (p, indexy)  // vloží index p do seznamu indexů indexy
  while ( i < hranice ){      //hledá se až do určené pozice v seznamu
    if ( neníV( i, indexy ) ) //pokud index i
```

```
                                není v seznamu použitých indexů
      if ( vBlízkosti ( nalezy[p], nalezy[i], vzdálenost ) ){
                    //pokud jsou objekty v blízkosti
        setridit ( i, nalezy, *indexy, hranice, hranice )
      }
      i = i + 1                //hledej na další pozici v seznamu
  }
}
```

Tato metoda se začne aplikovat na začátku seznamu postupně na každý prvek, jako hranice se zvolí celá délka seznamu. Po prvním projití celé metody „setridit“ se v seznamu „indexy“ objeví např. 6 indexů ukazující pozice nálezů v seznamu „nálezy“. Objekty na těchto indexech se přesunou na konec seznamu „nalezy“ a „hranice“ se zmenší o počet přesunutých objektů v našem případě o 6. V dalším kroku se opět vezme nález s prvním indexem, protože ten minulý byl přesunut nakonec, a postupuje se obdobně.

Testování, zda jsou dva nalezené objekty v blízkosti, probíhá podle návrhu, tedy prohledávají se oblasti ve tvaru rovnoramenných trojůhelníků napravo a nalevo od objektu. Trojúhelníky jsem popsal pomocí přímek $x - 2y = 0$ a $x + 2y = 0$, což platí v případě, že objekt, vzhledem ke kterému ostatní objekty hledáme, leží na souřadnici $[0, 0]$ Kartézské souřadné soustavy. V případě potřeby tyto přímky posuneme jinam. Takto jsem určil vlastně dvě ramena ohraničující oblast hledání. Poslední hranice je v určité vzdálenosti ve smyslu osy x. Tuto vzdálenost jsem experimentálně určil jako součet velikostí výchozího objektu a objektu nalezeného ležícího v blízkosti.

Poslední věcí, kterou bych rád zmínil v oblasti filtrace, je odstranění nálezů, které nejsou pravděpodobně nápisy. Pokud se po každém přesunutí setříděných nálezů na konec seznamu zaměříme na seznam „indexy“, uvidíme, kolik objektů tvoří nápis. Pokud je jenom jeden, rovnou ho odstraníme, protože žádný nápis není nalezen jen jedním objektem v sekvenci nálezů (někdy se písmeno na mapě samostatně vyskytuje, ale ze 75% jde o chybný nález). Pokud je nálezů v rámci jednoho nápisu tedy setříděného shluku víc jak sedm tak dále tyto nálezy netestuji a přesouvám nakonec. Pokud je seznam blízkých nálezů dlouhý mezi dvěma a sedmi testuji vzdálenost mezi nejlevějším a nejpravějším nalezeným objektem a pokud je větší než 2.3 krát průměrná velikost seskupených nálezů, tak je příjmu a zařadím nakonec, jinak je odstraním. Konstanta byla určena experimentálně.

Implementoval jsem také ještě funkci „optimalPosition“, která je předchůdcem již zmíněné metody. Hledá pouze samostatně ležící nálezy, které odfiltruje. Navíc se zde dá zadat parametr jak daleko se od sebe musí nálezy vyskytovat, aby se o nich dalo říct, že k sobě nepatří.

Tímto bych ukončil sekci o implmentaci vyhledávání textu. Dostal jsem přefiltrovaný seznam všech nálezů textu. V další části se věnuji odstranění písma z nalezených oblastí s písmem.

6.3 Odstranění nápisů

Tato část se dělí na dvě podčásti. První se týká samotného odstranění samotného nápisu pomocí prahu a druhá opět jisté optimalizaci celého procesu.

6.3.1 Prahování

Funkci prahu jsem už popsal. V implementované metodě slučuji prahování a optimalizaci dohromady s celou opravou nápisu. V rámci parametru této funkce lze nastavit ručně hodnotu prahu. Používal jsem práh 180 na stupnici 0–255. Prahování jsem implmentoval tak, že procházím pouze oblasti určené jako nálezy detektoru. A všechny pozice pixelů mající jas menší než daný práh zanáším do masky shodné velikosti s obrazem mapy. Tato maska má všechny hodnoty nastaveny předem na nulu, pouze pozice korespondující s pozicemi prahovaných pixelů mají hodnotu 255.

6.3.2 Optimalizace výsledků

Jak už jsem se zmínil v návrhu, je potřeba výsledek navíc dotvořit. Jelikož se spolu s nápisy prahováním odstranily (zanesly se do masky) také na některém místě četné šrafy, které tvořily pouze tenké linie v masce, implementoval jsem svou metodu podobnou erozi, kterou na masce provádím. Zde se ukazuje výhoda použití masky před přímým zásahem do obrázku mapy, lze totiž ještě masku upravit. V každé oblasti nálezu spočítám poměr pixelů které neprošly prahováním a byly v masce označeny a celkovým počtem pixelů oblasti. Pokud je poměr vyšší než 0,7 (opět experimentálně zjištěno), procházím oblast v rámci masky pixel po pixelu a u každého pixelu s hodnotou 255 a pokud nemá ve svém devíti-okolí minimálně 6 pixelů s hodnotou 255 (experimentálně určeno) nastavím jeho hodnotu na z 255 na 1. Nakonec všechny pixely masky s hodnotou 1 najdu a nastavím je zpět na 0.

Poté na celou masku aplikuji dilataci, čímž se plochy s hodnotou 255 zvětší o jednu vrstvu pixelů. Využívám funkci OpenCV („cvDilate“) s čtvercovou množinou bodů 3x3. (experimentálně jsem zjistil, že dosahuje nejlepších výsledků).

Tím mám celou masku hotovou a zbývá pouze podle této masky zaplnit místa označená na masce hodnotou 255.

6.4 Zaplnění místa po nápisech

Poslední část implmentace se skládá ze dvou fází, první je přípravná a druhá je ta hlavní, zaplňovací. Implementaci popíši odzadu protože takto probíhala. Nejdřív jsem místo, kde byl text (pořád tam je, ale na masce jsou tyto pixely označeny), zaplnil pomocí masky a algoritmu A. Telea popsané v návrhu a implementované v OpenCV („cvInpaint“). Poté jsem podle výsledků zjistil, že dochází k chybám kvůli velkým jasovým přechodům v místech, kde bylo hodně šraf. Po dlouhém zkoušení všech možných řešení jsem objevil poměrně jednoduchou optimalizaci. Vytvořil jsem kopii obrázku mapy a filtroval jsem ji mediánovým filtrem, tím se odstranily velké jasové přechody v místech šraf. Poté jsem na této kopii spustil již zmíněné zamalování podle masky a nakonec jsem všechny zamalované pixeli překopíroval opět podle masky do původního obrázku mapy. I když i po této optimalizaci dochází k chybám, jsou mnohem méně časté než byly předtím.

Další kapitola bude obsahovat testy a obrazovou dokumentaci k celému postupu.

Kapitola 7

Mezivýsledky, testy, možnosti rozšíření

V této kapitole je zhodnocení dosažených výsledků. Nejdříve se zaměřuje na všechny postupné kroky v implementaci, tedy nejdřív na detekci textu, na jeho odstranění a nakonec zaplnění odstraněného místa. V první části zhodnotím všechny postupné mezivýsledky, ve druhé provedu několik základních testů na různých mapách spolu se zhodnocením těchto testů a v poslední části popíši možnosti rozšíření a zlepšení celé práce s texty.

7.1 Postupné výsledky

V této části se postupně zaměřím na všechny mezivýsledky v průběhu zpracování textu. Pro přehlednost opět budou následovat 3 části. První se bude věnovat detekci, druhá odstranění nápisů a třetí zaplnění místa po odstraněném textu.

7.1.1 Detekce textu

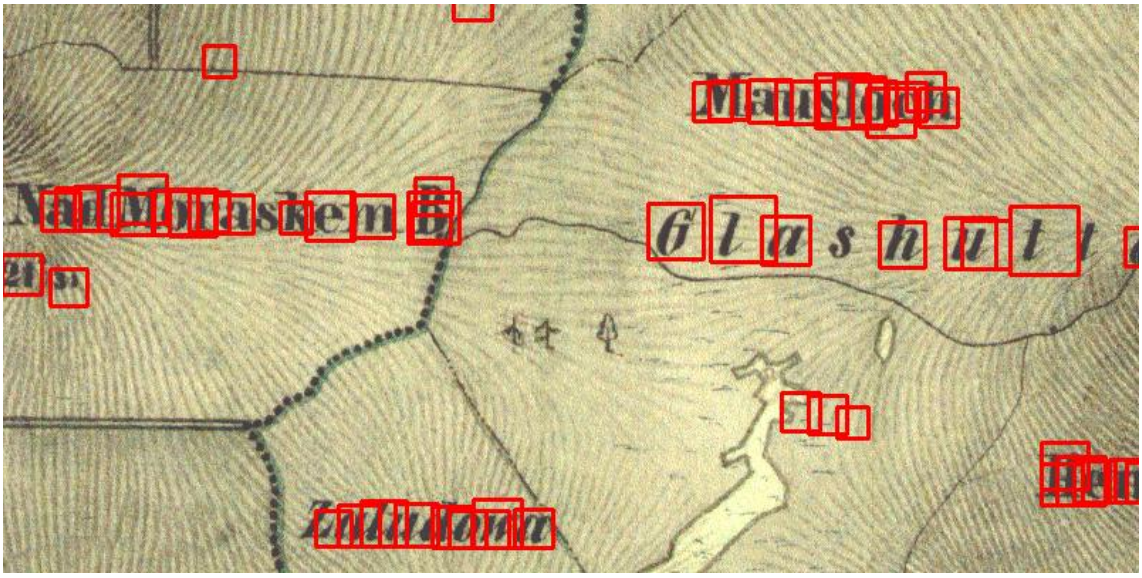
Část věnující se detekci pomocí pěti po sobě jdoucích detektorů a následně optimalizaci detekce, kde se odfiltrují nepravděpodobné detekce.

Zobrazuji výsledky na stejném úseku mapy.

Hledání písmen a, o, e

Detekce těchto písmen na obrázku 7.1 ukazuje, že se nenajdou pouze písmena, na kterých byl detektor natrénován, ale i většina ostatních. Jedná se o detektor, který zpravidla vykazuje nejvíce nálezů a není to pouze tím, že hledá časté samohlásky. Důvod bych spíš viděl v počtu pozitivních vzorů, se kterými byl tento detektor natrénován. Proto jsem také netrénoval detektor např s písmenem „c“. Taktéž je zajímavé, že při této detekci je nalezeno velké „M“ v pravé horní části obrazu. Dále je dobré si všimnout písmene „s“ v pravé části obrazu 7.1, které bude nalezeno až detektorem na písmeno „s“. Ve skutečnosti by se dalo říct, že pokud tento detektor hledá a označuje kromě písmen „a“, „o“ a „e“ v nezanedbatelné míře i jiná písmena, jedná se o typické vysoké FAR. To je pravda, ale je to naprosto záměrně, pokud by měl tento detektor rozeznávat jen svá písmena, bylo by nutné přidat do negativních vzorů všechna ostatní písmena. To jsem neudělal a tudíž dochází k velké detekci i jiných písmen.

Posledení věc, na kterou bych rád upozornil je falešná detekce v pravé spodní části obrazu (jedná se o místo se třemi červeně označenými oblastmi), tuto chybnou detekci odstraním filtrací po detekci.



Obrázek 7.1: Hledání písmen a, o, e.

Hledání písmen m, n, u, v, w

Tento detektor nepřináší na obrázku 7.2 moc nových nálezů, přesto je užitečný. Mnohem lépe nachází a ohraničuje velké písmeno „M“ v pravé horní části obrazu. Opět nebyl tento detektor na velké „M“ trénován, ale toto písmeno má podobné vlastnosti jako malé „m“ především jakési tři kolmé tmavé oblasti.

Falešné detekce opět lze odstranit ve filtraci.

Hledání písmen h, k, b

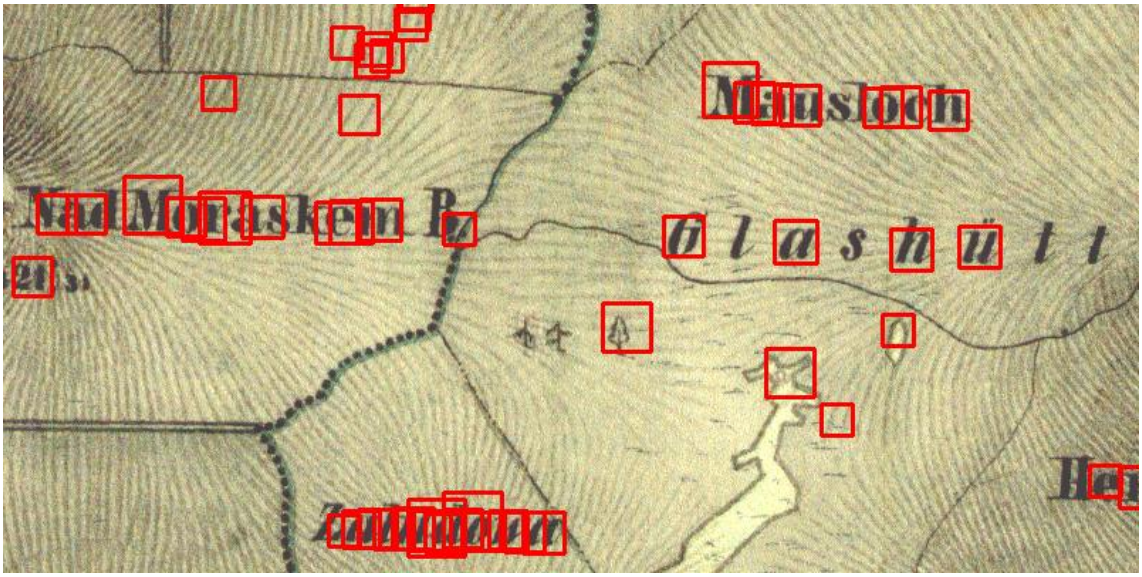
Výsledek je na obrázku 7.3. Nálezů je méně než v případě předešlých detekcí. I počet písmen v trénovací sadě je menší než byl v předchozích případech. Přesto bych zde vyzdvihl dva pozitivní dopady tohoto detektoru. První je perfektní detekce písmene „h“ v pravé části obrazu, kde předchozí detekce toto písmeno zvrchu osekly, a druhá výhoda je malá míra FAR, která může být i důsledkem malé detekce.

Hledání písmen t, i, l

Obrázek s výsledkem je 7.4. Na testovacím obrázku se plně neprojevil přínos této detekce. Poněkud záhadně nedetekoval dvě písmena „t“ v pravé části. Písmena „l“ detekuje dobře.

Hledání písmen s

Poslední detekcí je samostatná detekce písmene „s“ na obrazu 7.5. Tato detekce byla pouze jemné doplnění předchozích výsledků. Jejím pozitivem je nalezení písmene „s“ v pravé části



Obrázek 7.2: Hledání písmen m, n, u, v, w.

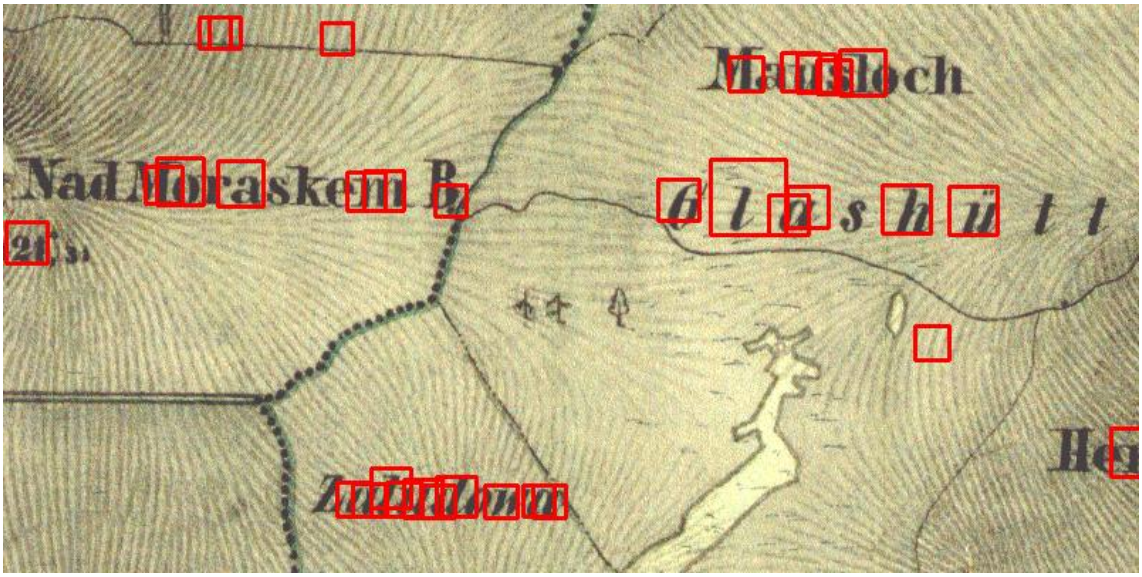
Obrázek 7.3: Hledání písmen h, k, b.

obrazu 7.5, které neoznačil žádný z předchozích detektorů. Toto písmeno mělo malý počet vzorů v trénovací sadě.

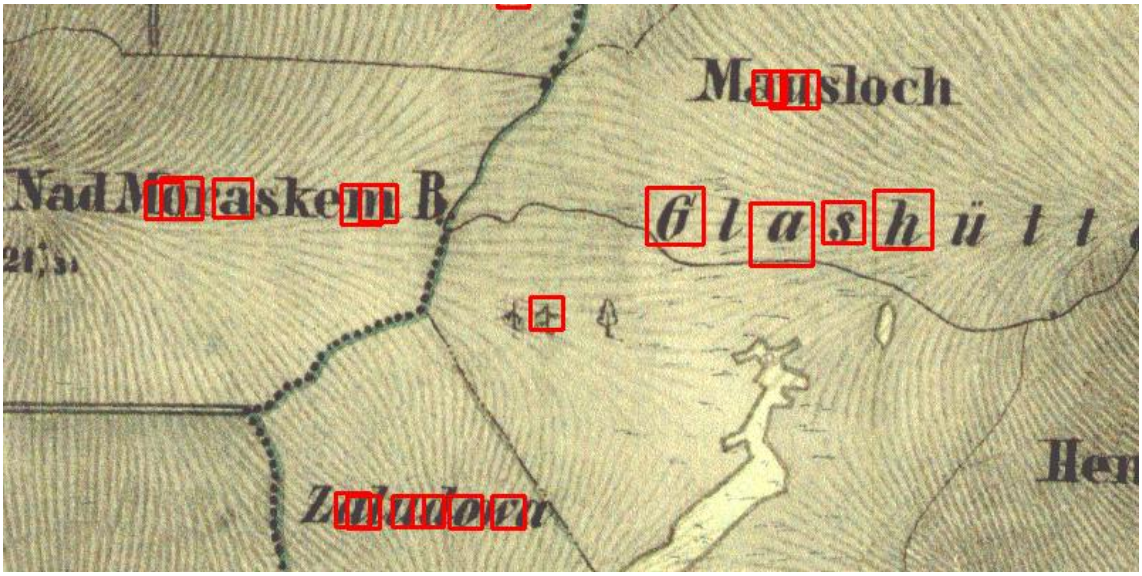
Zhodnocení předchozí detekce

Předchozí detekce ukazuje, že čím je trénovací sada větší, tím je zpravidla větší i počet nálezů. Nejspíše to souvisí s větší adaptabilitou takového detektoru, neboť s menším počtem detektor spíše detekuje pouze přesné vzory, na které byl natrénován.

Taktéž větší počet detekcí znamená zvýšenou velikost nechtěné FAR. Nechtěné uvádím



Obrázek 7.4: Hledání písmen t, i, l.



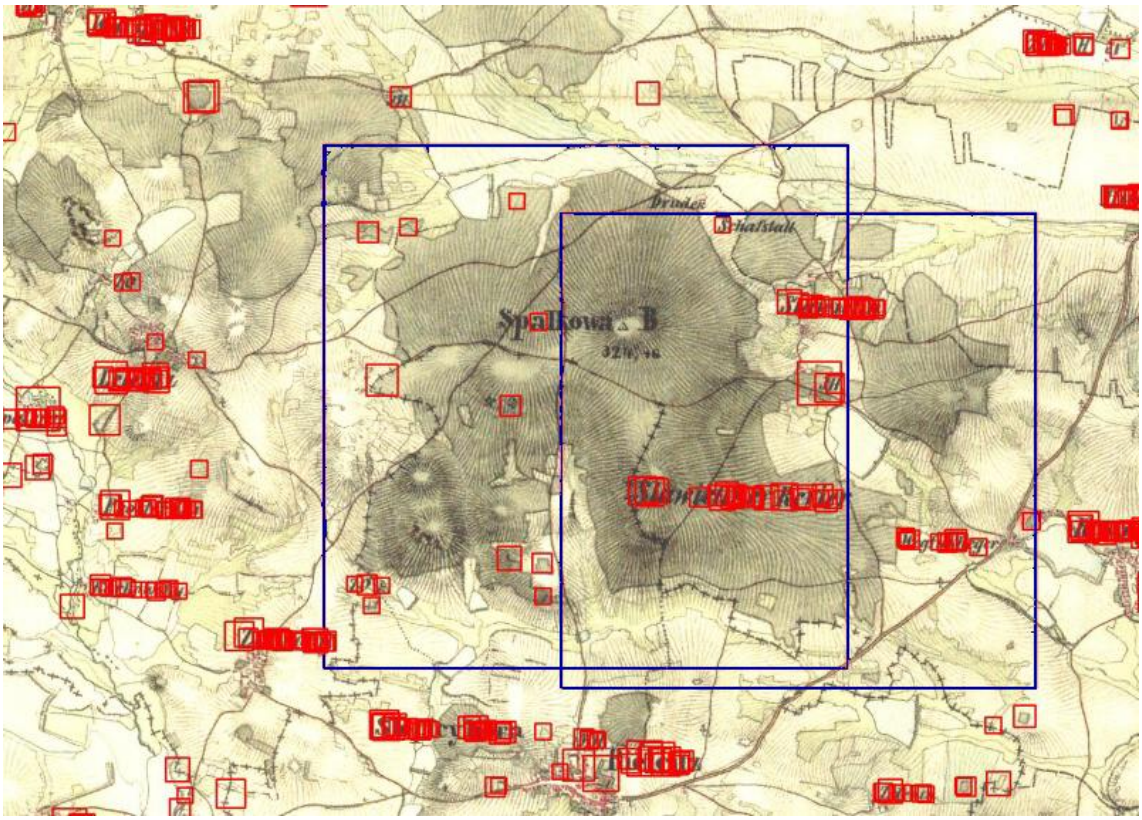
Obrázek 7.5: Hledání písmen s.

proto, že pokud se jedná o FAR, kdy se naleznou jiná písmena než ta, na která byl detektor trénován, jedná se o chtěné zvýšení FAR.

Každý detektor nalezne určitou množinu písmen, průnik množin všech detektorů, může být pořád poměrně velká množina detekcí. Tím chci říct, že každý další detektor po prvním detektoru hledající „a“, „o“ a „e“ pouze přidá několik dalších specifičtějších detekcí. Prvotní množina detekcí se tím ale už vícenásobně nezvětší.

Filtrace detekcí pomocí velikosti detekovaného objektu.

Výsledek po této filtraci je naznačen na obrázku 7.6. Detekce označené modrým čtvercem budou odstraněny, neboť text se v takovéto velikosti nevyskytuje.



Obrázek 7.6: Znázornění příliš velkých detekcí modrou barvou

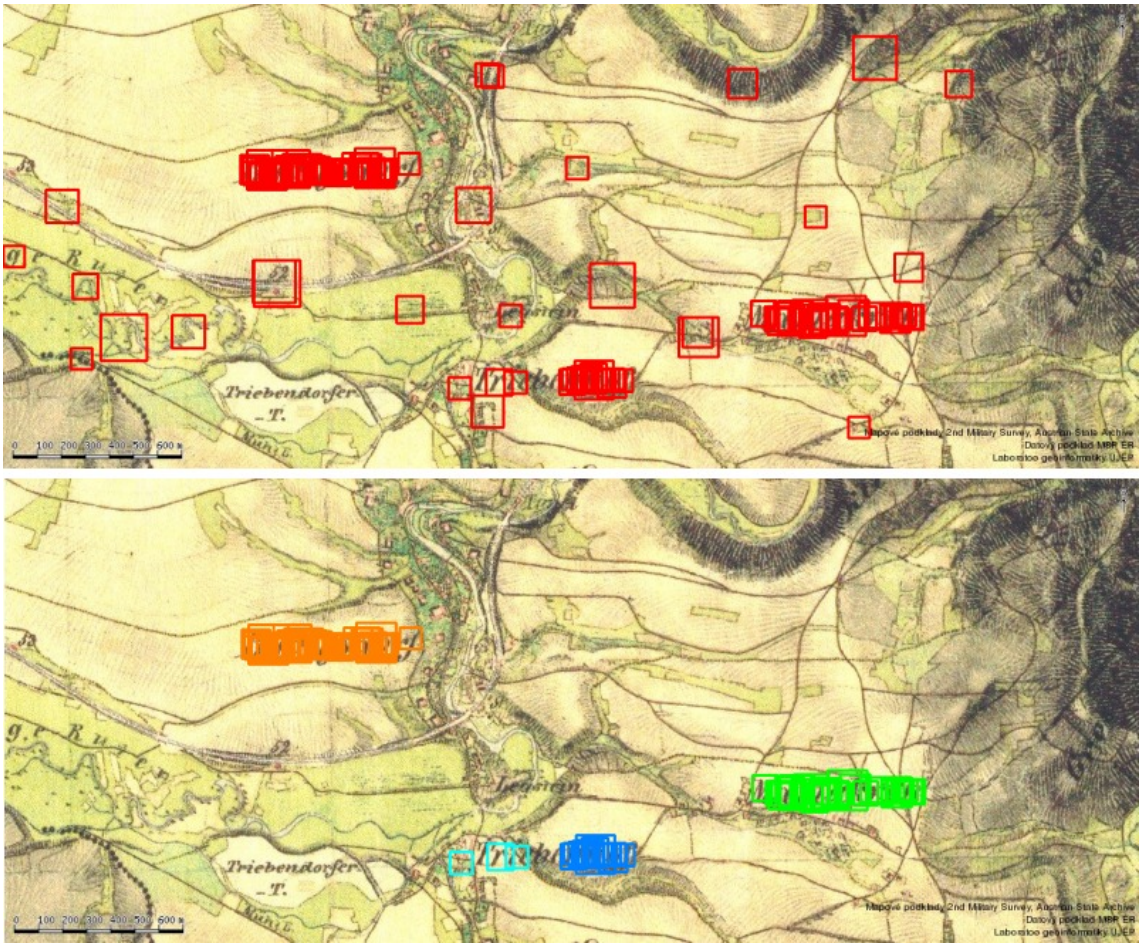
Filtrace detekcí seskládáním písmen

Tuto filtraci jsem popsal v návrhu a v implementaci jsem popsal jak pracuje. Její výsledky jsou perfektní. Lze to posoudit na obrázku 7.7. Písmena označená jednou barvou vždy tvoří nápis. Pro porovnání je na obrázku také výstup, kde byla provedena pouze filtrace detekcí pomocí velikosti. Jediný chybný výstup jež po filtraci na testovacím obrázku zůstal je světle modře označený nález, který bohužel leží v linii s nápisem. Jedná se o světle modrý čtverec nejvíce vlevo.

7.1.2 Odstranění textu z detekovaných oblastí

V této části se zaměřím na výsledky po prahování. Detekce již nebudu označovat barevnými čtverečky, ale pracuji vždy pouze v oblastech pixelů, které byli detekovány jako oblasti s textem. Při prahování zatím nezasahuji do obrázku mapy, ale vytvářím masku kde zaznačuji místa prahovaných pixelů.

Výsledek po pouhém prahování jsem již uvedl v návrhu. Nyní se pokusím prezentovat, proč je nutné prahovací masku dilatovat a rozšířit tak odstraňované oblasti o jeden obal



Obrázek 7.7: Rozdíl po filtraci pomocí seskupení písmen názvů, v horní části jsou detekce před touto filtrací.

pixelů. Další obrázek 7.8 zobrazuje rozdíl po zaplnění textu mezi obrázky, kde byla provedena dilatace masky a kde nebyla. Na levé straně je vidět jak tmavé okraje znehodnotily zaplnění místa po textu.

Pro vysvětlení masky uvádím další obrázek 7.9. Postupně ukazuje zpracování obrázku. Nejdřív se z originálního obrázku označeného A vytvoří maska B, a to pouze prahovaním. Poté je na masce v místech, kde jsou poměrně husté šrafy, provedena eroze. Lze ji vidět u pole C především na levém kraji nápisu. Na poli D je provedena dilatace pole B a nakonec pole E je po provedení dilatace na poli C, tedy dilatace po erozi.

Zhodnocení prahování a jeho úprav

Na maskách je vidět, že při provedení eroze se dá zachovat více užitečných segmentů původního povrchu. Důležitost tohoto kroku se ukáže hlavně v oblastech s velkým počtem šraf. Taktéž je vidět, že při prahu 180 se všechno písmo odstraní.

Obrázek 7.8: Rozdíl mezi zaplněnými místy po nápisech. Vpravo byla prahovací maska dilatována.

7.1.3 Zaplnění místa po nápisu

Poslední krok úprav textu se týká zaplnění míst po nápisech. Schválně zde uvedu příklad z místa, které je problemtické kvůli velkému počtu šraf. Na obrázku 7.10 na části označené písmenem B je vidět výsledek zaplnění. V návrhu popisuji optimalizaci, při které se má odstranit chyba jasu. Ta je vidět v části A obrázku 7.10 zhruba uprostřed obrázku. Tuto chybu lze zčásti omezit, pokud zaplnění provádím na obraze filtrované mediánovým filtrem. Na originální obraz mapy pak podle masky přenesu pouze zaplněné pixely.

Tímto jsem prošel všechny kroky celého postupu odstranění nápisů na mapě. V další části uvádím několik testů celého postupu.

7.2 Testy

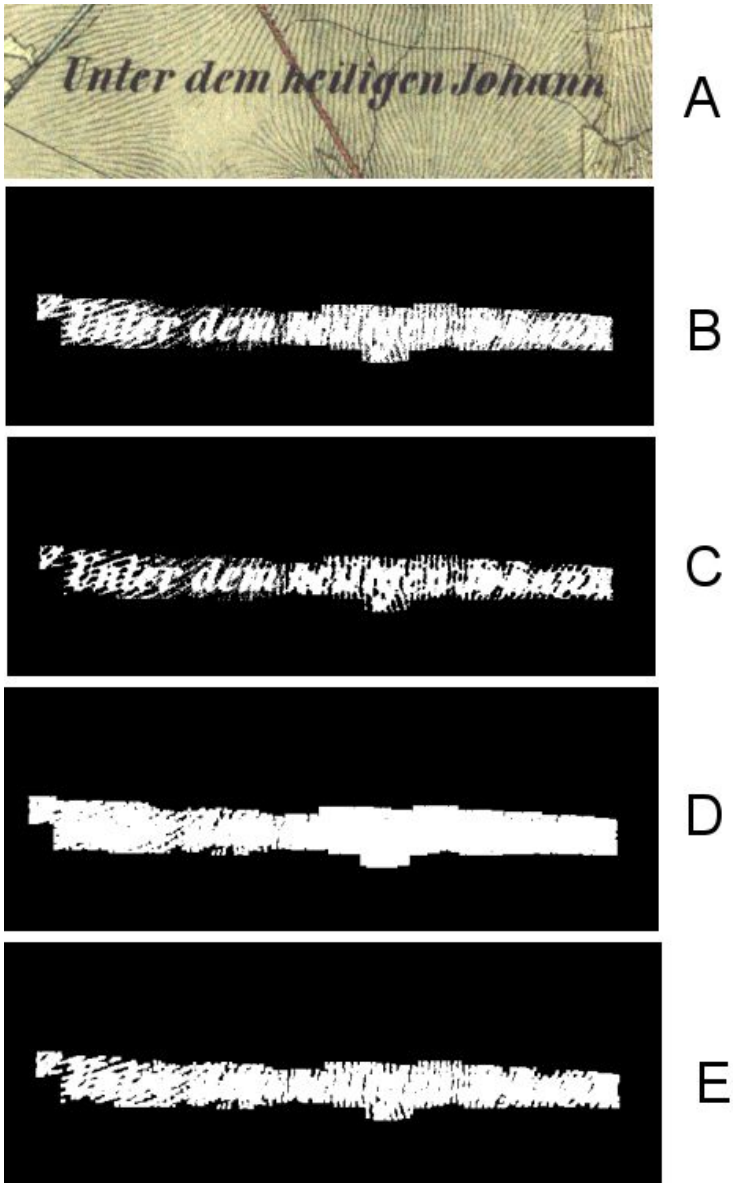
V této části provedu několik testů na zajímavých místech map. Pokusím se výsledky objektivně zhodnotit na konci této části. Testy větších mapových listů jsou na přiloženém CD.

7.2.1 Test množství zpracovaných nápisů

Na obrázku 7.11 se snažím zachytit jaká je asi úspěšnost detekce textu. Odhadem bych ji určil na 75%, ale je to jen ztěží měřitelné. U míst na mapě, kde jsou nápisy na světlém pozadí bez rušivých elementů, je detekce bezchybná. U velmi tmavých míst, kde se nápisy ztrácejí, je detekce horší. Na druhou stranu lze polemizovat s potřebou odstranit tmavé nápisy z tmavých ploch. Kousek mapy, který testuji, je běžně vypadající mapa, daly by se najít lepší, ale i horší. Zachytit šikmý nápis na mapě už vůbec nezvažuji.

7.2.2 Test jiného typu písma, na světlém pozadí

Obrázek 7.12 zobrazuje test zpracování nápisů napsaných jiným typem písma než předchozí test. Je vidět, že na světlém pozadí bez rušivých tmavých elementů, pracuje detekce mno-



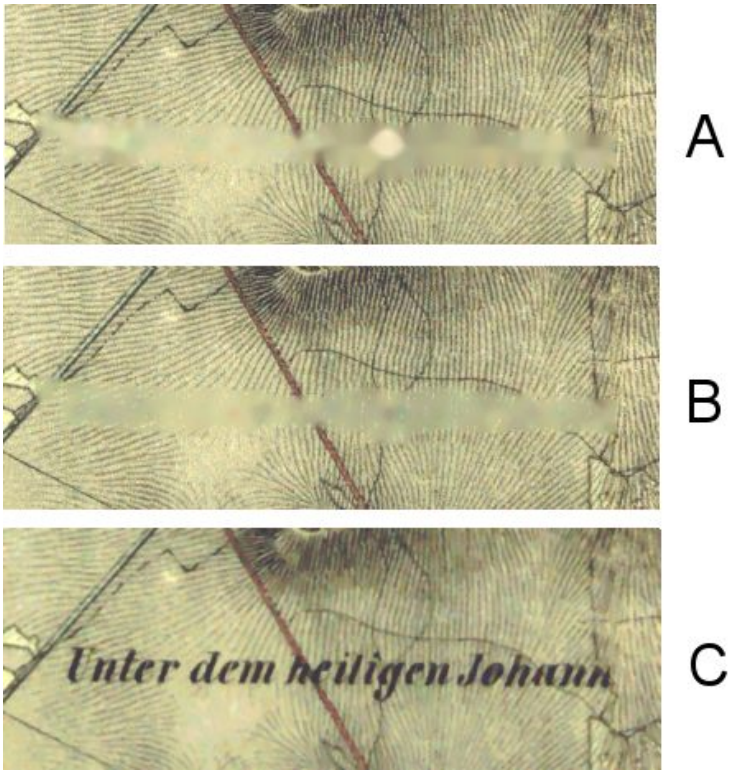
Obrázek 7.9: Nejdřív se z originálního obrázku označeného A vytvoří maska B, pouze prahovaním. Poté je na masce v místech, kde jsou poměrně husté šrafy, provedena eroze. Lze ji vidět u pole C především na levém kraji nápisu. Na poli D je provedena dilatace pole B a nakonec pole E je po provedení dilatace na poli C, tedy dilatace po erozi.

hem lépe. K tomuto testu bych ještě dodal, že nápisy jsou poměrně malé. Nápisy psané větším písmem jsou také mnohem lépe detekovatelné.

7.2.3 Závěr testování - zhodnocení výsledků

Z testů lze zjistit, že detekce nápisu není dokonalá, někde pracuje lépe a někde hůře. Taktéž zamalování nápisů na světlých pozadích vypadá lépe, než v místech se šrafami.

Celková doba zpracování nápisů na mapě o velikosti 3840x3840 trvá okolo dvou mi-

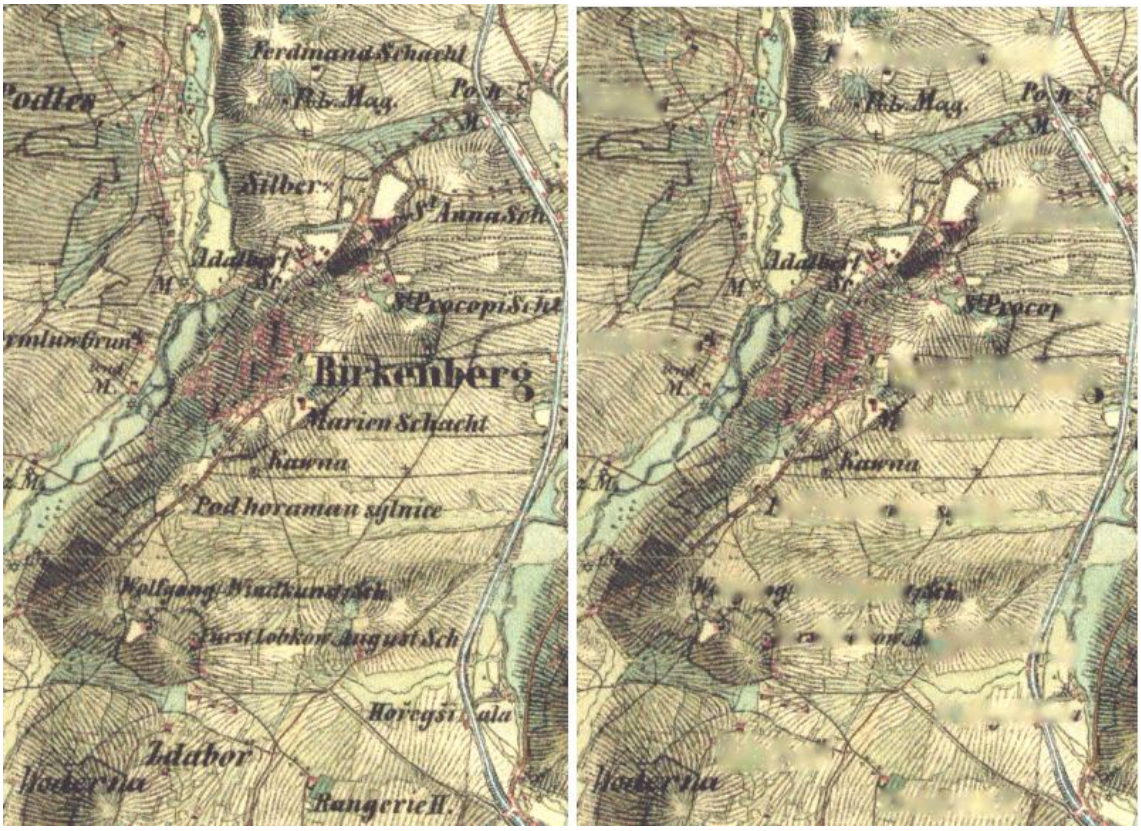


Obrázek 7.10: Část A popisuje chybu jasu - malá světlá oblast uprostřed obrazu. Část B je výsledek při použití mediánového filtru a C je obraz po mediánové filtraci, ze kterého také pomocí masky zaplňuji oblasti textu označené v masce

nut, na notebooku Lenovo 3000 N100 s procesorem Core 2 Duo T5200 a s 1,5 GB RAM. Knihovna pracuje s mapami, jež odpovídají naskenovaným originálům a maximálnímu přiblížení map na www.mapy.cz - při výběru starých map ze Druhého vojenského mapování. Nápisy zde mají určité běžné velikosti, při použití zmenšených map nemusí být detekce úspěšná. Stejně tak při velkém zvětšení map je ohrožena správná detekce.

Nejtěžším problémem je nejprve text na mapě identifikovat. Odstranění a zaplnění místa po nápisech už je úkol druhotný. V obojím je co zlepšovat, některé nápady uvádím v další části. Stojí za to zamyslet se, jestli detektor od Paula Violy a Michaela Jonese je vhodný výběr. Je zvyklý pracovat s mnohem většími trénovacími sadami, než jsem použil. Je zaměřen především na rychlost, což není úplně to co potřebuji. Spíš zvažuji jestli nebylo vhodné použít metody typu „template matching“ což je porovnávání vzorů. Sice je to pomalejší metoda, ale mohla by být přesnější. S odstraňováním nápisů toho moc vymyslet nejde, možná snad nějaká inteligentní eroze, či adaptivní práh nastavovaný podle okolí písmene.

Sběr vzorů do trénovací sady byl poměrně časově náročný. Samotné trénování by problematické nebylo, kdybych věděl, jak vše optimálně nastavit. Takhle jsem metodou pokus - omyl zkoušel mnohá a časově náročná řešení trénovací sady naprosto zbytečně.



Obrázek 7.11: Test ukazující míru úspěšnosti detekce textu

Obrázek 7.12: Test jiného písma na světlém pozadí

7.3 Možnosti rozšíření

V této části je popis některých možností zlepšení metod této knihovny.

- Použití větších trénovacích sad. Pro detekci obličejů byly použity sady s více jak 5000 obličeji, má největší sada má okolo 300 vzorů. Mohlo by to zkvalitnit detekci.
- Metody odstranění nápisu by bylo možné vylepšit tak, aby zůstali např. zachovány cesty v těsné blízkosti písmen.

- Metoda zamalování je volena trochu nešťastně. Lepší by byla metoda, která by nepropagovala jas, ale ve směru hran v okolí odstraněné oblasti by kopírovala sousední pixely. Tímto by se možná zachoval směr šraf v zaplněné oblasti.
- Šlo by dále pokračovat v systému digitalizace map a začít pracovat na zpracování oblastí map např. lesů.
- Metody implementované v knihovně by měly umožňovat širší paletu parametrů, aby se více daly měnit účinky metod.

Kapitola 8

Závěr

Tato práce obsahuje teoretický úvod do zpracování starých map se zaměřením na mapy z Druhého vojenského mapování. Taktéž se věnuje obecnému postupu zpracování obrazu, popisuje metody hledání hran a metodu detekce objektů pomocí AdaBoostu.

V návrhu jsou popsány metody, které jsem navrhl použít pro zpracování nápisů na mapách. Některé tyto metody jsem nakonec použil k vytvoření knihovny,která obsahuje metody k detekci, odstranění nápisů a k zaplnění místa po nápisech. U všech metod, které používám, jsem popsal implementaci. Po přečtení této práce by mělo být jasné proč a jak tyto metody používám.

Vše jsem zdokumentoval v poslední kapitole věnující se postupným mezivýsledkům při zpracování nápisů. V této kapitole také uvádím dva testy pro zdokumentování funkčnosti této knihovny.

Zásadním cílem této knihovny bylo odstranění nápisů na mapě a nahrazení tohoto místa odhadovaným původním povrchem. Dělá se to kvůli zpřesnění pozdější digitalizace. Velký nápis zabírá poměrně velkou plochu, která by se započetla při automatické digitalizaci např. lesů jako místo, kde les není, čímž by tuto digitalizaci znepřesnil. Předem jsem si nedefinoval kolik procent textu musím objevit a smazat, s výsledkem jsem však spíše spokojen, než nikoli. Pokud bych měl zhodnotit v procentech úspěšnost jednotlivých kroků při odstraňování textu získla by detekce 75%, prahování 95% a zaplnění 70%. Úspěšnost však lze těžko změřit, a proto se jedná pouze o mé odhady.

Obsahem je také velmi důkladně seznámení s detekcí pomocí AdaBoost, kterou jsem se poměrně do hloubky zabýval. Toto použití určitě dokazuje její velkou reálnou použitelnost i v jiných oblastech použití.

Celý text je poměrně hojně dokumentován obrázky pro lepší pochopení celé problematiky. V přílohách také přikládám vzory písmen z trénovacích sad. Věřím, že pokud bych měl znovu podobný detektor natrénovat zabralo by to mnohem méně času, jelikož jsem prošel spoustou neúspěšných pokusů a slepého bádání.

Literatura

[1] Digitalizace a zpracování obsahu. [online], [cit. 28. 12. 2007].
URL home.zcu.cz/~holota5/publ/DigZpr0.pdf/

[2] Encyklopedie publikačních formátů: TIFF. [online], [cit. 28. 12. 2007].
URL http://www.grafika.cz/art/polygrafie/enctiff.html?tisk=on

[3] Historická mapovaní českých zemí. [online], [cit. 10. 5. 2008].
URL http://projekty.geolab.cz/gacr/a/files/miks_zim_GEOS06.pdf

[4] Houghova transformace. [online], [cit. 28. 12. 2007].
URL http://cmp.felk.cvut.cz/cmp/courses/DZO/Cviceni/cv3/hough.htm

[5] How-to build a cascade of boosted classifiers based on Haar-like features. [online], [cit. 15. 4. 2008].
URL http:
//robotik.inflomatik.info/other/opencv/OpenCV_ObjectDetection_HowTo.pdf

[6] Knihovna OpenCV. [online], [cit. 28. 12. 2007].
URL http://www.intel.com/technology/computing/opencv/overview.htm/

[7] Mapové podklady - historická mapa 1836-1852. [online], [cit. 20. 12. 2007].
URL http://napoveda.seznam.cz/cz/historicka.html

[8] Parcel Mapping Using GIS. [online], [cit. 28. 12. 2007].
URL http://www.umass.edu/tei/ogia/parcelguide/index.html

[9] Prezentace starých mapových děl z území Čech, Moravy a Slezska. [online], [cit. 20. 12. 2007].
URL http://oldmaps.geolab.cz/

[10] Rozpoznávanie obrazcov. [online], [cit. 28. 12. 2007].
URL http://www.sccg.sk/~ftacnik/pattrek.htm/

[11] Skripta předmětu základy počítačové grafiky – fit. [online], [cit. 15. 4. 2008].
URL https://www.fit.vutbr.cz/study/courses/IZG/private/lecture03/izg_transformace_2d3d.pdf

[12] Zeměpis místní krajiny - Nymbursko. [online], [cit. 1. 5. 2008].
URL http:
//www.gym-nymburk.cz/projekty/zemepis/fyzickogeograf/stare_mapy.htm

[13] Telea, A.: An Image Inpainting Technique Based on the Fast Marching Method. [online], [cit. 18. 5. 2008].
URL http://www.win.tue.nl/~alext/ALEX/PAPERS/JGT04/paper.pdf

[14] Viola, P.; Jones, M.: Robust Real-time Object Detection. [cit. 20. 12. 2008].

[15] ústav grafiky VUT FIT, U.: Materiál předmětu POV a ZPO. [cit. 15. 4. 2006].

[16] Šochman, J.: AdaBoost. [online], [cit. 4. 5. 2008].
URL http:
//cmp.felk.cvut.cz/cmp/courses/recognition/Labs/adaboost/adaboost.pdf

[17] Železný, M.: Zpracování digitalizovaného obrazu. [online], [cit. 20. 4. 2008].
URL http://147.228.47.19/courses/zdo/ZDO_aktual_060217.pdf

[18] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, iSBN 80-251-0454-0.

Kapitola 9

Přílohy

Seznam příloh:

1. Pozitivní vzory pro trénování
2. Negativní vzory pro trénování
3. Informační plakát