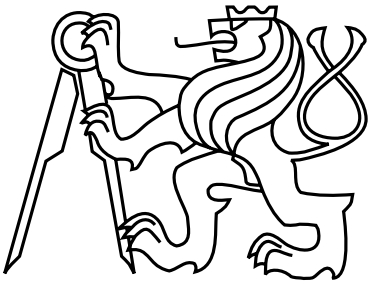
České vysoké učení technické v Praze
Fakulta stavební
Katedra mapování a kartografie



Bakalářská práce

Automatické vyhledávání vzorů v obraze

Zora Hořejšová

Vedoucí práce: Ing. Jiří Cajthaml, Ph.D.

Studijní program: Geodézie a kartografie, bakalářský

Studijní obor: Geoinformatika

13. května 2010

Poděkování

Děkuji především vedoucímu bakalářské práce Ing. Jiřímu Cajthamlovi Ph.D. za vedení a cenné připomínky a Ing. Janu Řezníčkovi za ochotu pomoci a předání nabytých zkušeností. Upřímné díky patří hlavně mé rodině za podporu během celého studia nejen na vysoké škole a také Ing. Martinu Fouskovi a jeho rodině za projevenou přízeň a podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval/a samostatně a že jsem uvedl/a veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

V Příbrami dne 13. května 2010 .....

Abstract

This thesis describes the analysis and the implementation of the application for automatic search of patterns within images, concretely in historical maps. The application facilitates the map tracing through the automatic search itself, as well as through the use of other tools, such as the correctness checking of searched patterns or the possibility of the resultant coordinates transformation at the export. In the thesis there is also an overview of chosen methods for two images similarity determination or description of application development based on NetBeans Platform.

Key words

image registration, patterns matching, cross-correlation coefficient, JAI, images in Java, NetBeans Platform, NetBeans modules, NetBeans layer files, NetBeans default lookup

Abstrakt

Tato práce popisuje analýzu a implementaci aplikace pro automatické vyhledávání vzorů v obraze, konkrétně v historických mapách. Aplikace usnadňuje vektorizaci map jak samotným automatickým vyhledáváním, tak použitím dalších nástrojů, jako je kontrolování správnosti vyhledaných vzorů či možnost transformace výsledných souřadnic při exportu. V práci se také vyskytuje přehled vybraných metod pro určení podobnosti dvou obrazů nebo popis vývoje aplikace založené na NetBeans platformě.

Klíčová slova

registrace obrazu, vyhledávání vzorů, vzájemný korelační koeficient, JAI, obrázky v Javě, NetBeans platforma, NetBeans moduly, layer soubory v NetBeans, defaultní lookup NetBeans

Obsah

Kapitola 1

Úvod

V dnešní době stále se zvyšujícího výpočetního výkonu je ve všech oblastech lidského života snaha k co největší automatizaci nejrůznějších úkonů. Jde o přirozený akt využívání dostupných technologií a bylo by škoda nevyužít výpočetní techniku, s níž je dnes možné zpracovat nejrůznější data během několika vteřin či minut, když bez ní by to zabralo i měsíce.

Geodézie a kartografie jsou plné činností vhodných k automatizaci pomocí počítače, ať už jde o běžný výpočet polygonového pořadu, vektorizaci libovolné mapy nebo zpracování dat z dálkového průzkumu země. Některé typy automatizace jsou algoritmicky jednoduché a jde pouze o přenesení výpočtů podle přesného vzorce z papíru do počítače (například zmíněný výpočet polygonového pořadu). Zrádnější je takové zpracování dat, kde je činnost prováděná člověkem sice jednoduchá, ale přece jen je při ní využit lidský rozum a zkušenost. Takovým zpracováním je právě vektorizace mapy. Lidské oko ihned rozezná, zda jde o mapovou značku pro město, nebo o chybu v tisku, či podobně vypadající písmeno v některém popisku. Jak ale tyto pro nás zřejmé věci naučit počítač?

Zatím nebyl vymyšlen takový algoritmus, který by nahradil lidský mozek s jeho nabytými zkušenostmi, a tudíž je nutné smířit se s často provizorními řešeními. Je tedy možné, a také velmi běžné, vytvořit aplikaci pro automatizaci nějaké činnosti, které určitým způsobem bude pomáhat člověk (například řízená klasifikace leteckých snímků).

Aplikace *Patterns Searching*, jejíž implementace je náplní této bakalářské práce s názvem *Automatické vyhledávání vzorů v obraze*, je právě takovým automatizačním programem, kterému musí asistovat člověk. Jak již název práce napovídá, tato aplikace má za úkol vyhledat v zadaném obrázku (například historické mapě) všechny výskyty určitého vzoru (mapové značky). Jejím využitím tedy bude vektorizace historických map.

Kvůli vzhledu tohoto druhu map, a hlavně faktu, že byly ručně kreslené, bylo předem jasné, že bezchybná vektorizace nebude možná. Aplikace tedy obsahuje také nástroj pomáhající uživateli s opravou vzniklých chyb. Algoritmus vyhledávací mapové značky je v této první verzi aplikace celkem jednoduchý a do budoucna by měl být určitě vylepšen, aby byl lidský faktor při vektorizaci co nejméně potřeba.

Tento dokument tedy popisuje vývoj aplikace *Patterns Searching* implementované v rámci bakalářské práce. Obsahuje jistý teoretický základ jak k samotnému vyhledávání, tak k vytváření grafického prostředí, nedílné součásti této aplikace. Další částí dokumentu je popis

vlastní implementace a také testování hotového programu. Důležitou přílohou k této práci je instalační a uživatelská příručka.

Kapitola 2

Popis problému, specifikace cíle

V této kapitole bude jemně nastíněn důvod vzniku aplikace pro automatické vyhledávání vzorů v obraze. V druhé části pak budou naznačeny hlavní rysy vznikajícího programu.

2.1 Analýza historických map

Historické mapy jsou velmi cenným zdrojem údajů o geografických poměrech v dobách minulých nejen na našem území. Těmito údaji může být například hustota osídlení jednotlivých regionů, propojenost sídel cestami či lokalizace těžby.

Získání těchto dat z papírové podoby map by bylo velice složité a v dnešní době výpočetní techniky hlavně zbytečně složité. K analýze se proto používá vhodný software, jako je například *ArcGIS*. K práci s takovouto aplikací je ale vhodné převést rastrovou reprezentaci dat na vektorovou.

2.2 Vektorizace

Vektorová reprezentace znamená uchování dat v podobě prvků (body, linie, polylinie, plocha, povrch, objem). Tyto prvky jsou určené souřadnicemi podle pravidel, která se mohou lišit pro různé topologické modely, ale také pro různé aplikace.

Převod rastrových dat na vektorové se nazývá vektorizace, která může být ruční nebo automatická. Oba typy vektorizace mají své klady a zápory. Nespornou výhodou automatického převodu je rychlost a v určitém směru také přesnost. Na druhou stranu ruční vektorizace je mnohem spolehlivější co se týče správnosti určení druhu jednotlivých prvků.

Vektorizovat mapu manuálně je tedy časově náročné, ale při automatizaci vzniká často mnoho chyb. Vhodné je tedy kombinovat obě metody, například kontrolovat výstup z automatické vektorizace člověkem.

2.3 Automatická vektorizace

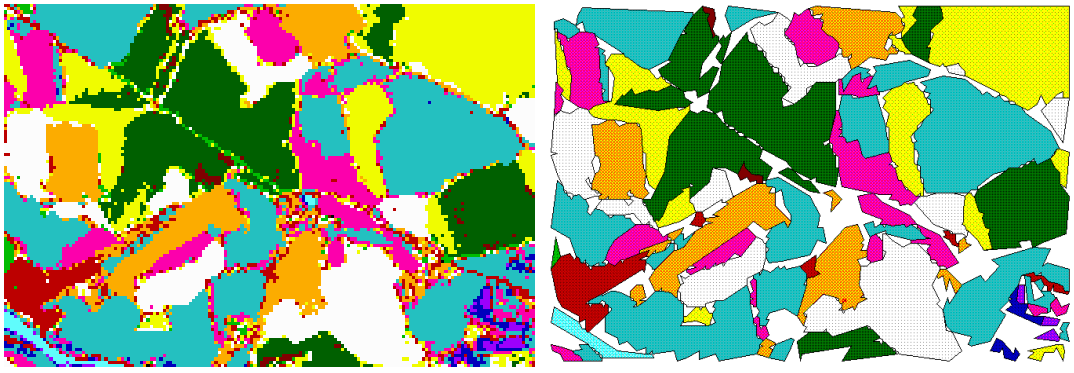
Automatická vektorizace může probíhat různými způsoby. Jakýkoliv rastr se dá automatický převést na vektorový model seskupením pixelů s podobnou (nebo stejnou) hodnotou

do ploch a tyto plochy se určí souřadnicemi rohových bodů. Výsledek takové vektorizace je vidět na obrázku 2.1 z [13].

(a) Původní obrázek (b) Vektorizovaný obrázek

Obrázek 2.1: Automatická vektorizace libovolného obrázku

Tento druh nejjednodušší vektorizace však při práci s mapami není užitečný. Další způsoby už využívají algoritmy navržené speciálně pro určité druhy rastrových dat, např. z leteckého snímkování (obr. 2.2 - algoritmus použitý pro tuto vektorizace je popsán v [3]). Tyto postupy mohou vytvářet již i linie či bodové prvky, na rozdíl od první metody, která vytvářela pouze plochy.



(a) Původní obrázek (b) Vektorizovaný obrázek

Obrázek 2.2: Automatická vektorizace upraveného rastru z leteckého snímkování

Speciálním druhem automatizace vektorizace je vyhledávání vzorů v obraze, kterým se zabývá právě tato práce. V principu jde o to, určit souřadnice všech výskytů určité značky na mapě. Postupem popsaným v tomto dokumentu vznikají pouze prvky typu bod, jde tedy o vyhledávání jednorozměrných (alespoň v měřítku mapy) objektů - převážně měst. Vektorizovaná mapa tudíž neobsahuje veškerý obsah map, ku příkladu výškopis či zobrazení cest a vodstva. Postup vektorizace je znázorněn na obrázku 2.3.

(a) Rastrová mapa (b) Vybraná značka (c) Vektorizovaná mapa

Obrázek 2.3: Postup vektorizace při vyhledávání vzorů

2.4 Požadované rysy aplikace pro vyhledávání vzorů

Hlavní funkcí takové aplikace je správné samotné vyhledávání, je však málo pravděpodobné, že by spolehlivost určování mapových značek u ručně vyráběných map (kterými historické mapy určitě jsou) byla stoprocentní. Nejde pouze o to najít všechny značky vyskytující se na mapě, ale také neoznačit chybně oblasti pouze podobné vzoru.

Dalším důležitým požadavkem je co nejvyšší rychlost. Vzhledem k značné velikosti obrázků zobrazující mapové listy lze tento požadavek splnit jen částečně (existuje určitá mez, pod kterou se s dnešním běžným výpočetním výkonem neparalelního systému, tedy jednoho počítače, nelze dostat) a často na úkor přesnosti.

Samozřejmou nutností je export souřadnic určených bodů do formátu, který může být dále zpracován, nejlépe načten přímo aplikací vhodnou pro analýzu mapy.

V dnešní době je samozřejmostí také uživatelsky přívětivé grafické rozhraní a pokud možno jednoduchá instalace i odinstalace.

Vlastnost, která není sice široce rozšířena, ale je velice užitečná, je možnost doplnění aplikace o takzvané zásuvné moduly neboli pluginy. Pro aplikaci, jejíž popis tvorby je hlavním tématem tohoto dokumentu, sloužící k vyhledávání vzorů, je vhodné, vytvořit možnost rozšíření především pro algoritmus určující, zda je oblast mapy požadovaným vzorem či nikoliv a také pro zmíněný export souřadnic do různých formátů.

Výčet všech požadovaných rysů, z nichž některé nejsou přímo nutností, je následující:

- přesnost vyhledávání - rozpoznání vzorů a také zamítnutí podobných oblastí
- vysoká rychlost vyhledávání
- export souřadnic do vhodného formátu
- uživatelsky přívětivé grafické rozhraní
- rozšiřitelnost některých částí pomocí zásuvných modulů (pluginů)
- prohlížeč obrázků s jednoduchým ovládáním

- manažer vzorů (mapových značek) - jednoduché určení a uložení nového vzoru, mazání vzorů, určení vztažného bodu vzoru
- možnost transformovat souřadnice při exportu
- určení transformace pomocí identických bodů - zadávání identických bodů klikáním na otevřenou mapu
- kontrola vyhledaných vzorů s možností opravy výsledků
- konfigurace vyhledávacího algoritmu
- správa projektů

Výsledná aplikace pro vyhledávání vzorů v mapách by tedy měla splňovat alespoň tyto základní požadavky. Je však možné, že ve všech ohledech nebude dosahovat optimálních výsledků, především kvůli časovému rozsahu určenému k implementaci.

Kapitola 3

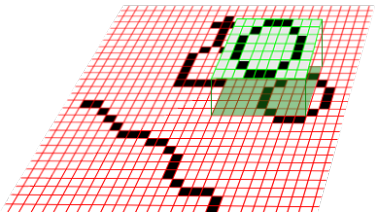
Analýza a návrh řešení

Tato kapitola je zaměřena především na popis technik a technologií použitých v implementaci aplikace zvané *Patterns Searching* (PS). Je v ní zmíněn základ teorie k určení podobnosti obrázků a také způsob práce s obrázky v jazyce Java. Nakonec jsou popsána základní fakta o grafickém rozhraní a s ním spjatou NetBeans platformou.

3.1 Vyhledávání vzorů

Vyhledávání vzorů je hlavní a nejdůležitější funkcí celé aplikace, její analýza byla tudíž nejsložitější a časově nejnáročnější. Od algoritmu vyhledávání se očekávají dvě zásadní věci, a to přesnost a rychlost. Je samozřejmé, že jsou tyto vlastnosti na sobě závislé, ve většině případů jsou nepřímo úměrné. Výsledná aplikace upřednostňuje rychlostní stránku věci, hlavně kvůli složitosti vylepšování algoritmů co se přesnosti týče.

Přesnost vyhledávání závisí na algoritmu použitého při porovnávání dvou obrazů, v tomto případě obrazu (mapy) a vzoru neboli masky. Tato maska se postupně posouvá po celém obraze. Po každém posunu tedy vzniká pro masku stejně velká část obrazu k porovnání (obr. 3.1). Určitým postupem se poté vypočítá, jak moc jsou si tyto rastry podobné.



Obrázek 3.1: Vyhledávání pomocí masky

Rozhodovací algoritmus, který určí, zda si jsou dva rastry podobné natolik, aby mohlo být řečeno, že jsou shodné, může být velice jednoduchý nebo naopak složitý algoritmus využívající neuronových sítí, fuzzy množin, nebo SVM (Support vector machines). Podrobná analýza a implementace těchto sofistikovanějších postupů se v této práci nevyskytuje, ale mohla by být náplní diplomové práce.

3.1.1 Podobnost dvou obrazů či jejich částí

V této sekci bude zmíněno několik metod možných k určování podobnosti dvou obrazů - masky a jí odpovídající části na prohledávaném obraze. Metody se dělí do dvou základních skupin: plošné a znakové. U obou těchto skupin výsledné rozhodnutí spočívá v porovnávání vypočítaného koeficientu s limitní hodnotou. Mohli by samozřejmě být základem pro získávání údajů z obrázků, které by poté byly zpracovány některým sofistikovanějším algoritmem zmíněným výše, to však není náplní této práce.

Následující přehled je výtahem z [14]. Tento článek se zabývá registrací obrazu[1], ale zmíněné metody se dají využít také při vyhledávání vzorů.

3.1.1.1 Znakově založené algoritmy (Feature based)

Znakově založené metody určení podobnosti, jak už název napovídá, pracují se znaky neboli objekty detekovanými ve scéně. Tyto znaky mohou být několika druhů podle dimenze:

- plošné znaky - 2D objekty s výraznými hranicemi (lesy, rybníky)
- liniové znaky - 1D objekty (cesty, řeky, hranice plošných objektů)
- bodové znaky - 0D objekty (lomy na objektech kontrastujících s pozadím, osamocené předměty, průsečíky linií)

Všechny metody, ať už využívající plochy, linie či body, se skládají ze dvou částí - detekce znaků a lícování znaků. První fáze je tedy nalezení výrazných objektů v obou obrazech, a ty se ve druhé fázi identifikují a vytvářejí páry shodných objektů.

Detekce znaků Detekování znaků se velice liší pro různé typy (dimenze) znaků, na rozdíl od následného lícování, kde již rozdíly nejsou tak markantní.

Plošné znaky se rozpoznávají metodami segmentace obrazu, které rozdělují zpracovávaný obraz na oblasti se společnými vlastnostmi. Mezi nejjednodušší, avšak pro tento účel nevhodnou, metodu tohoto typu patří prahování (thresholding), což je k jinému účelu v navrhované aplikaci také použito. Výčet a popis dalších metod je dostupný například v [11].

Liniové znaky jsou detekovány pomocí klasických filtrů pro detekci hran jako je Cannyho hranový detektor (viz [9]) nebo detektor LoG (Laplacian of Gaussian popsaný např. v [2]).

[1] Registrace obrazu (obecně signálů) je proces, při kterém se dva či více obrazů stejné scény, pořízených z různých míst (časů), spojují tak, aby výsledný obraz odpovídal skutečnosti. Jde tedy o to, najít ve scéně odpovídající si části či objekty.

Bodové znaky je možné získat, kromě využití speciálních detektorů, také vytvořením bodového popisu plošného objektu (těžiště), či jako průsečík detekovaných linií. V některých případech je ale takovéto vytváření bodů značně nevhodné (registrace fotografií budov) a jako výhodnější se jeví detekování rohů. Tato úloha je pro lidské oko velice jednoduchá, ale matematická definice rohu tak snadná není. Přesto existuje velké množství algoritmů pro detekci rohů, za všechny například metoda *SUSAN* nebo *FAST* detektor[2]. Rozsáhlý seznam s popisem dalších metod je k vidění v [10] a přehledné srovnání některých algoritmů je k nahlédnutí v [7].

Lícování znaků Po provedení detekce jsou k dispozici dvě skupiny znaků (bodů), které je potřeba spárovat buď pomocí prostorových vztahů, nebo s využitím různých deskriptorů. Příkladem prvního postupu je například metoda, kdy se počítá s množstvím bodů z jedné množiny, které jsou v dostatečné blízkosti bodů z množiny druhé.

Při užití druhého postupu by měli deskriptory vyhovět několika podmínkám, což ne vždy je možné a je třeba vytvořit rozumný kompromis:

- neměnnost - deskriptory dvou sobě odpovídajícím znakům (ze dvou obrazů) musí být shodné
- jedinečnost - dva různé znaky musí mít různé deskriptory
- stabilita - deskriptor mírně deformovaného znaku musí být podobný deskriptoru znaku původního
- nezávislost - pokud je popis znaku sdružen do vektoru, musejí být jeho prvky na sobě nezávislé

Znaky jsou poté spárovány podle podobnosti jejich deskriptorů, které je možné definovat různými způsoby, z nichž některé budou zmíněny níže.

Korelační koeficient intenzity je nejjednodušší způsob popisu znaku. Jedná se o výpočet normalizovaného korelačního (vzájemného[3]) koeficientu částí obrazů v okolí znaku. Funkční hodnotou ve výpočtu je světlost barvy daného pixelu. Koeficient se vypočítá podle vzorce 3.1, kde $\overline{f}$ a $\overline{g}$ jsou aritmetické průměry hodnot pixelů v dané oblasti registrovaných obrázků a $f_{x,y}$ a $g_{x,y}$ jsou hodnoty pixelů na pozicích daných souřadnicemi x a y.

$$CC = \frac{\sum_x \sum_y (f_{x,y} - \overline{f}) \cdot (g_{x,y} - \overline{g})}{\sqrt{\sum_x \sum_y (f_{x,y} - \overline{f})^2 \cdot \sum_x \sum_y (g_{x,y} - \overline{g})^2}} \tag{3.1}$$

[2]Detektor znaků FAST zkoumá body na Bresenhemově kružnici se středem v bodě (jádře), o němž se rozhoduje zda je rohem, a poloměrem r. Na této kružnici musí být buď n bodů světlejších o t či tmavších o t než je jádro. Pokud je toto splněno, je bod prohlášen rohem. Vhodné hodnoty konstant jsou $r = 3$, což je kružnice o obvodu 16 px, a $n = 9$, čímž se zajistí aby se neurčovali jako rohy linie.

[3]Vzájemná korelace (cross-correlation) je způsob určení podobnosti dvou signálů (obrazů).

Intuitivní deskriptory popsané v tomto odstavci většinou nesplňují podmínky pro invariantní deskriptory a pro aplikaci na porovnávání dvou malých rastrů (maska a oblast pod ní) nejsou vhodné. Použitelné jsou ku příkladu pro registraci leteckých snímků.

Takovým intuitivním deskriptorem je Sesterova metoda pro užití převážně u lesů, které se popíší pomocí jejich obvodu, kompaktnosti, počtu děr a informací o konvexní obálce. Pro registraci hvězd je možné použít Murtaghuv postup pracující s prostorovým rozdělením okolních znaků (hvězd). Dalším příkladem je popis znaku vzniklého průsečíkem pomocí nejdelší struktury a úhlů ostatních struktur podílejících se na průniku, který popsal Brzakovic.

Invarianty založené na momentech tvoří rozsáhlou skupinou deskriptorů používaných pro popis oblastí s uzavřenými hranicemi. Často se používají v kombinaci s ostatními metodami, jako například Holmova metoda popisující plochy pomocí jejich obvodu, plochy, celistvosti, momentů[4] a momentových invariantů[5].

3.1.1.2 Plošně založené algoritmy (Area based)

Plošně založené algoritmy spojují fázi detekce a lícování znaků. Většina těchto metod využívá přímo hodnoty pixelů. To sice celý výpočet zjednodušuje, naproti tomu jsou vypočítané koeficienty citlivé na šum či odlišnou celkovou světlost obrazu.

Korelační metody Takto nazvané metody jsou založené na vzájemné korelaci a jejích modifikacích. Jde o určení míry celkové podobnosti dvou obdélníkových oblastí. Vlastní *normalizovaný vzájemný korelační koeficient* se vypočítá podle vztahu 3.1.

Podobné koeficienty jako je korelační jsou *SAD* a *SSD*. První z nich vznikne součtem absolutních rozdílů hodnot pixelů (Sum of Absolute Differences) a druhý součtem druhých mocnin rozdílů (Sum of Squared Differences). Tyto koeficienty jsou z pohledu počítače snadněji a tudíž rychleji vypočitatelné, hůře se však určuje hraniční hodnota pro přijmutí oblasti jako dostatečně podobné.

Metodou z této skupiny, která využívá výpočtu *SAD* koeficientu, je *SSDA* neboli sequential similarity detection algorithm (algoritmus postupné detekce podobnosti). Ten počítá sumu absolutních rozdílů hodnot pixelů a při překročení určitého prahu oblast zavrhne jako málo podobnou. Tento postup je méně přesný než výpočet korelačního koeficientu, ale je rychlejší díky jednoduchosti operací při výpočtu (sčítání a odčítání oproti násobení, dělení a odmocňování) a možnosti včasného ukončení výpočtu.

Furierovy metody Algoritmy založené na Furierově transformaci jsou vhodné například u obrazů obsahující frekvenčně závislý šum. Konkrétní metoda z této skupiny je *fázová korelace*, která dosahuje proti běžné korelaci mnohem lepších výsledků při přítomnosti frekvenčně závislého šumu a jiné světelnosti obrazů.

[4] Ve většině případů se používají centrální momenty, které jsou nezávislé na posunutí objektu. Jsou však závislé na otočení a změně měřítka

[5] Momentové invarianty jsou popisné charakteristiky nezávislé na rotaci nebo měřítku. Například sedm invariantů vzhledem k rotaci odvodil Hu.

Vzájemná informace Vzájemná informace je míra statistické závislosti dvou množin dat. Vzorec pro výpočet informace:

$$I(A,B) = -\sum_{i=1}^{L} p_i(A)\log p_i(A) - \sum_{j=1}^{L} p_j(B)\log p_j(B) + \sum_{i=1}^{L}\sum_{j=1}^{L} p_{i,j}(A,B)\log p_{i,j}(A,B), \quad (3.2)$$

kde $p_i = \frac{n_i}{N}$ je relativní četnost jasu i v obraze a $p_{i,j}(A,B) = \frac{n_{i,j}}{N}$ je sdružená relativní četnost , kde $n_{i,j}$ je počet výskytů dvojic (i,j), vychází ze vzorce:

$$I(A,B) = H(A) + H(B) - H(A,B), \quad (3.3)$$

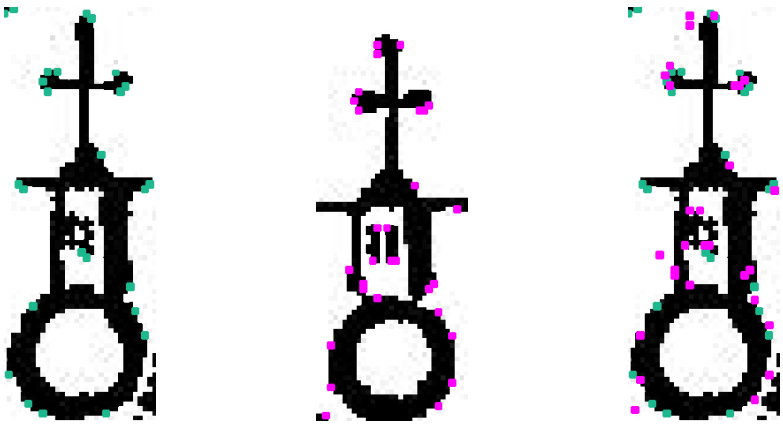
kde H je entropie[6].

Při registraci obrazu se jednoduše určuje maximum vzájemné informace, při vyhledávání vzorů je obtížné určit hranici dostatečné podobnosti. To je v tomto případě způsobeno hlavně specifickým vzhledem prohledávaných obrázků (historické mapy) a obecně ne příliš velkým rozdílem mezi hodnotami vzájemné informace podobných a rozdílných oblastí.

3.1.1.3 Zvolené řešení

Během analýzy bylo naimplementováno a vyzkoušeno na konkrétních datech (mapový list Müllerovy mapy) několik možností určení podobnosti dvou rastrů. Zdaleka nebyly vyzkoušeny všechny možnosti, spíše se jednalo o pár vybraných metod.

FAST detektor znaků Po vyzkoušení detekce bodových znaků a zhodnocení výsledků se tento způsob ukázal jako nevhodný pro využití v takovéto aplikaci. Kvůli faktu, že byly historické mapy tvořeny ručně, jsou často mezi značkami značné rozdíly a ty se velmi projeví v nalezených znacích. Příklad dvou mapových značek a na nich detekovaných znaků je vidět na obrázku 3.2.



(a) Sada A (b) Sada B (c) A a B

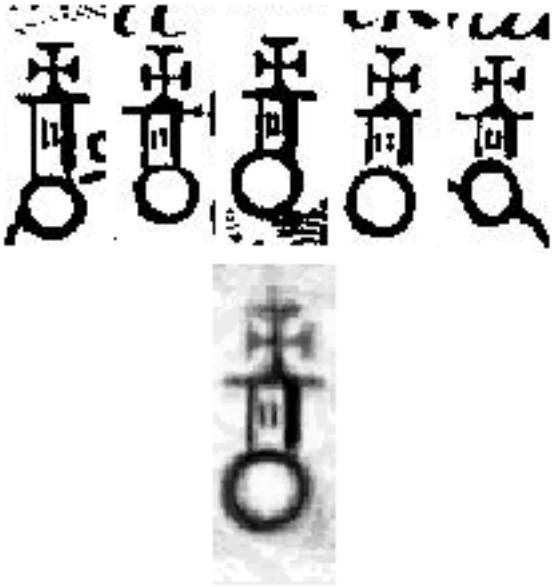
Obrázek 3.2: Detekce znaků algoritmem FAST feature detection

[6]Entropie je míra neurčitosti systému.

Je zřetelné, že některé body z množin znaků by mohly být prohlášené za odpovídající, na druhou stranu jsou si tyto dvě mapové značky velice podobné, a to není na mapách tohoto druhu samozřejmostí. Také je vidět nesouhlasnost bodů ve spodní části značek (na kolečku) a uvážíme-li existenci značek sestávajících pouze z těchto koleček, bylo by velice obtížné vyladit algoritmus tak, aby si s těmito nedokonalostmi poradil a zároveň chybně neurčoval některé oblasti jako značky.

Vzájemná informace Tato metoda by mohla být vhodná při vyhledávání složitějších vzorů, než jsou mapové značky na historických mapách. Při zkoušení algoritmu založeného na výpočtu informace se objevilo jako problémové určení rozhodující meze dostatečné podobnosti. Dalo by se provést jakési naučení pomocí trénovací množiny a uživatel by vždy používal mez zjištěnou tímto postupem. Vzhledem k malé variabilitě koeficientu i při větší změně porovnávaných obrazů byla však tato metoda zamítnuta.

Vzájemný korelační koeficient Tímto postupem bylo obecně dosaženo uspokojivého poměru úspěšnost ku jednoduchosti a rychlosti. Lepší výsledky jsou také dávány při vytvoření hledaného vzoru z několika různých obrázků reprezentujících daný vzor, jak je znázorněno na obrázku 3.3.



Obrázek 3.3: Jednotlivé vzory a kombinovaný vzor

Tento vytvořený vzor je nazván kombinovaný vzor (konkrétní vzor na obr. 3.3 je složen z přibližně třiceti jednotlivých vzorů), původní vzory jsou sdružovány do skupiny kterou navenek reprezentuje právě kombinovaný vzor. Jednotlivé vzory jsou složeny do kombinovaného aritmetickým průměrem daným vzorcem

$$v_c(x,y) = \frac{1}{N}\sum_{i=1}^{N} v_i(x,y) \tag{3.4}$$

kde $v_c(x, y)$ je hodnota pixelu kombinovaného vzoru na souřadnicích $[x, y]$, $v_i(x, y)$ je hodnota pixelu i-tého vzoru a N je počet vzorů.

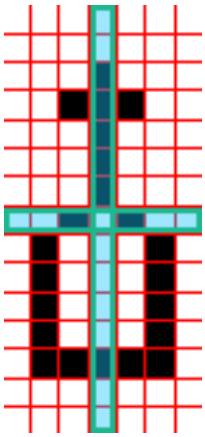
Bylo by do budoucna vhodné doplnit tuto metodu, která by zajišťovala rychlost, nějakým úspěšnějším algoritmem zajišťujícím přesnost.

3.1.2 Rychlost vyhledávání

Vzhledem k značné velikosti prohledávaných obrázků je optimalizace algoritmů velmi důležitá. Po první implementaci vyhledávacího algoritmu byl přibližný čas prohledávání obrázku o velikosti 54 Mpx 10 minut (na konfiguraci *Intel Core2 Duo T7500 2.20 GHz, 2 GB RAM, Ubuntu 9.04 32-bit*). Tento čas nebyl uspokojivý i vzhledem k tomu, že se tak dlouho hledal nejmenší možný vzor a na mapě se vyskytují značky i několikrát větší.

Jedna z možností, jak celý proces urychlit, je prohledání zmenšené mapy a teprve poté hledat na originálním obraze vzory v podezřelých oblastech. Tento mechanismus byl nakonec zamítnut, jelikož bylo vyhledávání zrychleno jiným způsobem, avšak mohl by být doimplementován jako součást diplomové práce, kvůli dalšímu urychlení především na slabších počítačových sestavách.

V rámci urychlení výpočtu byla do algoritmu pro určení podobnosti dodána funkce nazvaná *předvýpočet řezu*. Jde o intuitivní zajištění rychlejšího zamítnutí oblastí, které nejsou vzory, pomocí výpočtu korelačního koeficientu nejprve pro vertikální a horizontální řezy rastrů o tloušťce 1px. Princip je znázorněn na obrázku 3.4. Pokud oba korelační koeficienty splňují limit pro dostatečnou podobnost, vypočítá se koeficient znovu pro celý vzor, pokud nesplňují, výpočet se ukončí a maska se na obraze posune o jeden pixel. Tato metoda urychlila vyhledávání na výše zmíněné konfiguraci přibližně na 6 minut, což ale stáje není vyhovující. Je také ale nutno podotknout, že tento postup mírně zhorší úspěšnost - jak moc se liší pro různé vzory.

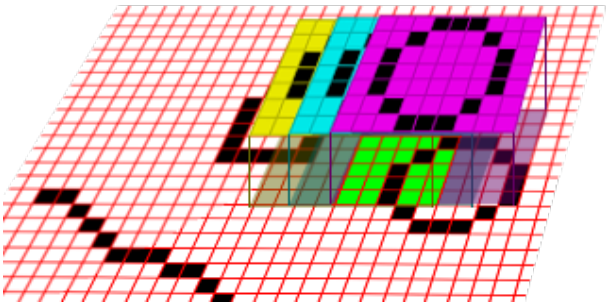


Obrázek 3.4: Vertikální a horizontální řez maskou

Pro zjištění největších časových ztrát při procesu vyhledávání bylo zapotřebí algoritmus analyzovat pomocí *NetBeans Profileru*[7]. Tímto bylo zjištěno, že většinu času program stráví

[7] Profiler je nástroj pro dynamickou analýzu programu a zjišťování jeho chování z pohledu využité paměti nebo času stráveného v určitých funkcích.

v metodě[8] `getRGB` třídy `BufferedImage`. Bližší popis této metody (metoda jako taková není složitá, ale způsobem, jakým byla použita, se provádělo mnoho duplicitních výpočtů) bude popsán v sekci 3.2.1. Tato metoda byla volána pokaždé, když byla pro výpočet potřeba hodnota určitého pixelu a vzhledem ke způsobu prohledávání obrázku pomocí masky byla volána pro jeden pixel více než jednou (tudíž se více než jednou zbytečně provedl její obsah) jak je zobrazeno na obr. 3.5. Tento problém je tedy vyřešen zkopírováním dat (hodnot pixelů) z objektu typu `BufferedImage` do běžného dvojrozměrného pole. Tím je odstraněno duplicitní provádění výpočtu v metodě `getRGB`, jelikož se hodnoty pixelů získávají přímo z vytvořeného pole, jehož indexace není vůbec časově náročná. Vytvoření pole před začátkem vyhledávání zabere nějaký čas (10-20 vteřin), což je však proti urychlení celého procesu zanedbatelná nevýhoda. Čas zpracování obrázku se touto metodou zkrátil o několik řádů a zabere tedy kolem 20 vteřin (s aplikováním *předvýpočtu řezu*).



Obrázek 3.5: Naznačení postupu prohledávání - barevné masky znázorňují posuv a sytě zelená oblast na prohledávaném obrázku označuje plochu, ve které je v tomto případě každý pixel využit pro 3 výpočty podobnosti

Využitím zmíněných metod je tedy při určení podobnosti korelačním koeficientem rychlost dostačující (pro výše zmíněnou konfiguraci). Do budoucna je možné další zrychlování například uvedeným prohledáním zmenšené mapy, rozdělením procesu do více vláken podle hardwarové výbavy uživatelova počítače nebo využitím výpočetního výkonu grafické karty.

3.1.3 Vyhledávání na okrajích obrázku

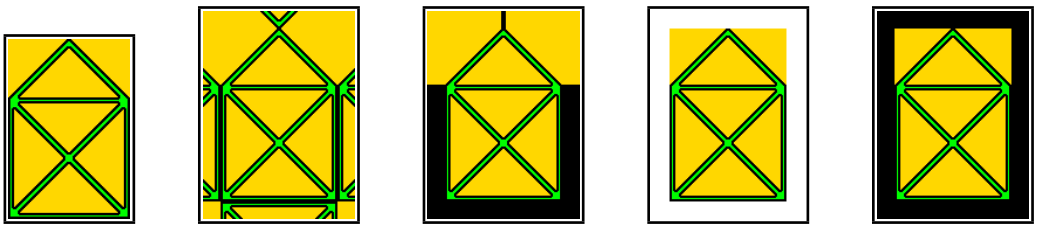
Při vyhledávání je vždy nutné vyřešit chování masky na okrajích obrázku. Nejjednodušším řešením je v podstatě ignorování okrajů. To znamená, že se maska pohybuje po obrázku tak, že nikdy nepřesahuje přes jeho okraje. Tímto postupem se však mohou opomenout některé vzory, které sice nejsou na obrázku zobrazeny celé, ale jejich část je dostatečně velká, aby byly algoritmem rozpoznány. Proto se chování masky může upravit tak aby mohla hranice obrázku mírně přesahovat.

V tom případě je nutné masce předložit určitá data i tam, kde již obrázek není. Tvorba těchto dat se řídí podle zvoleného pravidla z několika možností, jež jsou také zobrazeny na obrázku 3.6:

- zrcadlové (mirror) - kolem obrázku se vytvoří jeho zrcadlové kopie

[8] Metoda je funkce třídy.

- roztažené (stretch) - hraniční pixely se roztáhnou směrem kolmo na okraj obrázku
- konstantní barva - kolem obrázku mají pixely konstantní hodnotu



(a) Originál (b) Zrcadlové (c) Roztažené (d) Konstantní barva (bílá) (e) Konstantní barva (černá)

Obrázek 3.6: Chování na okrajích obrázků

3.2 Práce s obrázky v jazyce Java

Běžná práce s obrázky v jazyce Java je velice jednoduchá, ať už jde o jejich načítání, vykreslování, vytváření či ukládání. Nejpoužívanější třída pro reprezentaci obrázku se nazývá `BufferedImage`. Problém nastává při práci s velkými obrázky, kdy je třeba šetřit pamětí co nejvíce to jde. Další, ne příliš intuitivní, záležitost je práce s barvami jako hodnotami pixelů obrázků. S těmito barvami se neoperuje v podobě instancí třídy `Color`, jak by se člověk mohl domnívat, ale v podobě čísel složených z jednotlivých barevných složek.

Popis užívání takovýchto barev a také práce s velkými obrázky je náplní této sekce dokumentu.

3.2.1 Barvy v obrázcích

Třída `BufferedImage` zapouzdřuje především data obrázku ve formě `Rasteru` a nástroj pro převod hodnoty pixelu do složek jednotlivých barev a složky průhlednosti. Tento nástroj reprezentuje třída `ColorModel`. `Raster` obsahuje veškeré hodnoty obsažené v obrázku ve struktuře `DataBuffer`. Způsob jak se z `DataBufferu` získávají požadovaná data popisuje `SampleModel`.

Obrázek typu `BufferedImage` může existovat v podobě několika typů, daných způsobem uložení dat a množství paměti potřebného k uložení podoby jednoho pixelu. Tyto typy jsou následující:

- TYPE_3BYTE_BGR - tři osmi bitové složky (modrá, zelená, červená)
- TYPE_4BYTE_ABGR - čtyři osmi bitové složky (modrá, zelená, červená, průhledná)
- TYPE_BYTE_BINARY - jedno, dvou nebo čtyř bitový obrázek, jehož každý pixel je uložen do jednoho bajtu
- TYPE_BYTE_GRAY - neindexovaný šedotónový obrázek s jedním bajtem pro každý pixel
- TYPE_BYTE_INDEXED - indexovaný obrázek s jedním bajtem pro každý pixel

- TYPE_CUSTOM - nerozpoznaný typ
- TYPE_INT_ARGB - 4 osmi bitové složky (průhledná, červená, zelená, modrá) reprezentované celočíselným pixelem
- TYPE_INT_BGR - 3 osmi bitové složky (modrá, zelená, červená) reprezentované celočíselným pixelem
- TYPE_INT_RGB - 3 osmi bitové složky (červená, zelená, modrá) reprezentované celočíselným pixelem
- TYPE_USHORT_555_RGB - 3 pěti bitové složky (červená, zelená, modrá)
- TYPE_USHORT_565_RGB - 3 složky s proměnným počtem bitů (pěti bitová červená, šesti bitová zelená, pěti bitová modrá)
- TYPE_USHORT_GRAY - neindexovaný šedotónový obrázek s pixely ukládanými jako `unsigned short`

Některé z nich je možné vidět na obr. 3.7. Tento obrázek je upraveným výstupem z pomocného programu, který byl vytvořen jen pro zobrazení možných typů obrázků v Javě. Program načte obrázek 3.7a a převede ho do několika typů. Vpravo od každé varianty obrázku je vypsaná hodnota vracená metodou třídy `BufferedImage` `getRGB` a za ní se nachází výpis jednotlivých složek (červená, zelená, modrá). Poslední z variant je vytvořena pomocí vlastní funkce `countGrey` nacházející se ve třídě `Colors`. Tato metoda vypočítá šedou barvu ze tří barevných složek pomocí vzorce

$$v = 0.299 \cdot r + 0.587 \cdot g + 0.114 \cdot b, \tag{3.5}$$

kde v je hodnota šedé barvy, r hodnota červené barvy, g hodnota zelené barvy a b barvy modré. Tento vzorec je určen tak, aby výsledná barva světlostí co nejlépe vyjadřovala původní barvu a koeficienty jsou zvoleny podle míry citlivosti lidského oka na jednotlivé složky. Tento rozdíl však není v aplikaci pro vyhledávání vzorů, kde se navíc pracuje převážně s černobílými obrázky, nijak důležitý.

Jak ukazuje obr. 3.7 mezi cvičnými paletami převedenými na šedotónové jsou drobné odlišnosti. Z rozdílů mezi paletou převedenou vlastní funkcí a těmi převedenými změnou typu obrázku je patrné, že druhý ze zmíněných způsobů převodu vytváří ze zelených odstínů ještě světlejší šedou a z modrých a červených naopak o něco tmavší.

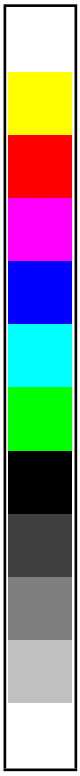
Jak se již dalo vytušit, návratová hodnota metody `getRGB` je typu `int` a obsahuje v sobě všechny tři (s průhlednou složkou čtyři) barevné složky. Tato metoda není klasickým *getterem*[9], ale návratovou hodnotu přímo vytváří (přesněji řečeno tuto hodnotu vypočítá metoda `getRGB` třídy `ColorModel`). To také způsobí značné zpomalení aplikace při jejím častém a také duplicitním volání, jak bylo uvedeno v sekci 3.1.2. Výpočet hodnoty reprezentující pixel se provádí následovně

$$p = (a \ll 24) \mid (r \ll 16) \mid (g \ll 8) \mid (b \ll 0) \tag{3.6}$$

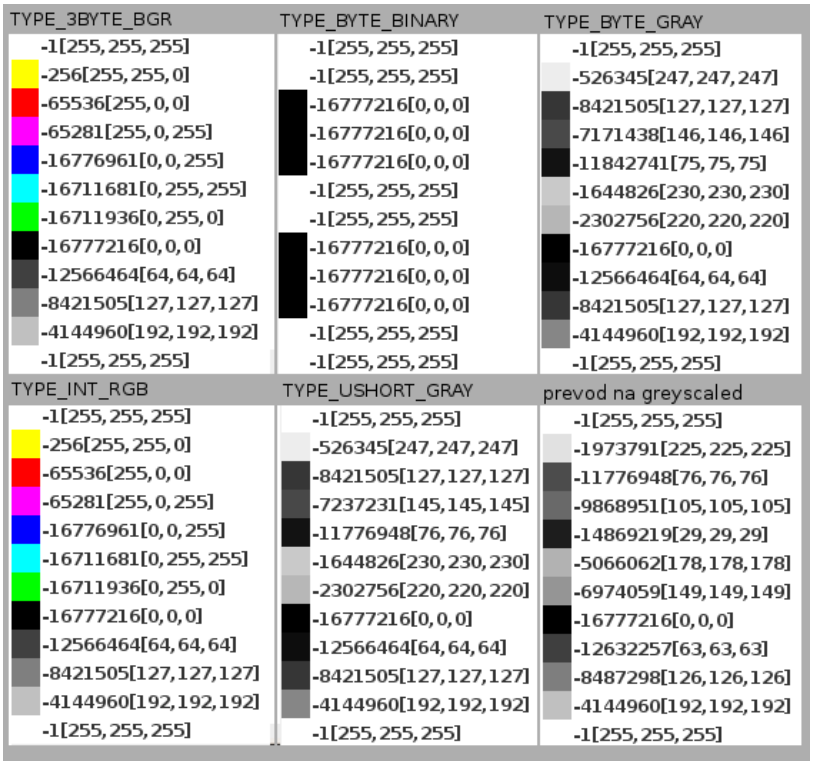
a naopak získání jednotlivých složek z této hodnoty se provede podle vzorců

$$\begin{aligned} a &= (p \,\&\, 0xff000000) \gg 24 \\ r &= (p \,\&\, 0x00ff0000) \gg 16 \\ g &= (p \,\&\, 0x0000ff00) \gg 8 \\ b &= (p \,\&\, 0x000000ff) \end{aligned}$$

[9] Getter je typ metody která vrací hodnotu určité proměnné.



(a) Orig.



(b) Varianty

Obrázek 3.7: Barevné typy obrázků v jazyce Java

V těchto vzorcích je p hodnota pixelu a a, r, g, b jsou jednotlivé barevné složky (průhlednost, červená, zelená, modrá).

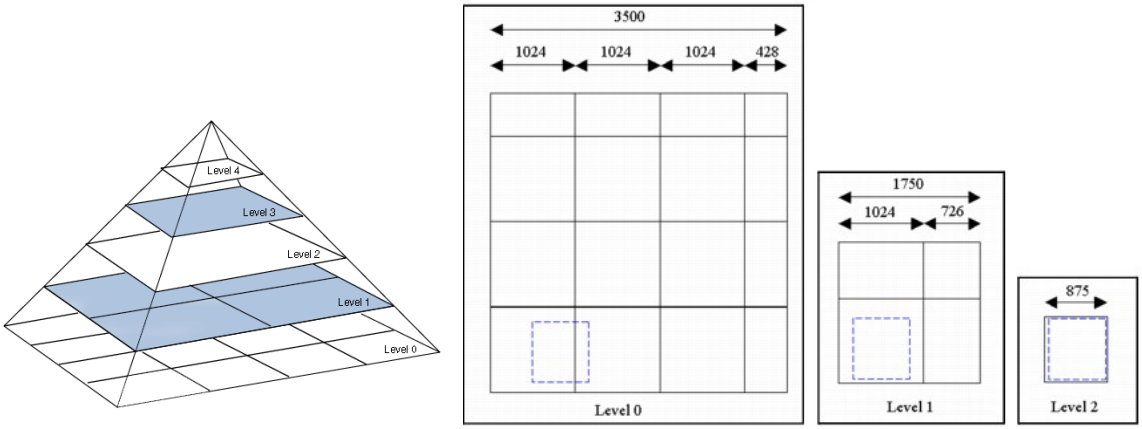
3.2.2 JAI

Pro pokročilejší práci s obrázky je vhodné použít knihovnu *JAI* neboli *Java Advanced Imaging API*. Jedná se o knihovnu rozšiřující možnosti ve zpracování obrazu především velkým množstvím operací, které je možné na obrázky aplikovat, nebo nástroji pro zvyšování výkonu, jako je například dlaždicování[10] (tiling).

Nutnost využití této knihovny vznikla hlavně kvůli značným paměťovým nárokům práce s velkými obrázky a kvůli zrychlení vykreslování. Z celé knihovny je použit prakticky jen jeden nástroj, a to již zmíněné dlaždicování. V budoucnu by mohlo býti užitečné zapracování dalších funkcí v rámci diplomové práce. Dlaždicování jako takové vylepší hlavně výkonovou stránku vykreslování obrázku, co se paměti týče, je samotné celkem bezmocné. Pro správu dlaždic v paměti je třeba využít třídu implementující rozhraní `TileCache`, které se dá nastavit počet dlaždic držených v paměti, ale také maximální paměť, kterou mohou zabrat. Výhoda

[10] Dlaždicování neboli tiling je metoda zpracovávání obrazu po částech k urychlení výkonu a omezení paměťových nároků.

dlaždicování se vytrácí při zobrazení zmenšeného obrázku, kdy se musí stejně vykreslit celý obrázek a to vede ke zpomalení aplikace. Pro tento případ by bylo vhodné implementovat podporu takzvané pyramidy (obr. 3.8 vypůjčený z [4]) - vytvoření obrázků o menším rozlišení, které jsou následně zobrazovány místo zmenšeného původního obrázku.



(a) Pyramida

(b) Dlaždicování (tečkovaný obdélník znázorňuje zobrazovanou oblast)

Obrázek 3.8: Zobrazení dlaždic z pyramidového obrázku při zmenšování obrázku

3.3 NetBeans platforma

NetBeans (NB) platforma je framework pro vývoj swingových aplikací, který obsahuje správu oken, propojení akcí s položkami v menu nebo s tlačítky v nástrojové liště. Tyto nástroje vytvoří příjemně ovladatelnou aplikaci, jejíž výhody sice uživatel vůbec nemusí postřehnout, ale jejich absence by si jistě dříve či později všiml. Naprogramování možností jaké NB platforma obsahuje by bylo velice časově náročné.

Při tvorbě této podkapitoly týkající se NetBeans platformy bylo čerpáno z [1], [8], [5] a [6].

3.3.1 Struktura aplikace

NB Platform aplikace je vždy složena z jednoho či více modulů. Tyto moduly se zpravidla seskupují do takzvaných Module Suite.

Moduly Moduly pro aplikace založené na NetBeans platformě jsou Java archivy (.jar soubory) distribuované jako balíčky s příponou .nbm. Tyto balíčky se dají do aplikace nainstalovat pomocí *Plugin manažeru*.

Z pohledu programátora jsou moduly speciálním druhem projektů, které využívají NetBeans API a je možné z nich vytvářet pluginy do NetBeans-based aplikací, jejichž zákla-

dem je vždy plugin *RCP Platform*[11] zprostředkující veškerou funkcionalitu platformy. Doinstalováním různých modulů do základní formy aplikace je možné zcela změnit její chování a funkčnost.

Moduly se dělí na tři typy podle okamžiku načítání - regular, autoload a eager. Typ **regular** se načítá při startu aplikace a je tedy vhodný pro nástroje a funkce, které jsou uživatelem 'viditelné' a často používané. Naopak **autoload** se používá pro knihovny a jeho načtení proběhne až ve chvíli, kdy jsou třídy v něm obsažené potřeba. Posledním typem je **eager**, jenž se načte hned jak je to možné. Například modul závisející na některých dalších se načte až v okamžiku, kdy jsou k dispozici všechny potřebné moduly.

Suity modulů (Module Suites) Suity jsou, jak již název napovídá, soubory modulů. Jejich obsahem jsou pouze konfigurační soubory a informace o modulech jim náležících. Suity mohou zaštiťovat moduly tvořící samostatnou aplikaci, nebo pouze sdružují moduly z podobného oboru pro jejich lepší správu a pro přehlednost.

Mezi vlastnosti suity tvořící samostatnou aplikaci patří mimo jiné informace o splash screen (úvodní obrazovce), ikona a název aplikace a také je možné upravit nápisy vyskytující se v obecných grafických prvcích základu aplikace (např. nadpis okna *Options*).

Library Wrapper Library wrapper je jednoduchý modul sloužící jako knihovna. Takové moduly jsou zpravidla typu *autoload*. Tyto moduly musí být vytvořeny vždy, když je potřeba využít externího JAR souboru, jelikož moduly mohou využívat třídy pouze z jiných modulů.

3.3.2 Layer soubory

Některé moduly obsahují mezi svými zdrojovými kódy také soubor *layer.xml*. Tento soubor mimo jiné umožňuje přidávat položky do menu a přiřazovat jim požadované akce, akce spojovat s klávesovými zkratkami, upravovat nástrojové lišty či spravovat JDBC[12] připojení. Při startu aplikace se spojí všechny layer soubory jednotlivých modulů v pořadí inicializace modulů.

Struktura souboru je podobná struktuře souborového systému a také z něj přebírá jména položek (filesystem, folder, file). Konkrétní příklad užití je vidět v ukázce 3.1, kde se nejprve zaregistruje akce (jakou je reprezentována třídou určuje následující tag `attr` (`csv.actions.CsvExportAction`)) do složky *Actions/File* na řádce 4, poté se na řádce 14 a 23 vytvoří položka v menu *File -> Export* odkazující na tuto akci a tlačítko na nástrojové liště *Export*.

Pomocí dalších xml tagů je možné například napozicovat jakýkoliv prvek způsobem, jaký je vidět na řádkách 13 a 16.

Jak je možné si v uvedeném příkladu všimnout, soubory zde mohou mít dvě různé přípony, a to .instance a .shadow. První z nich znamená klasické vytvoření objektu. Přípona .shadow určuje, že se nevytvoří nový objekt, ale použije se již vytvořený, na něhož bude

[11] RCP je zkratkou pro *Rich Client Platform*, platformu usnadňující programátorům práci se správou menu, nástrojových lišt, oken, nastavení, atd.

[12] JDBC (Java Database Connectivity) je API jazyka Java pro jednotný přístup k relačním databázím.

tento soubor pouze ukazovat (ekvivalent zástupce nebo symbolického linku). O jaký soubor se bude jednat je určeno tagem `attr` s názvem *originalFile*.

Díky tomuto systému je například možné, aby nový modul sám pro sebe přidal položku do menu aplikaci, o které nic neví, což je velice užitečné.

```
<filesystem>
    <folder name="Actions">
        <folder name="File">
            <file name="csv-actions-CsvExportAction.instance">
                <attr name="delegate"
                      newvalue="csv.actions.CsvExportAction"/>
                ...
            </file>
        </folder>
    </folder>
    <folder name="Menu">
        <folder name="File">
            <folder name="Export">
                <attr name="position" intvalue="780"/>
                <file name="csv-actions-CsvExportAction.shadow">
                    <attr name="originalFile"
                          stringvalue="Actions/File/csv-actions-
                              CsvExportAction.instance"/>
                    <attr name="position" intvalue="320"/>
                </file>
            </folder>
        </folder>
    </folder>
    <folder name="Toolbars">
        <folder name="Export">
            <file name="csv-actions-CsvExportAction.shadow"> ... </file>
        </folder>
    </folder>
</filesystem>
```

Ukázka 3.1: Struktura layer souboru

3.3.3 Lookup a registrace objektů

Lookup neboli celým názvem *Default Lookup* je seznam tříd a jejich instancí, které byly registrovány jedním ze tří způsobů:

1. použití anotace `ServiceProvider`
2. vytvořením souboru v *META-INF/sevices*
3. přidávním souboru do *Services/Hidden* v layer souboru

Tyto metody budou popsány níže s využitím příkladu 3.2, bude tedy registrována třída `MyClass` implementující rozhraní `MyInterface`. Za použití tohoto příkladu bude také přiblížena vlastní funkce lookupu.

```
1  package org.foo.mymodule;
2
3  public class MyClass implements MyInterface {
4      @override
5      public void printName() {
6          System.out.println("I am class MyClass!!!");
7      }
8  }
9
10 ------------------------------------------
11 package org.foo.apimodule;
12
13 public interface MyInterface {
14     void printName();
15 }
```

Ukázka 3.2: Třída MyClass a rozhraní MyInterface

Ad 1) Tento způsob registrace je ve výsledku totožný s metodou následující (je ale pohodlnější), jelikož NetBeans při kompilaci vytvoří soubory v *META-INF/sevices* tak, jak je potřeba. Příklad použití je znázorněn v ukázce 3.3.

```
1  @ServiceProvider{service=MyInterface.class}
2  public class MyClass implements MyInterface {
3      @override
4      public void printName() {
5          System.out.println("I am class MyClass!!!");
6      }
7  }
```

Ukázka 3.3: Registrace pomocí anotace ServiceProvider

Ad 2) Pro použití této metody je třeba vytvořit adresář *META-INF/services* v kořenovém adresáři zdrojových kódů. Do něj se poté umístí soubor pojmenovaný jako registrovaný typ a do tohoto souboru se zapíší všechny registrované třídy. Vše je patrné z příkladu 3.4.

```
soubor
    META-INF/services/org.foo.apimodule.MyInterface
obsahuje
    org.foo.mymodule.MyClass
```

Ukázka 3.4: Registrace pomocí soubrou v META-INF/services

Ad 3) Poslední ze tří metod není příliš používaná a je zde uvedena jen pro úplnost - příklad 3.5. Jde o registraci přidáním souboru do filesystému v layer souboru. Tento způsob se používá například při dynamické změně virtuálního souborového systému.

Ještě je zapotřebí vysvětlit k čemu je lookup a tedy i zmíněná registrace potřeba. Jedná se o seznam objektů jednotný pro celou aplikaci (přístup metodou `Lookup.getDefault()`

```
<filesystem>
    <folder name="Services">
        <folder name="Hidden">
            <file name="org-foo-jakekoliv-Jmeno.instance">
                <attr name="instanceClass"
                      stringvalue="org.foo.mymodule.MyClass"/>
                <attr name="instanceOf"
                      stringvalue="org.foo.apimodule.MyInterface"/>
            </file>
        </folder>
    </folder>
</filesystem>
```

Ukázka 3.5: Registrace pomocí layer souboru

naznačuje spřízněnost se singletony[13]), který umožňuje jak správu instancí, jež se mají vyskytovat pouze jednou v celé multi class loaderové aplikaci, tak vyhledávání tříd odvozených od zvolené bázové třídy či implementující určité rozhraní. Příklad takového použití je uveden v ukázce 3.6.

```
for (MyInterface objekt : Lookup.getDefault().lookupAll(MyInterface.class)) {
    objekt.printName();
}
```

Ukázka 3.6: Příklad použití defaultního lookupu

Tento příklad je typickou ukázkou v situaci, kdy je hlavní aplikace rozšiřitelná o zásuvné moduly, jako aplikace PS. Po přidání nového modulu s další třídou implementující určité rozhraní do takové aplikace je většinou požadováno, aby se instance této třídy mohla zařadit po bok dalších takových instancí. To je právě zajištěno registrací této třídy a následným využitím lookupu pro její vyhledání a použití.

3.3.4 Singleton ve vícemodulové aplikaci

Singleton je velice jednoduchý avšak mocný návrhový vzor. Jde o třídu, jejíž instance může být vytvořena jen jednou a je dosažitelná odkudkoliv (toto platí jen v aplikaci s jedním *class loaderem*). Použití singletonu je vhodné například pro uchovávání nastavení programu za jeho běhu. Příkladný singleton je vidět v java kódu 3.7.

Jak již bylo řečeno, problém nastává při užití více *class loaderů* v rámci jedné aplikace. Při zavolání metody `getInstance`, se totiž vytvoří nová instance pro každý *class loader*. Při používání NB platformy a více modulů v jedné aplikaci nastává právě tento problém. Implementace singletonů je možná využitím lookupu, ale musí se slevit z požadavku na `protected` konstruktor. Takový singleton potom vypadá tak, jak je zobrazeno v ukázce 3.8.

[13]Singleton je návrhový vzor, který zapříčiňuje, že se instance třídy vytvořené podle tohoto vzoru může v programu vyskytovat pouze jednou (toto není pravda v případě použití více class loaderů).

```
1  public class MySingleton {
2      private static MySingleton instance = null;
3
4      /* Konstruktor musi byt protected, aby objekt nemohl byt vytvoren mimo
           tridu. */
5      protected MySingleton() {}
6
7      /* Pristup k jedine existujici instanci pouzitim teto metody. */
8      public static MySingleton getInstance() {
9          if (instance == null) {
10             instance = new MySingleton();
11         }
12         return instance;
13     }
14 }
```

Ukázka 3.7: Návrhový vzor singleton

```
1  /* Je nutne tridu zaregistrovat napriklad pomoci anotace ServiceProvider. */
2  @ServiceProvider(service = LookupSingleton.class)
3  public class LookupSingleton {
4      /* Toto zustava nemenne. */
5      private static LookupSingleton instance = null;
6
7      /* Konstruktor musi byt kvuli lookupu public.
8         Upozornenim, ze se nema uzivat je anotace Deprecated*/
9      @Deprecated
10     public LookupSingleton() {}
11
12     /* Pristup k jedine existujici instanci pouzitim teto metody. */
13     public static LookupSingleton getInstance() {
14         if (instance == null) {
15             instance = Lookup.getDefault().lookup(LookupSingleton.class);
16         }
17         return instance;
18     }
19 }
```

Ukázka 3.8: Singleton vytvořený pomocí lookupu

3.4 Uživatelské rozhraní aplikace

Tato sekce se bude zabývat především návrhem komunikace aplikace s uživatelem. Převážně tedy půjde o vzhled a funkčnost grafického rozhraní.

3.4.1 Grafické rozhraní

Nedílnou částí implementace aplikace *Patterns Searching* je tvorba uživatelsky přívětivého grafického rozhraní. První návrh tohoto rozhraní se stylově podobal například aplikaci

Gimp[14], tedy celý program rozdělen do jednotlivých oken způsobem, který je naznačen na obrázku 3.9. Co se vlastní implementace GUI týče, mělo být využito swingu[15] a návrháře[16] v *NetBeans IDE*[17].

Obrázek 3.9: Naznačení prvního návrhu grafického rozhraní

Použití rozvržení aplikace do několika oken má své výhody a nevýhody. Hlavní předností je lepší modifikovatelnost celého GUI z pohledu uživatele, ten je v tomto modelu schopen optimalizovat velikost oken ke svému uspokojení, může je zavírat a otvírat podle potřeby. Toho všeho se dosáhne velice snadno z pohledu vývojového. Pokud by aplikace byla tvořena jedním oknem a její obsah byl rozdělen pouze do logických oddílů, implementace těchto možností by byla náročnější.

Na druhou stranu je rozdělení jednoho programu do více oken pro uživatele často nepřehledné, okna se různě překrývají a při nešikovné implementaci zabere každé okno další místo na systémovém hlavním panelu (panel zpravidla ve spodní části plochy s tlačítky odpovídajícími otevřeným oknům).

Tento model grafického rozhraní byl tedy zamítnut, také díky existenci vhodného řešení, a to využití platformy NetBeans. Aplikace využívající tuto platformu mají právě zmíněné výhody modelu rozděleného do oken, ale jeho nevýhody nesdílejí, protože je celý program obsažen v jednom okně.

[14] Gimp je aplikace pro úpravu obrázků.

[15] Swing je knihovna jazyka Java pro vytváření grafického rozhraní.

[16] Návrhář neboli designer je vývojářský nástroj, který umožňuje vytvářet grafické rozhraní přetahováním a upravováním jednotlivých komponent. Data o rozložení a vzhledu komponent se ukládají do externího souboru, z kterého je generován příslušný kód.

[17] NetBeans IDE je open-source vývojové prostředí primárně pro jazyk Java, obsahující nástroje pro návrh, ladění a profilování aplikací. Nyní podporuje vývoj v mnoha dalších jazycích, jako je c++, php, javascript, python, ruby a další.

3.4.2 Grafické rozhraní využívající NetBeans platformu

Návrh finálního GUI se tedy od původního liší hlavně ve spojení oken do jednoho hlavního rámce a také mírnou změnou uspořádání oken, kvůli maximálnímu využití možností platformy. Některé části se také schovaly pod takzvané wizardy[18], jejichž implementace je díky NB platformě velice příjemná. Nový návrh je naznačen na obrázku 3.10.

Obrázek 3.10: Naznačení návrhu grafického rozhraní využívajícího NB platformu

3.4.3 Prohlížeč obrázků

Prohlížeč obrázků pro aplikaci PS sice není stěžejní nástroj, co se týče vlastního vyhledávání, je však potřeba pro pohodlné prohlížení výsledků či vytváření vzorů. Jednodušší verze prohlížeče je také přímo základem funkce *Kontrola výsledků*.

Každý kvalitní nástroj pro zobrazování obrázků by měl obsahovat následující funkce, ke kterým jsou přidány také ty speciální pro tuto konkrétní aplikaci:

- zoomování
- posun obrázku
- označování oblastí
- zobrazování výsledků vyhledávání
- možnost odchytávat souřadnice kliknutí

[18] Wizard je grafická komponenta, která uživatele provede krok za krokem sekvencí dialogů. Používá se například při instalaci, nebo při spuštění procesu potřebujícího rozsáhlejší nastavení.

Důležitou vlastností prohlížeče je také dostatečná rychlost reakcí i při zobrazování velkých obrázků. Této rychlosti se dosáhne použitím dlaždicování popsaného v 3.2.2. Jak je ve zmíněné kapitole řečeno, při zobrazení celého obrázku se výhoda dlaždicování ztrácí. Vyřešení toho nedostatku by bylo vhodné v rámci diplomové práce.

Nástroje k ovládání obrázků budou umístěny na toolbaru a některé z nich je možné využívat klasickými postupy, jako je např. *CTRL + scroll* pro zoomování. Označování oblastí je úzce spjato s vytváření nového vzoru, a tak bude na nástrojové liště také přítomno tlačítko pro vyvolání dialogu pro vytvoření vzoru z označené části zobrazovaného obrázku.

Zobrazování výsledků bude provedeno vykreslením daného obrázku a na místech s nalezenými vzory se objeví průhledné obdélníky ukazující oblast vzoru. Mezi výsledky vyhledávání různých vzorů bude možné přepínat pomocí rolovací nabídky umístěné na nástrojové liště.

3.4.3.1 Kontrola výsledků

Kontrola výsledků využívá ve svém dialogovém okně upravený prohlížeč obrázků. Ten se od původního liší v několika vlastnostech:

- zobrazuje pouze část obrázku bez možnosti posunu na další část přetažením
- kliknutím prostředním tlačítkem myši se zobrazí následující část obrázku
- kliknutím na nalezený vzor se vzor odstraní
- kliknutím na obrázek mimo vzor se oblast přidá jako vzor

Tento upravený prohlížeč má také svůj vlastní panel nástrojů obsahující tlačítka pro používání uvedených funkcí.

3.4.4 Manažery

Manažery jsou okna, většinou menšího rázu, která zprostředkují uživateli možnost spravovat jistá data.

3.4.4.1 Manažer skupin vzorů (Patterns Groups Manager)

Funkcí komponenty *Patterns Groups Manager* je správa skupin vzorů, které jsou na pevném disku počítače reprezentovány složkami na určitém umístění. Tento manažer by měl obsahovat seznam existujících skupin a tlačítka pro jejich přejmenování, přidávání či mazání.

3.4.4.2 Manažer vzorů (Patterns Manager)

Okno zvané *Patterns Manager* obsahuje komponenty pro úpravu jedné konkrétní skupiny vzorů. Musí tedy obsahovat nabídku se seznamem dostupných skupin. Náplní skupiny jsou především její vlastní vzory, ty musí být k nahlédnutí spolu s možností nějaký vzor pro výpočet nepoužívat. Správu vzorů poté budou umožňovat tlačítka k editaci a mazání (funkce pro přidání je obsažena v nástrojové liště prohlížeče obrázků). V neposlední řadě musí být přítomno vyobrazení kombinovaného vzoru, jako reprezentace celé skupiny.

Tlačítko pro editaci vyvolá dialog pro úpravu vzorů, shodný s dialogem pro vytváření. Tento dialog musí umožňovat určení skupiny, do které vzor náleží, mírné upravení jeho velikosti a v neposlední řadě nástroj pro určení referenčního bodu vzoru.

3.4.4.3 Okno transformace (Transformation Window)

Toto okno jmenující se *Transformation Window* je vždy spjato s otevřeným obrázkem (musí existovat také jeho dialogová podoba, ke správě transformace bez otevřeného obrázku). Okno musí obsahovat seznam identických bodů pro transformaci a také nástroj na jejich úpravu a přidávání. Vhodné je odchytávání souřadnic kliknutí myší na daný obrázek a nabídnout uživateli tyto souřadnice jako bod ve výchozím souřadnicovém systému. K tomuto bodu stačí dopsat souřadnice v požadovaném systému.

3.4.5 Wizardy

Wizard je grafická komponenta, která uživatele provede krok za krokem sekvencí dialogů. Používá se například při instalaci, nebo při spuštění procesu potřebujícího rozsáhlejší nastavení. Tyto wizardy jsou v aplikaci PS využity ke spouštění hlavních akcí.

3.4.5.1 Vyhledávání (Run)

Ke spuštění vlastního vyhledávání je potřeba tří vstupních údajů (obrázky pro vyhledávání, vyhledávací vzory a algoritmy), tudíž je wizard *Run* složen ze tří panelů.

Prvním z nich je panel pro určení souborů obrázků, které se mají prohledávat. Tento panel musí obsahovat seznam těchto souborů a především tlačítka pro jejich přidávání a ubírání. Vhodným vylepšením je také možnost přidání celého adresáře, z něhož se použijí všechny soubory rozeznaného typu obrázek. Vzhledem k tomu, že aplikace pracuje s několika podobami jednoho obrázku (v tomto případě jsou důležité podoby *threshold* a *original*), musí zde existovat možnost určení požadované podoby.

Panel pro určení vyhledávaných vzorů bude obsahovat seznam skupin vzorů s náhledem kombinovaného vzoru a také tlačítka pro jeho správu. Tlačítka musí ovládat jak přidávání a ubírání, tak změnu pořadí.

Posledním panelem v průvodci spuštění je ten s nastavením algoritmů počítajících podobnost. Tento panel také obsahuje příslušný seznam s tlačítky, ale také se zde musí vyskytovat panel pro nastavení každého z použitých algoritmů.

S funkcí *Vyhledávání* souvisí také ***Rychlý start*** neboli *Quick Run*. Tato funkce bude sloužit k okamžitému spuštění vyhledávání vzoru určeného *Manažerem vzorů* na právě otevřeném obrázku za použití defaultního nastavení algoritmů.

3.4.5.2 Kontrola výsledků (Check)

Wizard provádějící uživatele nastavením *Kontroly vyhledávání* obsahuje pouze dva panely.

První z nich uživateli nabízí možnost vybrat výsledek vyhledávání ke kontrole a také velikost zobrazení obrázku. Výběr výsledku probíhá nejprve vybráním obrázku na kterém bylo vyhledávání provedeno, a poté určením vzoru, který se vyhledával.

Druhý a také poslední panel obsahuje obecné nastavení kontroly.

3.4.5.3 Prahování (Thresholding)

Průvodce zvaný *Thresholding* je spíše dialogovým oknem, jelikož obsahuje jediný panel. Ten zobrazuje seznam obrázků pro aplikaci prahování, umožňuje přidávání a mazání obrázků ze seznamu a také obsahuje políčko pro určení samotného prahu.

3.4.6 Projekty

Podpora projektů není pro tuto aplikaci nepostradatelná a tudíž bude implementována jen velmi jednoduše. Projekt bude tedy uchovávat informace o otevřených souborech (obrázcích) a veškeré výsledky vyhledávání. Krom toho je projekt definován vlastním jménem a samozřejmě ho bude možné ukládat a znovu otevírat.

3.4.7 Nastavení

Díky využití NB platformy a konkrétně s pomocí třídy `NBPreferences` není uchovávání uživatelského nastavení nijak složité. Veškeré nastavení uchovávané tímto způsobem je ukládáno do userdiru v souborech formátu .properties.

Ke správě nastavení je však kromě ukládání také třeba snadná možnost úpravy z pohledu uživatele. V první verzi aplikace PS bude tedy možné nastavovat pouze defaultní algoritmy určující podobnost dvou obrazů, v panelu *Algortihms* okna *Options*.

Další možnosti nastavení zůstávají ze samotné NB platformy a je jimi především správa klávesových zkratek a chování oken.

3.4.8 Rozšiřitelnost aplikace

U moderních aplikací je běžně možné rozšíření o nové funkce pomocí pluginů. *Patterns Searching* je také vhodná pro takovéto rozšíření hlavně v oblasti exportu souřadnic do různých formátů a také vlastních algoritmů určujících podobnost.

Pro tyto funkce je tedy třeba důkladně navrhnout veřejné API, konkrétně rozhraní, které budou nové třídy implementovat, aby mohly být rozpoznány lookupem (kapitola 3.3.3).

Export souřadnic si vystačí s jedním rozhraním, jehož podoba je ukázána v kódu 3.9. Třída implementující toto rozhraní musí být definována příponou souboru, kterou bude vracet ve funkci `getFileExtension`, a také musí implementovat funkci `export`, jež provede vlastní vytvoření souboru se souřadnicemi.

```
1 public interface Exporter {
2     public String getFileExtension();
3     public boolean export(File file, Result points, boolean transform);
4 }
```

Ukázka 3.9: Rozhraní Exporter

Algoritmus rozpoznávání vzoru, reprezentovaný abstraktní třídou, je nutné rozšířit také o jednu grafickou komponentu sloužící pro uživatelské nastavení tohoto algoritmu. Návrh

těchto tříd je možné vidět v ukázce 3.10. V třídě `Algorithm` jsou důležité především metody `getAlgorithm`, které vytvářejí ze zadaného obrázku masku s určitým postupem určování podobnosti - vlastní algoritmus. Třída `AlgorithmOptionsPanel` má kromě komunikace s uživatelem za úkol nastavování komponent podle zadaného nastavení (metoda `setupComponents`) a naopak vytváření nastavení z dat v grafických komponentách (metoda `setupOptions`).

```
1  public abstract class Algorithm implements Comparable<Algorithm>{
2      protected AlgorithmOptionsPanel optionsPanel;
3
4      abstract public String getOptionsInfo();
5      abstract public Class getAlgorithmClass();
6      abstract public SimilarityMask getAlgorithm(BufferedImage image);
7      abstract public SimilarityMask getAlgorithm(int[][][] image);
8      abstract public void saveAlgorithm(Preferences pref);
9  }
10
11 public abstract class AlgorithmOptionsPanel extends JPanel{
12     protected AlgorithmOptions options;
13
14     protected abstract void setupOptions();
15     protected abstract void setupComponents();
16 }
```

Ukázka 3.10: Abstraktní třídy Algorithm a AlgorithmOptionsPanel

Kapitola 4

Realizace

Tato kapitola se bude zabývat vlastní implementací aplikace *Patterns Searching* (PS). Budou zmíněny konkrétní třídy a jejich funkce. S přihlédnutím k rozsahu práce a množství tříd se nebudou všechny dopodrobna rozebírat. Blíže popsané budou pouze ty stěžejní či problémové a v příloze A bude uveden seznamem všech tříd se stručným popisem jejich funkce.

Obecným problémem celé realizace bylo užívání API NB platformy, do kterého se pronikalo pomalu a každá novinka se musela dlouho hledat v dokumentaci. Dalším úskalím bylo užívání knihovny JAI, jejíž potenciál nebyl ani zdaleka využit. Hloubkové porozumění této knihovny by zabralo značnou dobu, jež by spolu s časem stráveným s dalšími problémovými i běžnými částmi realizace překročila rozsah bakalářské práce.

Vlastní implementace byla kompletně provedena ve vývojovém prostředí NetBeans IDE (verze 6.7, 6.8 a 6.9) a bylo užito NetBeans Platform 6.8.

4.1 Knihovna Utils

Knihovna `Utils` je tvořena obecnými třídami a funkcemi, které je možné používat v různých projektech. Je tedy vytvářena i pro osobní potřebu a obsahuje tudíž také třídy, které v aplikaci PS nejsou použity.

Pro tento projekt je z knihovny nepostradatelný balíček[1] `cz.crishpean.utils.image`. Z něj jsou stěžejní především třídy `Colors`, `ImageUtils` a hierarchie tříd odvozených od `Mask`.

Třída `Colors` slouží k práci s barvami, jak již její název napovídá. Obsahuje pouze statické metody pro převody mezi dvěmi reprezentacemi barev, a to jednou celočíselnou hodnotou a trojicí (či čtveřicí) hodnot pro jednotlivé barevné složky. S těmito převody také souvisí vytváření šedotónových obrázků z barevných, na což jsou v této třídě také dvě metody. Ty jsou sice funkčně totožné, ale jedna pracuje s klasickým `BufferedImage` a druhá s objekty knihovny JAI `RenderedOp`. Takovéto rozdělení na varianty se vyskytuje v celé aplikaci, jelikož se v ní pracuje s různými podobami obrázků. Dvě z nich již byly zmíněny a třetí je běžné dvourozměrné pole celočíselných hodnot.

[1] Balíček neboli package je název pro strukturu sjednocující třídy podobného významu, nebo vztahující se k podobnému tématu. Na pevném disku jsou balíčky reprezentovány adresáři.

`ImageUtils` obsahuje také statické metody a krom toho i několik statických konstant. Metody této třídy vytvářejí různé možnosti zvětšení obrázků popsané v kapitole 3.1.3. Vzhledem k časté nutnosti konverze obrázku do šedých tonů a následného vytvoření dvourozměrného pole obsahuje tato třída také metody pro spojení těchto dvou kroků a vytváří tak rovnou šedotónové pole.

Třída `Mask` je jednou z nejzákladnějších tříd v celé aplikaci. Je to programová reprezentace masky nejen pro vyhledávání, ale také pro jiné zpracování obrazu. Nedílnou součástí této třídy je `MaskMatrix`, což je třída reprezentující vlastní data masky a obsahuje metody pro jejich zpracování a provádení výpočtů.

Nejdůležitejším potomkem třídy `Mask` je `SimilarityMask`, což je abstraktní třída, jejíž potomci jsou již implementací algoritmů. Její struktura je uvedena níže (ukázka 4.1). Zahrnuje čtyři abstraktní metody, jejichž přepsáním se vytvoří třída schopná určovat míru podobnosti dvou vzorů a je možné ji užívat v celé aplikaci, protože se všude pracuje právě s třídou `SimilarityMask`.

Metoda `countSimilarity` má za úkol vypočítat koeficient podobnosti s oblastí obrázku předaného parametrem spolu se souřadnicemi určujícími tuto oblast. Tyto souřadnice určují místo kam se přiloží střed masky. Návratovou hodnotou je instance třídy `PointValue`, což je `Point` z jazyka Java rozšířený o jednu hodnotou typu `double`, která je ve většině případů užití v aplikaci koeficientem podobnosti. Další tři abstraktní metody pracují s hodnotou koeficientu a určují její význam.

```
1  public abstract class SimilarityMask extends GreyScaledMask{
2      public SimilarityMask(BufferedImage mask){
3          super(mask);
4      }
5      public SimilarityMask(int[][][] mask){
6          super(mask);
7      }
8      public SimilarityMask(short[][][] mask){
9          super(mask);
10     }
11     public abstract PointValue countSimilarity(short[][][] image, int w, int h);
12     public abstract boolean isSimilarEnough(double coef);
13     public abstract boolean isBetterThan(double coef1, double coef2);
14     public abstract boolean isUnsimilarEnough(double coef);
15 }
```

Ukázka 4.1: Třída SimilarityMask

Odvozením od `SimilarityMask` vznikla třída `CorrelationMask`, která obsahuje naprogramování vzorce 3.1 a je základem modulu *Correlation Algorithm*, jenž je základním algoritmem pro výpočet podobnosti distribuovaný s aplikací.

4.2 Vykreslování obrázků

O vykreslování obrázků se v aplikaci starají třídy pojmenované canvas neboli plátno. Obecně jsou odvozené od komponenty `JPanel` z grafické Java knihovny Swing, jíž přepisují

metodu `paint`. Jako většina dalších grafických komponent jsou i tato plátna náležitostí modulu *Patterns Searching*.

Bázovou třídou, od níž jsou odvozena všechna ostatní plátna, je třída `TiledImageCanvas`. Ta vykresluje dlaždice, jež mají být zobrazeny na obrazovce, v metodě `paint`, ale také vykresluje znázornění označené oblasti a reaguje na zoomování. Toto plátno je zcela využitelné jako grafická komponenta, ale jak již bylo řečeno, slouží také jako základ k odvození dalších tříd.

Potomkem třídy `TiledImageCanvas` je `ResultImageCanvas`, která navíc zobrazuje výsledky vyhledávání jako obdélníky reprezentující nalezené vzory. Vykreslování těchto oblastí musí správně reagovat na zoomování a posun zobrazované části obrázku, proto dochází při každém vykreslení k jednoduché transformaci souřadnic změnou měřítka a posunem.

Poslední třídou v této hierarchii je `CheckImageCanvas` sloužící pro zobrazování obrázku při kontrole výsledků. Jejím jediným rozdílem od rodičovské `ResultImageCanvas` je vykreslování obdélníku pod kurzorem myší. Ten pomáhá uživateli při přidávání nevyhledaných vzorů.

K ovládání pláten slouží komponenty zvané editory, které zprostředkovávají komunikaci s uživatelem a obrázek vykreslí pomocí plátna, jež obsahují. Editorem pro práci s `TiledImageCanvas` a pláten odvozených je `TiledImageEditor`. Pro obrázky u nichž není předpokládaná značná velikost se používá editor `ImageEditor`, který místo dlaždicového plátna obsahuje jednodušší plátno bez schopnosti práce s dlaždicemi zvané `ImageCanvas`. Tyto třídy jsou jinak shodné s jejich "tiled" protějšky.

`ImageCanvas` a `ImageEditor` jsou použity jen jako základ pro odvození `ReferencedImageCanvas` a `ReferencedImageEditor`. Ty jsou použity například při vytváření nového vzoru a jejich schopnost tkví v zobrazování referenčního bodu zobrazovaného obrázku. Bod se vykreslí jako červený křížek. Důležitou vlastností `ReferencedImageEditoru` je odchytávání událostí myši a tedy určování pozice referenčního bodu pomocí klikání na plátno.

Speciálním případem plátna je takzvaný renderer, který se používá při vykreslování obrázků do buněk tabulky. Třída `ImageRenderer` je jednoduchý `JPanel`, který vykresluje obrázek pomocí zadané, neměnící se transformace. Jeho potomek `PatternImageRenderer` se liší pouze v práci se vzory, čímž se spolu s obrázkem vykreslí také jeho referenční bod.

Pro uzpůsobení vykreslování objektů v tabulkách, jako je zmíněné zobrazení obrázku v buňce, je třeba vytvořit třídu odvozenou od `TableCellRenderer` a přepsat jí jednu metodu, jak je vidět na konkrétním použití níže v kódu 4.2, kde je vidět také užití třídy `PatternImageRenderer`. Objekt typu `ImageTableCellRenderer` je poté nutné nastavit požadované tabulce následujícím způsobem:

```
myTable.setDefaultRenderer(PatternImage.class, new ImageTableCellRenderer());
```

což znamená, že se pro všechny objekty typu `PatternImage` vložené do tabulky použije k vykreslení třída `ImageTableCellRenderer`.

4.3 Správa obrázků

Vzhledem k častému používání obrázků v celé aplikaci je pro tuto záležitost vytvořena jedna třída, konkrétněji singleton, nazvaná `ImagesStore`. Funkce tohoto singletonu jsou následující:

```
1  public class ImageTableCellRenderer implements TableCellRenderer{
2      @Override
3      public Component getTableCellRendererComponent(JTable table,
4                                                     Object value,
5                                                     boolean isSelected,
6                                                     boolean hasFocus,
7                                                     int row,
8                                                     int column) {
9          PatternImage pattern = (PatternImage)value;
10         PatternImageRenderer component = new PatternImageRenderer(pattern,
11                 table.getColumnModel().getColumn(column).getWidth(),
12                 table.getRowHeight() - table.getRowMargin(),
13                 isSelected);
14
15         return component;
16     }
17 }
```

Ukázka 4.2: Třída ImageTableCellRenderer

- načítání obrázků (metoda `getImage` vrátí načtený obrázek, pokud není tak ho načte a vrátí)
- ukládání obrázků na disku (`saveImage`)
- uchovávání načtených obrázků v mapě[2] pro rychlejší přístup (obrázky se nemusí načítat, když už jsou načtené)
- načítání bodů pro transformaci určitého obrázku (metoda `getTransformation` vrátí načtenou transformaci, pokud není tak ji načte a vrátí)
- přiřazení transformace k obrázku (`putTransformation`)
- uchování načtených transformací

V neposlední řadě tato třída spravuje varianty jednoho obrázku, jako je *threshold* a *greyscaled*. Je tedy možné za použití stejných metod jako při práci s originálními obrázky získávat či ukládat tyto varianty, pomocí konstant `ImagesStore.THRESHOLD_IMAGE_SIGN` a `ImagesStore.GREY_IMAGE_SIGN` určujících tyto varianty.

4.4 Správa výsledků

Třídy pro správu výsledků jsou umístěny ve veřejném modulu aplikace PS *Patterns Searching API*, jelikož je potřeba, aby byly výsledky přístupné i z jiných modulů a především samotné třídy musí být použitelné i mimo hlavní modul aplikace.

Třída uchovávající souřadnice vyhledaných bodů se nazývá `Result`. Její stěžejní částí je mapa uchovávající pro každý vzor seznam bodů:

[2]Mapa je datová struktura využívající k indexaci uložených dat objekty libovolného typu.

```
private Map<PatternsGroup, List<GeoPoint>> results;
```

Body jsou uchovány jako objekty typu `GeoPoint`, což je třída odvozená od `Point.Double` a je jí přidána z-ová souřadnice, identifikátor souřadnicového systému a také vzor, který je bodem reprezentován. V aplikaci se ještě souřadnicové systémy v podstatě nepoužívají, `GeoPoint` je na to však připraven.

`Result` také obsahuje člen typu `Transformation` udávajícího transformaci, jež má být na body aplikována.

Třídou, která tyto jednotlivé výsledky sdružuje, je `Results`. Ta obsahuje mapu přiřazující každému souboru (obrázek je identifikován názvem souboru na disku) objekt typu `Result`:

```
private Map<File, Result> results;
```

Tato třída je singletonem vytvořeným postupem popsaným v 3.3.4 a kromě metod pro získávání konkrétních seznamů souřadnic pro obrázek, či obrázek a vzor, obsahuje také metody pro ukládání a načítání seznamů z XML formátu, což je používané pro správu projektů (4.6).

Výsledky je také možno vyexportovat do souboru formátu, který je k dispozici. S aplikací je dodáván pouze modul *CSV Export* umožňující export souřadnic do formátu csv (Coma Separated Values). Tento modul musí obsahovat třídy implementující dané rozhraní z modulu *Patterns Searching API*:

```
public class CsvExporter implements Exporter;
```

Také je nutné, aby měl soubor *layer.xml* a v něm definovaný způsob jak export vyvolat pomocí GUI. Modul *CSV Export* toto řeší přidáním tlačítka na panel nástrojů a také vytvořením položky v menu *File -> Export -> To CSV*. Akce vyvolaná těmito tlačítky je tvořena třídou `CsvExportAction`, která volá statickou metodu `export` třídy `CsvExportAction` z veřejného API (modul *Patterns Searching API*).

4.5 Správa vzorů

Hierarchie správy vzorů se skládá ze tří úrovní sestávajících ze tříd `PatternImage`, `PatternsGroup` a `PatternsManager`.

První ze tříd (`PatternImage`) reprezentuje vlastní obrázek typu `BufferedImage` a přidružuje k němu ještě další informace, jako je poloha referenční bodu a údaj, zda se má tento konkrétní obrázek využít pro vytvoření kombinovaného vzoru (vzor pro vyhledávání). V neposlední řadě obsahuje metody pro práci se vzorem a také pro ukládání a načítání zmíněných informací do souborů typu .properties.

Třída sdružující různé obrázky typu `PatternImage` jednoho vzoru se jmenuje `PatternsGroup` a krom jiných metod vlastní hlavně funkci `getCombinedPattern` pro výpočet kombinovaného vzoru a `loadPatterns` načítající jednotlivé vzory. Reprezentací této třídy na disku je adresář na konkrétním umístění.

Poslední ze zmíněných tříd je singleton `PatternsManager`, jenž obsahuje seznam všech skupin vzorů (`PatternsGroup`), metody pro jejich načítání a také úpravu skupin i jednotlivých vzorů.

4.6 Projekty

Programová reprezentace projektu je běžný singleton `Project`. Obsahuje především metody pro tvorbu XML dokumentu a následně jeho načtení. Pro práci s XML byla použita knihovna *JDOM*. Do projektového souboru (koncovka .psp (Patterns Searching Project)) se ukládá název projektu, právě otevřené obrázky (v aktuální verzi je možné mít otevřen pouze jeden obrázek) a výsledky vyhledávání v podobě souřadnic bodů.

4.7 Vyhledávání

Spouštění vyhledávání se odehrává v metodě třídy `SearchPatterns`. Jde o singleton klasického typu, jelikož je používán pouze v modulu *Patterns Searching*, tedy v hlavním modulu aplikace.

Jedná se o celkem jednoduchou třídu hlavně se *settery*. Ty jsou důležité pro nastavení vstupních dat pro vyhledávání (obrázky, vzory, algoritmy). Dále jsou stěžejní dvě metody provádějící vlastní vyhledávání. Metoda `run` je spouštěč vlákna, které provede pro všechny obrázky vyhledání všech vzorů pomocí druhé metody `findPatterns`, která pohybuje maskou (vzorem) po daném obrázku a ukládá souřadnice nalezených vzorů.

K této třídě je bohužel nutné dodat, že její implementace není dovedena k dokonalosti, jelikož při spuštění vyhledávání na více velikých obrázcích dochází k pádu aplikace kvůli nedostatku paměti. Je předpokládáno, že se tento nedostatek v budoucnu odstraní.

S vyhledáváním také souvisí vlastní algoritmus určení podobnosti dvou vzorů. Tyto algoritmy jsou k aplikaci přidány formou modulů, aby bylo možné pohodlně rozšiřovat jejich počet a kvalitu. Defaultní modul distribuovaný s aplikací se nazývá *Correlation Algorithm* a využívá výpočtu korelačního koeficentu podle vzorce 3.1. Modul využívá třídu `CorrelationMask` z knihovny *Utils* a tvoří ho pouze dvě třídy - implementace abstraktních tříd:

```
public class CorrelationAlgorithm extends Algorithm;
public class CorrelationAlgorithmOptionsPanel
             extends AlgorithmOptionsPanel
             implements ChangeListener;
```

4.8 Tvorba grafických komponent

Pro tvorbu veškerých grafických komponent byl využit návrhář (designer) prostředí NetBeans IDE. Při vytváření grafických tříd přímo souvisejících s NB platformou se užívalo také průvodců NetBeans IDE a jednalo se tedy o velmi rychlou záležitost. NetBeans IDE se také stará o generování některých souborů, jež jsou pro aplikaci důležité, avšak programátor se jimi vůbec nemusí zabývat.

Při vytváření vlastní grafiky nevznikl žádný problém a používání NetBeans IDE není nutné v tomto směru popisovat.

4.9 Instalátor

Kvůli tvorbě instalátoru muselo být použito vývojové prostředí NetBeans IDE 6.9, a to vývojová verze, jelikož stabilní ještě nebyla k dispozici. Jedna z novinek v této verzi je právě jednoduché generování instalátoru, stejného jako používá vlastní NetBeans IDE. Tvorba instalátorů (pro operační systémy Windows, Linux, Solaris a Mac OS) byla opravdu velmi snadná a osobně tento nový nástroj velmi oceňuji a doporučuji.

Jediným problémem bylo nastavení parametrů v budoucnu nainstalované aplikace. Program PS se musí spouštět s parametrem -Xmx1024m, kvůli značným nárokům na paměť. Možné řešení by bylo uvést v instalační příručce postup, jak se má nainstalovaný software nastavit, což není příliš vhodné kvůli pohodlí uživatele.

Nakonec se podařilo získat radu přímo od vývojáře Dmitryho Lipina společnosti Sun Microsystems, Inc. (společnost vyvíjející NetBeans IDE i platformu), který se zabývá produktem NetBeans Installer, ne němž jsou založeny vygenerované instalátory. Řešení bylo vskutku snadné, ale jelikož je tento nástroj novinka, která svým způsobem ještě ani nevyšla, je málo zdokumentovaný. Řešení tedy bylo následující:

1. zkopírovat soubor *adresář_kde_jsou_nainstalovane_netbeans/harness/etc/app.conf* do složky projektu (suity) a pojmenovat ho například *myapp.conf*
2. v tomto souboru upravit položku *default_options* podle vlastních potřeb (je také možné změnit například userdir)
3. do souboru *adresar_projektu/nbproject/project.properties* přidat řádek *app.conf=myapp.conf*
4. provést clean projektu a vytvořit instalátor

Kapitola 5

Testování

Náplň této kapitoly se omezí pouze na testování úspěšnosti vyhledávacího algoritmu. Kompletní testování by mělo zahrnovat také vytvoření a výsledky jednotkových[1] testů a důkladné vyzkoušení GUI a veškerých jeho funkcí. To však pro udržení jakéhosi rozsahu práce nebylo provedeno.

Ze stejného důvodu také nebylo provedeno testování přesnosti určení polohy vzoru, které bylo původně zamýšleno. Toto by se testovalo pomocí některého GIS softwaru a porovnávaly by se odchylky od ručně vektorizovaných map (otázkou zůstává, zda by šlo o testování ruční či automatické vektorizace).

Testy byly provedeny na konfiguraci *Intel Core2 Duo T7500 2.20 GHz, 2 GB RAM, Ubuntu 9.04 32-bit*.

5.1 Úspěšnost vyhledávání

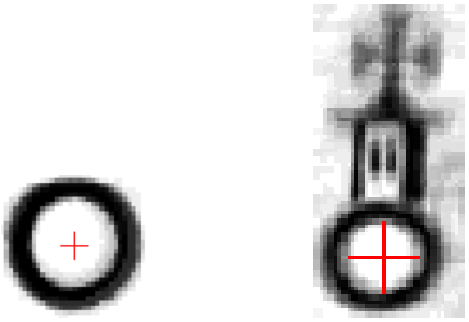
Testování vyhledávání a hlavně určování podobnosti dvou rastrů bylo provedeno s několika variantami nastavení na jednom mapovém listě Müllerovy mapy reprezentovaným souborem *I-1-137-19.jpg*. Tento obrázek byl ještě před vytvářením vzorů převeden prahováním (práh = 128) na černobílý.

Pro vyhledávání byly zvoleny dva vzory vyskytující se na mapě často (obr. 5.1). Na počátku testování byl určen počet těchto vzorů na dané mapě s využitím funkce *Kontrola výsledků* aplikace PS. V tomto případě se jako vzor považovala také část jiného vzoru, pokud v něm byl ten první obsažen, např. vzor na obrázku 5.1a je také součástí vzoru na obrázku 5.1b a tudíž je počet výskytů vzoru B započten do počtu vzoru A.

Vzhledem k počtu výskytů vzoru B v jednom mapovém listě, bylo využito také jiného mapového listu k vytváření vzoru (není žádoucí, aby se testovalo poznávání vzoru, který je tvořen téměř polovinou všech jeho výskytů na mapě).

Veškeré výsledky vyhledávání jsou patrné z tabulek 5.1 a 5.2. Tabulky jsou rozděleny do tří částí podle počtu obrázků tvořících vlastní vzor neboli kombinovaný vzor (vytvořený

[1] Jednotkové testy neboli unit tests popisuje následující definice podle [12]: "Pod pojem unit testing se zahrnují nástroje, metodika a činnost, jejímž cílem je ověřování správné funkčnosti dílčích částí neboli jednotek zdrojového textu."



(a) Vzor A (b) Vzor B

Obrázek 5.1: Vzory zvolené k testování

zprůměrováním zmíněných obrázků). To bylo provedeno ke zjištění chování při zvyšování počtu obrázků.

První sloupec v tabulkách nazvaný *nastavení* obsahuje dvě hodnoty určující chování algoritmu pro počítání míry podobnosti. Tyto hodnoty se nacházejí ve formátu a/b, kde a je hraniční korelační koeficient a b je 1, pokud byl a 0, pokud nebyl použit *předvýpočet řezu*.

Počty chyb ve vyhledávání byly určeny pomocí funkce *Kontrola výsledků*. Po doběhnutí vyhledávacího algoritmu byla spuštěna tato kontrola a byly z výsledků odstraněny špatně určené vzory. Následně se určily požadované rozdíly mezi skutečným počtem vzorů, nalezeným počtem a tím opraveným o chybně určené oblasti.

Z hodnot v tabulkách je možné zjistit následující závěry:

- s počtem obrázků tvořící vzor klesá počet nerozpoznaných vzorů
- s počtem obrázků tvořící vzor roste počet chybně určených vzorů
- využití *předvýpočtu řezu* řádově zvyšuje rychlost
- využití *předvýpočtu řezu* snižuje počet rozpoznaných vzorů
- využití *předvýpočtu řezu* snižuje počet chybně určených vzorů
- velikost limitního korelačního koeficientu snižuje počet chybně určených vzorů ale také zvyšuje počet nerozpoznaných vzorů

Ukázky oblastí, které byly chybně rozpoznány jako vzor A, jsou na obrázku 5.2. Chyby ve výsledcích vyhledávání vzoru B byly vždy oblasti se vzorem A, nad kterým se nacházel nějaký další objekt, jako například písmeno nebo řeka. Potlačení takovýchto chyb ve vyhledávání by bylo možné zavedením takzvaných *antivzorů* - každý nalezený vzor by se porovnal s tímto antivzorem, pokud by byl korelační koeficient vyšší než při porovnání s vyhledávaným vzorem, byl by zamítnut. Tato funkcionalita by vyhledávání jistě zpomalila a určitě by všechny chyby neodstranila, je to však možnost dalšího zlepšování úspěšnosti aplikace.

Bylo by také do budoucna vhodné, doplnit aplikaci o nástroj, který by uživateli pomohl s určením vhodného limitu korelačního koeficientu (či jiného koeficientu určujícího podobnost). Z testování jasně vyplývá značná odlišnost dosažených výsledků při změně limitu o 0.1, a proto by se hodilo nějaké přesnější určení.

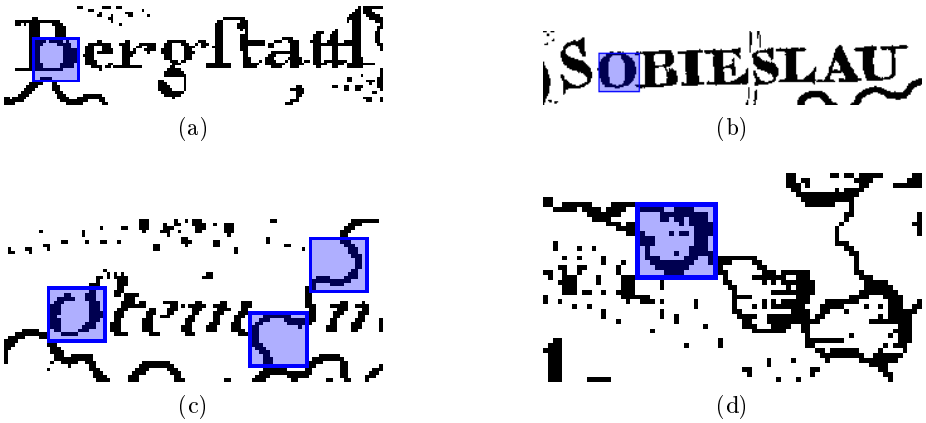
Celkový počet výskytů vzoru A na mapovém listě: 961

Vzor A - počet obrázků: 1						
Nastavení	Čas [s]	Správně určeno [%]	Nalezených vzorů	Nenalezeno [ks] / [%]	Chybně určeno [ks] / [%]	Celkem chyb [ks] / [%]
0.5 / 1	20	89	858	109 / 11	6 / 1	115 / 12
0.6 / 1	17	85	822	140 / 15	1 / 0	141 / 15
0.5 / 0	354	96	954	42 / 4	35 / 4	77 / 8
0.6 / 0	352	88	850	112 / 12	1 / 0	113 / 12

Vzor A - počet obrázků: 10						
Nastavení	Čas [s]	Správně určeno [%]	Nalezených vzorů	Nenalezeno [ks] / [%]	Chybně určeno [ks] / [%]	Celkem chyb [ks] / [%]
0.5 / 1	18	80	793	194 / 20	26 / 3	220 / 23
0.6 / 1	18	67	649	318 / 33	6 / 1	324 / 34
0.5 / 0	376	89	1068	104 / 11	211 / 22	315 / 33
0.6 / 0	376	72	710	271 / 28	20 / 2	291 / 30

Vzor A - počet obrázků: 47						
Nastavení	Čas [s]	Správně určeno [%]	Nalezených vzorů	Nenalezeno [ks] / [%]	Chybně určeno [ks] / [%]	Celkem chyb [ks] / [%]
0.5 / 1	20	94	936	53 / 6	28 / 3	81 / 8
0.6 / 1	19	92	887	78 / 8	4 / 0	82 / 9
0.5 / 0	417	97	1061	23 / 2	123 / 13	146 / 15
0.6 / 0	417	93	901	68 / 7	8 / 1	76 / 8

Tabulka 5.1: Výsledky testování pro vzor A



(a) (b) (c) (d)

Obrázek 5.2: Chybně určené oblasti jako vzor A

Celkový počet výskytů vzoru B na mapovém listě: 78

Vzor B - počet obrázků: 1						
Nastavení	Čas [s]	Správně určeno [%]	Nalezených vzorů	Nenalezeno [ks] / [%]	Chybně určeno [ks] / [%]	Celkem chyb [ks] / [%]
0.5 / 1	18	36	28	50 / 64	0 / 0	50 / 64
0.6 / 1	18	9	7	71 / 91	0 / 0	71 / 91
0.5 / 0	849	44	34	44 / 59	0 / 0	44 / 59
0.6 / 0	852	10	8	70 / 90	0 / 0	70 / 90

Vzor B - počet obrázků: 10						
Nastavení	Čas [s]	Správně určeno [%]	Nalezených vzorů	Nenalezeno [ks] / [%]	Chybně určeno [ks] / [%]	Celkem chyb [ks] / [%]
0.5 / 1	19	68	53	25 / 32	0 / 0	53 / 25
0.6 / 1	19	58	45	33 / 42	0 / 0	45 / 33
0.5 / 0	904	97	99	2 / 3	15 / 19	17 / 21
0.6 / 0	908	78	61	17 / 22	0 / 0	17 / 22

Vzor B - počet obrázků: 37						
Nastavení	Čas [s]	Správně určeno [%]	Nalezených vzorů	Nenalezeno [ks] / [%]	Chybně určeno [ks] / [%]	Celkem chyb [ks] / [%]
0.5 / 1	21	85	66	12 / 15	0 / 0	12 / 15
0.6 / 1	21	60	47	31 / 40	0 / 0	31 / 40
0.5 / 0	913	100	123	0 / 0	45 / 58	45 / 58
0.6 / 0	911	83	65	13 / 17	0 / 0	13 / 17

Tabulka 5.2: Výsledky testování pro vzor B

5.2 Nedostatky zjištěné při testování

Při testování úspěšnosti byla aplikace značně vytížena a byla tedy testována i její další funkcionalita. Byly objeveny některé zásadní nedostatky, které bude nutné co nejdříve opravit, jelikož některé z nich hraničí s chybou.

Zde bude uveden jejich stručný výčet (spolu s dalšími plánovanými opravami či rozšířeními aplikace budou také shrnuty v závěru (kapitola 6)):

- okno *Patterns Manager* nezvládá práci s větším počtem vzorů
- wizard *Kontroly výsledků* se načítá zbytečně velice dlouho (načítá se celý prohledaný obrázek a zobrazí se z něj jenom část, navíc nerespektuje variantu obrázku)
- okno *Kontroly výsledků* by mělo umožňovat také krok zpět, ne jen vpřed
- při špatném nastavení aplikace, které znemožní start *Rychlého spuštění*, není uživatel nijak informován

Je jisté, že nedokonalostí a chyb v aplikaci bude mnohem více, ostatně jako ve všech softwarech (i těch velmi drahých), ale již byl zmíněn důvod nedostatečného testování aplikace jako celku, tedy časový rozsah této práce. Ladění tedy bude probíhat v budoucnu při nasazení do praxe, nebo také jako součást diplomové práce.

Kapitola 6

Závěr

Tato závěrečná kapitola bude shrnutím celé práce hlavně co se týče splněných a nesplněných cílů. Může se to zdát trochu nepatřičné kritizovat vlastní práci, ale právě to bude nedílnou součástí této kapitoly. Je však nutno podotknout, že přestože se některé nedostatky zdají být samostatně snadno opravitelné, jejich celkové odstranění by opět zabralo nějaký ten čas a hlavně by vedlo k nekontrolovatelnému objevování se dalších a dalších nedostatků a chyb a člověk by těžko určil pevnou hranici, kdy má s opravováním přestat.

Nejprve si dovolím shrnout v pár větách kapitolu 2.4, která vytyčuje hlavní požadavky na implementovanou aplikaci. Jak už název tohoto dokumentu napovídá, cílem bylo vytvořit aplikaci, která alespoň částečně zautomatizuje vektorizaci historických map pomocí vyhledávání vzorů. Důležitým požadavkem byla vysoká rychlost zmíněného vyhledávání a v neposlední řadě vytvořit uživatelsky přívětivé grafické rozhraní obsahující všechny potřebné prvky k pohodlnému využívání funkce vyhledávání.

Z předchozího odstavce tedy vyplývá, že práce nebyla zaměřena pouze na vyhledávací algoritmus a jeho optimalizaci. Cílem bylo vytvořit první verzi aplikace k nasazení do praxe, která bude v budoucnu vylepšována a rozšiřována. Zda byl tento cíl splněn, není možno objektivně rozhodnout, jelikož záleží na náročnosti konkrétního uživatele.

Také je nutné zopakovat myšlenku z kapitoly 5 o závěrečném testování vzniklé aplikace, že testování nebylo provedeno v rozsahu, jaký by byl potřeba. Tím pádem je možná existence mnoha dalších chyb či nedostatků, které ve výčtu nacházejícím se níže budou chybět jednoduše proto, že nebyly objeveny.

Následuje tedy výčet splněných cílů a požadavků a také seznam nalezených nedostatků či chyb.

6.1 Splněné cíle

- se správným nastavením vyhledávání je dosaženo dostačující(vzhledem ke vzhledu Müllerovy mapy) přesnosti vyhledávání
- rychlost vyhledávání je možné značně urychlit využitím konkrétního nastavení (rychlost je poté více než vyhovující)

- aplikace obsahuje export souřadnic vyhledaných vzorů do formátu csv a také umožňuje rozšíření o další formáty pomocí pluginů
- díky využití NetBeans platformy je GUI aplikace uživatelsky přívětivé, intuitivní a s vysokou variabilitou uspořádání oken a nástrojových panelů
- kromě exportu do různých formátů je možné rozšířit pomocí zásuvného modulu aplikaci také o nové algoritmy určování podobnosti dvou rastrů
- kvůli zaměření aplikace na práci s obrázky byl vytvořen prohlížeč obrázků s intuitivním ovládáním obsahujícím zoomování, posouvání nebo výběr oblastí pro vytváření nového vzoru
- aplikace obsahuje manažer vzorů s možností editace a mazání existujících vzorů a také jednoduchý manažer skupin vzorů s podobnými funkcemi
- byla vytvořena podpora transformace souřadnic nalezených vzorů v podobě manažeru identických bodů určujících transformaci (souřadnice na mapě je možné získat klikáním na obrázek ve vytvořeném prohlížeči) a dotazu na uživatele při exportu souřadnic, zda si přeje transformace využít
- aplikace obsahuje nástroj pro kontrolu a zároveň opravu výsledků vyhledávání
- kvůli lepším výsledkům vyhledávání je lepší použít černobílé obrázky, proto je uživateli k dispozici nástroj na provedení prahování s libovolným prahem
- algoritmy určující podobnost rastrů (s aplikací je distribuován algoritmus založený na vzájemném korelačním koeficientu) je možné konfigurovat v grafickém rozhraní
- aplikace umožňuje jednoduchou správu projektů umožňující uložení výsledků vyhledávání a následně opětovné načtení

6.2 Nalezené nedostatky

- okno *Patterns Manager* nezvládá práci s větším počtem vzorů
- wizard *Kontroly výsledků* se načítá zbytečně velice dlouho (načítá se celý prohledaný obrázek a zobrazí se z něj jenom část, navíc nerespektuje variantu obrázku)
- okno *Kontroly výsledků* by mělo umožňovat také krok zpět, ne jen vpřed
- při špatném nastavení aplikace, které znemožní start *Rychlého spuštění*, není uživatel nijak informován
- při spuštění vyhledávání na větším počtu obrázků dochází k pádu aplikace z důvodu nedostatku paměti
- přesnost vyhledávání by mohla být značně vylepšena

- uživatel by pravděpodobně ocenil možnost opravit výsledek vyhledávání přímo v prohlížeči obrázků (například při náhodném nalezení chyby by nyní musel spouštět nástroj *Kontrola výsledků*)
- při zobrazení velkého obrázku velice zmenšeného je chování aplikace značně pomalé
- správa projektu je velice jednoduchá a neobsahuje v podstatě žádné funkce

Jak již bylo několikrát zmíněno, oprava nedostatků vypsaných výše i těch ještě neobjevených bude prováděna během testovacího nasazení do praxe, nebo například v rámci diplomové práce.

Pro zadávání a správu chyb byl vytvořen *issue tracking system* (systém pro sledování chyb) FlySpray přístupný na adrese:

http://crishpean.geodetpribram.cz/bugs/index.php?project=3

Do tohoto systému je možné se zaregistrovat a zadávat nalezené chyby, nebo požadavky na změny a vylepšení v aplikaci. Systém je také dosažitelný přes jednoduchou statickou stránku vytvořenou jako podpora aplikace PS na adrese:

http://crishpean.geodetpribram.cz/bc/

Ta se možná v budoucnosti rozšíří a přesune, podle úspěchu aplikace v praxi a požadavků uživatelů.

Literatura

[1] T. Boudreau, J. Tulach, and G. Wielenga. *Rich Client Programming: Plugging into the NetBeans Platform.* Prentice Hall, 1. edition, 2007. ISBN 0-13-235480-2.

[2] M. Dobeš. *Zpracování obrazu a algoritmy v C#.* BEN - technická literatura, Praha, 1. edition, 2008.

[3] J. Fojtík. Algoritmy vektorizace leteckých snímků. `http://www.penguin.cz/~fojtik/vectoris/vektoris_cz.htm`, 28. 1. 1998.

[4] IDL Online Help. Image Tiling. `http://star.pst.qub.ac.uk/idl/Image_Tiling.html`, 16. 6. 2005.

[5] NetBeans Community. NetBeans Platform. `http://platform.netbeans.org/`, 12. 3. 2010.

[6] NetBeans Community. NetBeans Wiki. `http://wiki.netbeans.org/`, 5. 1. 2010.

[7] D. Parks and J.-P. Gravel. Corner detection - Comparison. `http://www.cim.mcgill.ca/~dparks/CornerDetector/comparison.htm`, 3. 4. 2010.

[8] A. Vieiro. Cooking with the NetBeans Platform v 5.5. `http://www.antonioshome.net/kitchen/netbeans/index.php`, 30. 12. 2009.

[9] Canny edge detector - Wikipedia,the free encyclopedia. `http://en.wikipedia.org/wiki/Canny_edge_detector`, 7. 3. 2010.

[10] Corner detection - Wikipedia,the free encyclopedia. `http://en.wikipedia.org/wiki/Corner_detection`, 8. 3. 2010.

[11] Segmentation (image processing) - Wikipedia,the free encyclopedia. `http://en.wikipedia.org/wiki/Segmentation_(image_processing)`, 30. 3. 2010.

[12] Unit testing. `http://cs.wikipedia.org/wiki/Unit_testing`, 4. 3. 2010.

[13] Vectorization (computer graphics) - Wikipedia,the free encyclopedia. `http://en.wikipedia.org/wiki/Vectorization_(computer_graphics)`, 10. 3. 2010.

[14] B. Zitová and J. Flusser. Image registration methods: a survey. *Image and Vision Computing*, 26. 6. 2003.

Seznam obrázků

Seznam tabulek

Seznam ukázek

Příloha A

Přehled modulů a jejich tříd

Tato kapitola obsahuje přehled tříd jednotlivých modulů rozčleněných podle balíčků. Popis je převzatý z javadocu aplikace.

A.1 Modul Patterns Searching

Vzhledem k faktu, že většina tříd tohoto modulu jsou grafické třídy, a tudíž je zde hodně akcí spojených s tlačítky v menu či na nástrojové liště, je jeho nedílnou součástí také soubor *layer.xml*.

patternssearching Kořenový balíček obsahující všechny zdrojové kódy.

- `Main` - Třída upravující akce modulu.

patternssearching.actions - Balíček obsahující samostatné akce.

- `OpenAction` - Akce pro otevření projektu.
- `OpenImageAction` - Akce pro otevření obrázku.
- `ResultsToolbarAction` - Tato třída není vlastní akcí, ale stará se o grafickou reprezentaci lišty nástrojů spravující výsledky.
- `SaveAction` - Akce pro vynucení uložení projektu do souboru.
- `SaveAsAction` - Akce pro uložení projektu do souboru pod vybraným názvem.

patternssearching.actions.run Obsahuje akce související s procesem vyhledávání.

- `QuickRunAction` - Akce, která spustí vyhledávání vzorů v obrázku.

patternssearching.gui.frames Balíček pro grafické komponenty okenního typu.

- `ImageEditorTopComponent` - Komponenta pro zobrazovaní obrázků, případně i výsledků vyhledávání vzorů.
- `PatternsGroupsManagerTopComponent` - Okno pro správu skupin - neboli složek v daném adresáři.
- `PatternsManagerTopComponent` - Okno pro správu vzorů.
- `TransformationTopComponent` - Okno pro správu identických bodů pro transformaci.

patternssearching.gui.frames.dialogs Balíček pro okna typu dialog - nejsou zakomponovaná do hlavního okna.

- `CheckDialog` - Dialogové okno pro kontrolu nalezených vzorů.
- `PatternEditDialog` - Dialogové okno pro editaci konkretního obrázku vzoru.
- `TransformationPointsDialog` - Dialogové okno pro správu identických bodů pro transformaci.

patternssearching.gui.frames.wizards.check Balíček obsahující wizard pro spuštění kontroly vysledků.

- `CheckVisualPanel1` - Panel pro zadání vstupních dat do kontroly výsledků.
- `CheckVisualPanel2` - Panel pro zadání nastavení kontroly výsledků.
- `CheckWizardAction` - Akce spouštějící wizard kontroly výsledků.
- `CheckWizardPanel1` - Kontroler panelu pro určení vstupních dat do kontroly výsledků.
- `CheckWizardPanel2` - Kontroler panelu pro nastavení kontroly výsledků.

patternssearching.gui.frames.wizards.run Balíček pro wizard spuštění vyhledávání.

- `RunVisualPanel1` - Panel pro zadání souborů (obrázků), které se mají prohledat.
- `RunVisualPanel2` - Panel pro určení vzorů, které se mají vyhledávat.
- `RunVisualPanel3` - Panel pro nastavení algoritmů použitých pro vyhledávání.
- `RunWizardAction` - Akce spouštějící wizard pro vyhledávání vzorů.
- `RunWizardPanel1` - Kontroler panelu pro zadání souborů, které se mají prohledat.
- `RunWizardPanel2` - Kontroler panelu pro určení vzorů, které se mají vyhledat.
- `RunWizardPanel3` - Kontroler panelu pro nastavení algoritmů vyhledávání.

patternssearching.gui.frames.wizards.threshold Balíček obsahující wizard pro prahování.

- `ThresholdVisualPanel1` - Panel pro nastavení souborů, které se mají prahovat, a také hodnoty prahu.
- `ThresholdWizardAction` - Kontroler panelu pro nastavení souborů, které se mají prahovat, a také hodnoty prahu.
- `ThresholdWizardPanel1` - Wizard pro spuštění prahování.

patternssearching.gui.imageeditor Balíček obsahující grafické komponenty pro zobrazování obrázků a jejich ovládání.

- `ImageEditor` - Třída pro zobrazování obrázků a jednoduchou práci s nimi.
- `ReferencedImageEditor` - Komponenta zobrazující obrázek se vztažným bodem.
- `TiledImageEditor` - Komponenta zobrazující obrázky po dlaždicích.

patternssearching.gui.imageeditor.canvas Grafické komponenty vykreslující obrázek v zadané velikosti a umístění.

- `CheckImageCanvas` - Plátno pro vykreslování obrázků při kontrolování výsledků vyhledávání.
- `ImageCanvas` - Grafické vykreslování obrázků.
- `ReferencedImageCanvas` - Plátno pro vykreslování obrázků se vztažným bodem.
- `ResultImageCanvas` - Plátno vykreslující obrázek a obdélníky jako zobrazení výsledků.
- `TiledImageCanvas` - Plátno pro vykreslování obrázků po dlaždicích.

patternssearching.gui.imagerenderer Balíček pro třídy související s vykreslováním obrázků do tabulek.

- `ImageRenderer` - Plátno pro vykreslování obrázků v tabulce.
- `ImageTableCellRenderer` - Vykreslovač buněk v tabulce zobrazující vzory.
- `PatternImageRenderer` - Plátno pro vykreslování vzorů v tabulce.

patternssearching.gui.options Balíček pro panely v Options NetBeans aplikace.

- `AlgorithmsOptionsPanelController` - Kontroler panelu pro nastavení defaultních algoritmů vyhledávání.
- `AlgorithmsPanel` - Panel pro nastavení defaultních algoritmů vyhledávání.

patternssearching.gui.toolbars Nástrojové lišty.

- `ResultsToolbar` - Grafická komponenta reprezentující panel nástrojů spravující výsledky vyhledávání.

patternssearching.images Správa obrázků.

- `ImagesStore` - Singleton starající se o uchování, ukládání a loadování obrázků ze souboru.

patternssearching.io Balíček zahrnující třídy související se vstupem a výstupem aplikace.

- `Info` - Pomocná třída obsahující statické metody pro tvorbu dialogů pomocí JOptionPane.
- `LogIDEFormater` - Formátovač logu do IDE okna.
- `LogIDEOutputStream` - Výstupní proud vypisující do IDE okna Log.
- `Output` - Třída obsahující statické metody pro výpis do IDE output okna.

patternssearching.misc Balíček pro nespecifické třídy.

- `DataGetter` - Třída obsahující statické metody pro práci s daty načtenými a zpracovávanými pomoci GUI.

patternssearching.project Balíček pro třídy správy projektu.

- `Project` - Třída uchovávající informace o otevřeném projektu.

patternssearching.resources Balíček pro jiné soubory než .java, např. obrázky, xml soubory.

patternssearching.searching Třídy související s vlastním vyhledáváním vzorů.

- `SearchPatterns` - Singleton třída pro provedení vlastního vyhledávání vzorů v obraze.

A.2 Modul Patterns Searching API

patternssearchingapi.algorithms API pro vytváření algoritmů.

- `Algorithm` - Rozhraní pro vytváření algoritmů určujících podobnost dvou masek.

patternssearchingapi.file.export API pro export souřadnic do různých formátů.

- `Exporter` - Rozhraní pro export bodů do souboru.
- `ExportAction` - Akce vyvolaná uživatelem pro uložení výsledků do souboru.
- `ResultsDialog` - Dialog pro určení souřadnic k exportu.

patternssearchingapi.listeners.filelistener Třídy pro podporu naslouchání změn v souborech.

- `FileChangeListener` - Listener pro změnu souboru na disku.
- `FileEvent` - Událost vyvolaná při změně souborového systému.
- `FileChangeSpeaker` - Třída upozorňující všechny své listenery o změně souboru na disku.

patternssearchingapi.listeners.resultslistener Podpora pro odchytávání událostí souvisejících se změnou výsledků vyhledávání.

- `ResultsChangeListener` - Listener pro změnu výsledků.
- `ResultsEvent` - Událost vyvolaná při změně výsledků.
- `ResultsChangeSpeaker` - Třída upozorňující všechny své listenery o změně výsledků.

patternssearchingapi.options Balíček uchovávající třídy pro správu nastavení.

- `AlgorithmOptions` - Třída obsahující hashmapu pro uchování objektů, slouží k uchování nastavení algoritmu.
- `AlgorithmOptionsPanel` - Třída sloužící jako bázová pro komponenty sloužící k nastavení algoritmu určující míru podobnosti dvou masek.
- `Options` - Sigleton uchovávající veškeré nastavení aplikace.
- `SearchingOptions` - Nastavení vyhledávání vzorů.

patternssearchingapi.pattern Hierarchie správy vzorů.

- `PatternImage` - Třída znázorňující vzor pro vyhledávání.
- `PatternsGroup` - Vlastní vzor, který je vytvářen z obrázků představujících stejný objekt na mapě.
- `PatternsManager` - Singleton spravující vzory.

patternssearchingapi.point Třídy představující body.

- `GeoPoint` - Bod uchovávající souřadnice v přesnosti double.

patternssearchingapi.results Balíček obsahující třídy pro správu výsledků.

- `Result` - Výsledek vyhledávání pro jednu mapu.
- `Results` - Singleton uchovávající výsledky vyhledávání vzorů.

patternssearchingapi.transformation Třídy související s transformací souřadnic.

- `Transformation` - Třída uchovávající body pro transformaci.

A.3 Modul Correlation Algorithm

correlationalgorithm Kořenový balíček obsahující implementaci rozhraní pro tvorbu algoritmu.

- `CorrelationAlgorithm` - Implementace obecného algoritmu do podoby korelačního algoritmu
- `CorrelationAlgorithmOptionsPanel` - Grafický panel pro nastavení parametrů algoritmu CorrelationAlgortihm.

A.4 Modul CSV Export

Tento modul je vybaven také souborem *layer.xml*, jelikož je nutné přidání tlačítka do toolbaru a také položky do menu, aby bylo možné tento export ovládat.

csv.actions Balíček s akcemi.

- `CsvExportAction` - Akce vyvolávající export do csv.

csv.export Třídy provádějící export do csv.

- `CsvExporter` - Implementace exportu do csv.

csv.resources Balíček obsahující jiné než .java soubory, např. layer.xml nebo obrázky.

A.5 Knihovna Utils

V přehledu jsou uvedeny jen relevantní balíčky a třídy.

cz.crishpean.utils.geometry Balíček pro třídy související s geometrií.

- `GeometryUtils` - Obsahuje metody pro práci s geometrickými útvary.

cz.crishpean.utils.image Balíček obsahující třídy související se zpracováním obrazu.

- `Colors` - Třída obsahující metody pro práci s barvami.
- `FeaturesDetection` - Třída obsahující metody pro hledání znaků v obraze.
- `ImageUtils` - Obecné metody k práci s obrázky.
- `Patterns` - Třída obsahující metody pro hledání vzorů v obraze.

cz.crishpean.utils.image.mask Masky pro zpracování obrazu.

- `GreyScaledMask` - Maska pro korelační analýzu obrazu pracující pouze v šedých tónech.
- `GreyScaledMaskMatrix` - Třída znázorňující dvojrozměrné pole, které se používá jako maska pro korelační výpočty.
- `Mask` - Maska pro korelační analýzu obrazu.
- `MaskMatrix` - Třída znázorňující dvojrozměrné pole, které se používá jako maska pro korelační výpočty.
- `PointValue` - Třída obsahující jednu hodnotu umístěnou na určitých souřadnicích.

cz.crishpean.utils.image.mask.similaritymask Masky určující podobnost dvou rastrů.

- `SimilarityMask` - Abstraktní maska používající se při vyhledávání vzorů, pomocí výpočtu podobnosti implementovanými algoritmy.

cz.crishpean.utils.image.mask.similaritymask.areabased Masky určující podobnost dvou rastrů na základě plošných metod.

- `CorrelationMask` - Maska pro hledání vzoru v obrázku využívajícího korelační koeficient jako míru podobnosti.

Příloha B

Instalační a uživatelská příručka

B.1 Instalace a zásuvné moduly

B.1.1 Požadavky na systém

Díky faktu, že je aplikace napsaná v jazyce Java, lze ji spustit na všech operačních systémech, které podporují Java VM (Virtual Machine) a je na nich tedy nainstalováno JRE verze 1.6 a vyšší.

Pro NetBeans IDE 6.8, jehož platforma je základem programu *Patterns Searching*, jsou uváděny následující minimální a optimální hardwarové požadavky na operačním systému Ubuntu 9.04:

Minimální požadavky:

- Procesor: 800MHz Intel Pentium III nebo podobný
- Operační paměť: 512 MB
- Volné místo na disku: 650 MB

Doporučené požadavky:

- Procesor: 2.6 GHz Intel Pentium IV nebo podobný
- Operační paměť: 2 GB
- Volné místo na disku: 850 MB

Pro samotnou aplikaci je však minimální požadavek na operační paměť roven 2 GB, jelikož se aplikace spouští s parametrem -Xms1024M, což nastaví počáteční množství alokované paměti na 1024 MB.

B.1.2 Instalace

Instalace se provádí spuštěním souboru určeného pro uživatelův operační systém. Spustitelný soubor se nazývá *patternssearching-linux.sh* pro operační systém Linux a *patternssearching-windows.exe* pro systém Windows. Po spuštění instalátoru je uživatel proveden instalací klasickým wizardem.

B.1.3 Instalace zásuvných modulů

Po prvním spuštění aplikace je pro správnou funkci nutné nainstalovat některé zásuvné moduly (plugins), pokud uživatel nevlastní verzi s již předinstalovanými moduly. Bez nich bude sice základní funkčnost neporušena, ale některé funkce nebudou dostupné, a to export souřadnic do jakéhokoliv formátu a samotné vyhledávání vzorů. Tyto dvě funkce jsou kvůli případné implementaci jejich dalších variant zprostředkovány formou pluginů.

S instalací programu jsou distribuovány dva soubory typu .nbm:

- *CorrelationAlgorithm.nbm* - umožňuje vyhledávání vzorů využívajíce korelačního koeficientu pro určení podobnosti dvou oblastí obrázku
- *CSVExport.nbm* - provádí export seznamu souřadnic do formátu CSV

Tyto soubory je třeba nainstalovat pomocí NetBeans nástroje *Plugins*:

1. spusťte aplikaci *Patterns Searching*
2. otevřete okno *Hlavní menu -> Tools -> Plugins*
3. v záložce *Downloaded* klikněte na *Add Plugins* a otevřete dva nbm soubory zmíněné výše
4. v okně plugins stiskněte *Install*

Po instalaci modulů je třeba aplikaci restartovat, aby byly jejich funkce k dispozici.
Stejným postupem je možné kdykoliv doinstalovat další kompatibilní zásuvné moduly.

B.2 Základní práce s aplikacemi využívající NB platformu

Předností aplikací, které využívají platformu NetBeans, je především kompaktní grafické rozhraní s možností velmi rozsáhlé variability z pohledu uživatele. Vlastní konfigurace nástrojových lišt (Toolbars) je ve většině softwarů samozřejmostí, co však již tak běžné není, je možnost vlastního uspořádání vnitřních oken aplikace.

B.2.1 Nastavení

Nastavení neboli *Options* je přístupné v *Hlavní menu -> Tools -> Options*. V okně se nachází jedna záložka pro aplikaci *Patterns Searching* (*Algorithms*), která bude popsána níže a defaultní záložky, z kterých uživatel nejvíce ocení *Keymap*, kde je možné nadefinovat vlastní klávesové zkratky.

B.2.2 Nástrojové lišty

Konfigurace nástrojových lišt je možná klasickým přetahováním lišt po nástrojovém menu, nebo pomocí menu, které je dosažitelné buď jako *Popup Menu* kliknutím pravého tlačítka na nástrojové menu, nebo jednoduše přes *Hlavní menu -> View -> Toolbars*.

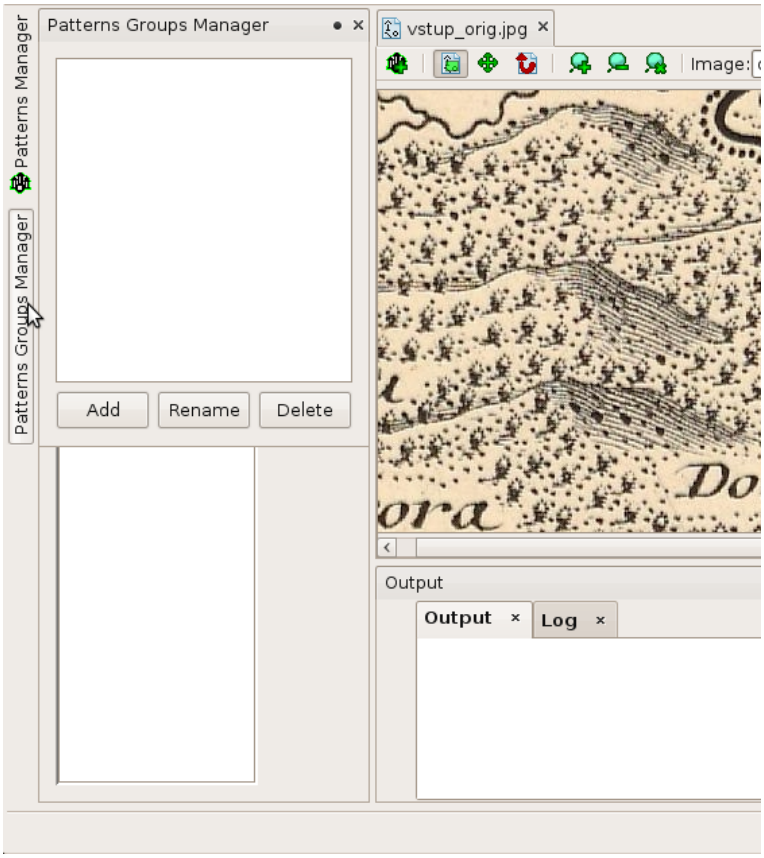
B.2.3 Okna

Veškerá vnitřní okna aplikace lze zavírat běžným způsobem (křížkem v pravém horním rohu) a otvírat pomocí menu *Window*. Velká výhoda aplikací využívající NB platformu je možnost měnit uspořádání oken pomocí přetahování, či vytažení zanořeného okna mimo hlavní okno aplikace funkcí *Undock Window* přístupné v *Popup Menu* daného okna.

Okno se uchopí myší za svou horní lištu a táhne se na požadované místo. Při pohybu se na různých místech objevují oranžové obrazce, které naznačují jak se okno uchytí při puštění.

Okno, které zobrazuje načtené obrázky se dá také maximalizovat kliknutím na tlačítko v pravém horním rohu tohoto okna.

Okna vyskytující se převážně na okrajích můžou existovat ve dvou stavech - minimalizovaný a klasický, jak je znázorněno na obrázku B.1. Klasický stav je vidět na obrázku na okně *Output*, není na něm nic zvláštního a dá se minimalizovat pomocí ikony v pravém horním rohu. Na levé liště jsou vidět dvě minimalizovaná okna (*Patterns Groups Manager* a *Patterns Manager*). Pokud se na takováto okna najede nebo klikne myší, dočasně se zviditelní. Kliknutím na tmavou tečku v pravém horním rohu (*Pin*) se okno přepne do klasického stavu a je tedy viditelné stále.



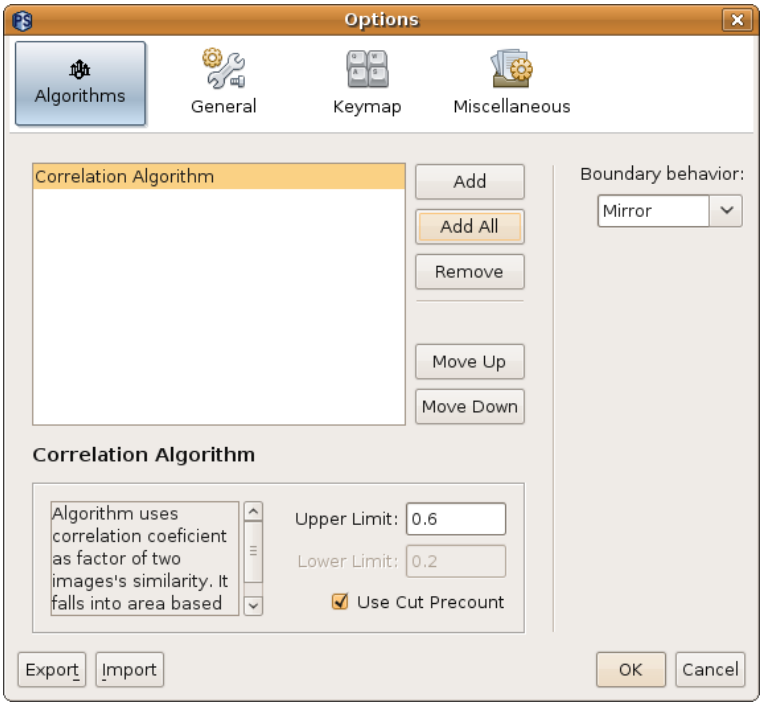
Obrázek B.1: Dva stavy oken

B.3 Používání aplikace Patterns Searching

B.3.1 Nastavení

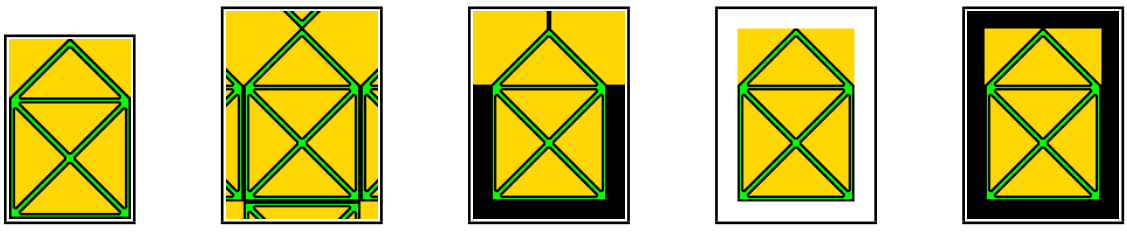
Okno nastavení se otevře v menu *Tools -> Options* a obsahuje stěžejní záložku *Algorithms* obsahující základní nastavení vyhledávání vzorů, jak je vidět na obrázku B.2.

V pravé části okna se nachází combo box s výběrem několika možností chování okrajů obrázku při vyhledávání. Zobrazení těchto možností na jednoduchém objektu je vidět na obrázku B.3.



Obrázek B.2: Nastavení

Levá část okna *Options* obsahuje nastavení defaultních algoritmů používaných hlavně při vyhledávání funkcí *Quick Run*. Do skupiny algoritmů je možné přidat další (pokud jsou dostupné a pokud již v seznamu nejsou) tlačítky *Add* nebo *Add All*, či odstranit vybrané



(a) None (b) Mirror (c) Stretch (d) White (e) Black

Obrázek B.3: Chování na okrajích obrázků

tlačítkem *Remove*. Jsou k dispozici také tlačítka pro změnu pořadí algoritmů, a to *Move Up* a *Move Down*. Každý algoritmus má dále své vlastní parametry, které je možné nastavit ve spodním panelu.

B.3.2 Práce s obrázky

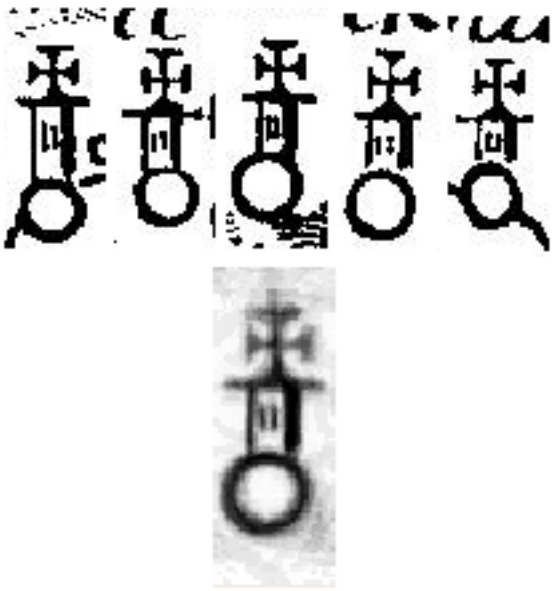
Obrázek (konkrétně mapa) se otevře položkou *Hlavní menu -> File -> Open Image* a po vybrání požadovaného souboru se zobrazí v okně typu *Edtior*. Toto okno obsahuje vlastní panel nástrojů viditelný na obrázku B.4. Toolbar obsahuje následující funkce (v pořadí zleva na zmíněném obrázku):

- *Add New Pattern* - pokud je v otevřeném obrázku vybraná oblast znázorňující vzor, otevře dialogové okno pro přidání tohoto vzoru (obrázek B.6)
- *Selection Tool* - umožní vybírat části obrázku pomocí tažení myši
- *Move* - umožní pohybovat obrázkem tažením myši
- *Set points for transformation* - klikáním pravého tlačítka myši se vyplní příslušná políčka souřadnic v okně *Transformation Window*, které je přístupné z menu *Window*
- *Zoom In* - přiblíží obraz
- *Zoom Out* - oddálí obraz
- *No Zoom* - nastaví původní velikost obrázku
- *Image combo box* - obsahuje seznam existujících variant obrázku, které jsou aplikací podporované (original, threshold, gray) - varianta se zobrazí jejím výběrem
- *Results combo box* - obsahuje výsledky vyhledávání na daném obrázku - výběrem výsledku (název se shoduje s názvem vzoru, který byl hledán) se v obrázku zobrazí barevné obdélníky znázorňující nalezené vzory

Obrázek B.4: Panel nástrojů vieweru obrázků

B.3.3 Správa vzorů

V této aplikaci se vzory dělí do skupin, které z pohledu vyhledávání představují vlastní vzory. Tyto skupiny obsahují libovolný počet obrázků (většinou částí původní mapy), které se pro vyhledávání zprůměrují a vytvoří jeden *kombinovaný vzor*. Příklad této struktury je vidět na obrázku B.5, kde je první řada ukázka jednotlivých vzorů z necelých třiceti vzorů skupiny a osamocený vzor dole je *kombinovaný vzor*.



Obrázek B.5: Jednotlivé vzory a kombinovaný vzor

B.3.3.1 Skupiny vzorů

Skupiny vzorů se editují v okně *Patterns Groups Manager*, kde je vidět jejich seznam s počtem jednotlivých vzorů v závorce a je možné vytvářet nové skupiny, přejmenovávat či mazat stávající, a to tlačítky *Add*, *Rename* a *Delete* umístěnými pod seznamem skupin.

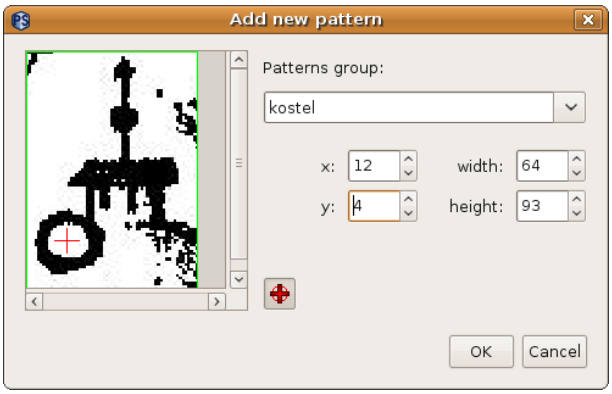
Okno je zobrazeno na obrázku B.7b.

B.3.3.2 Vzory

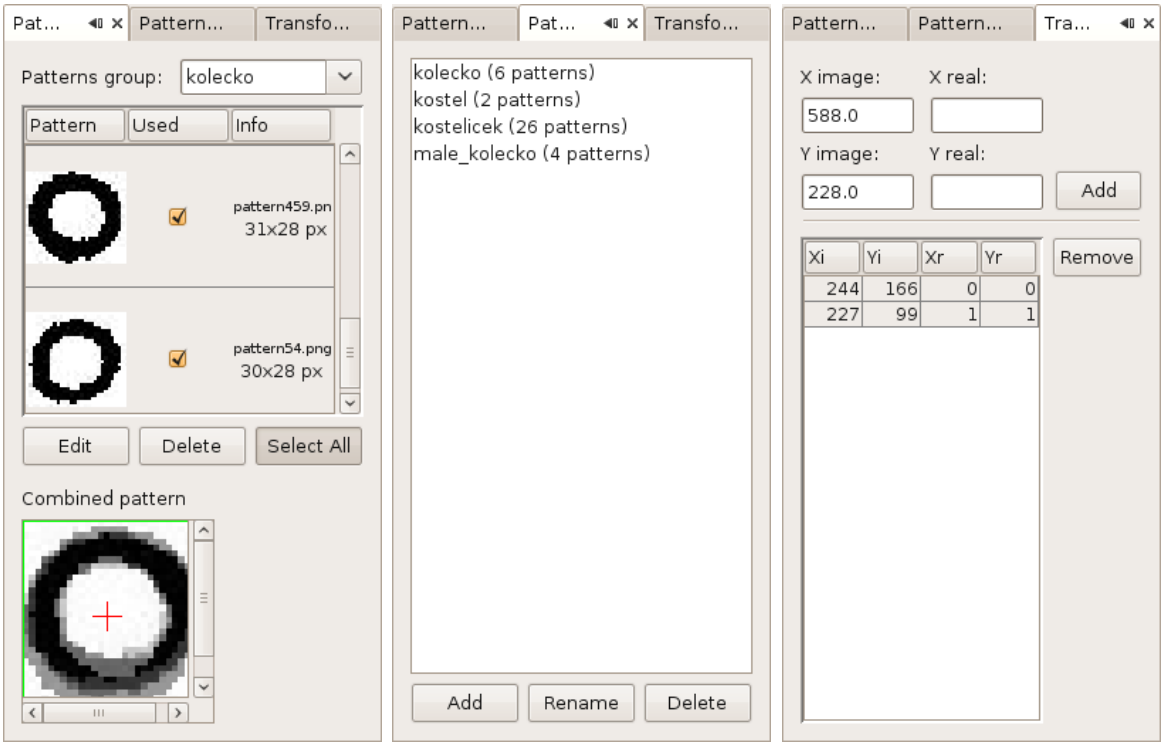
Okno pro spravování jednotlivých vzorů skupin se nazývá *Patterns Manager* (obrázek B.7a). Ve vrchní části je k výběru nabídka skupin, pod níž se nachází seznam vlastních vzorů s náhledem a check boxem určujícím, zda se má vzor použít. Pod seznamem jsou k dispozici tlačítka pro manipulaci se vzory. *Edit* otevře dialogové okno s názvem *Edit Pattern*, jehož obsah a funkce je shodná s dialogem pro uložení nového vzoru (obrázek B.6). Tlačítko *Delete* smaže vybraný vzor a poslední tlačítko *Select All* zaškrtne/odškrtne všechny vzory. Poslední komponenta v okně *Patterns Manager* je náhled *kombinovaného vzoru*.

Při vytváření nového vzoru je třeba držet se následujícího postupu:

1. označit na mapě oblast znázorňující vzor (oblast se dá následně oříznout ale ne zvětšit)
2. kliknout na tlačítko *Add New Pattern* v toolbaru vieweru obrázků
3. v otevřeném dialogu (obrázek B.6) vybrat skupinu, pomocí souřadnic a velikosti určit výsledný vzhled vzoru a případně přidat referenční bod aktivováním tlačítka vpravo dole u náhledu vzoru a kliknutím na požadované místo ve vzoru
4. potvrdit tlačítkem *OK*



Obrázek B.6: Uložení nového vzoru



(a) Patterns Manager (b) Patterns Groups Manager (c) Transformation Window

Obrázek B.7: Okna pro správu vzorů a transformací

B.3.4 Transformace souřadnic

Pro každou mapu je možné nastavit transformaci souřadnic pomocí bodů určených ve dvou souřadnicových soustavách. Tyto body se editují v okně *Transformation Window* (obrázek B.7c).

Souřadnice bodů z obrázku se získávají klikáním pravým tlačítkem na mapu s aktivovanou volbou *Set points for transformation* v panelu nástrojů. Souřadnice z druhé soustavy se vepíší

do políček *X real* a *Y real* a bod se přidá do seznamu tlačítkem *Add*. Ze seznamu se dají body odstranit stisknutím tlačítka *Remove*.

Souřadnice se transformují pouze při exportu, a to podobnostní transformací.

B.3.5 Zpracování obrázků a vyhledávání

Pro spuštění vyhledávání existují dvě možnosti, a to klasický start, kterým uživatele provede wizard se třemi panely pro nastavení, nebo rychlý start bez jakéhokoliv dialogu. Pro kterýkoliv druh je doporučené (pokud to povaha obrázku dovoluje), provést na obrázku prahování, neboli thresholding (převedení obrázku na černobílý). Samozřejmě je pro optimalizaci vyhledávání požadováno, aby vzory byly také vytvářeny z těchto prahovaných obrázků.

B.3.5.1 Prahování (Thresholding)

Prahování je typ zpracování obrazu, při kterém je obrázek převeden na černobílý pomocí takzvaného prahu, který rozdělí pixely obrazu na dvě skupiny - světlejší pixely než je práh a ty tmavší. Barvy těchto pixelů se tedy změní na černou a bílou podle jejich příslušnosti k dané skupině, jak to znázorňuje obrázek B.8 s prahem 128.

Obrázek B.8: Prahování (práh = 128)

Provést thresholding je možné dvěma způsoby. První z nich je otevřením dialogu *Thresholding* z menu *Hlavní menu -> Run -> Thresholding*. Příslušný dialog je podobný prvnímu panelu z wizardu *Run*, který je zobrazen na obrázku B.9. Tři radiová tlačítka ve vrchní části dialogu určují způsob vybírání obrázku, na který se má prahování aplikovat. První dva jsou v téhle chvíli totožné a určují obrázek otevřený programem. Třetí volbou uživatel může přesně specifikovat vstupní soubory tlačítky *Add* a *Remove*. Ve spodní části se nachází políčko pro určení hodnoty prahu. Proces se spustí tlačítkem *Finish*. Při použití funkce *Thresholding* se přepíší u daných obrázků již vytvořené prahované obrázky.

Druhá ze dvou možností, je použití prahování přímo při spuštění vyhledávání. Jak je vidět na obrázku B.9, ve spodní části lze zaškrtnout políčko pro prahování a nastavit mu hodnotu. Pokud se tak učiní, bude u obrázků které již prahování někdy absolvovaly, použita

již vytvořená prahovaná verze (i přesto že mohla být vytvořena s jinou hodnotou prahu). Pro mapy, které threshold verzi nemají, se vytvoří se zadaným prahem.

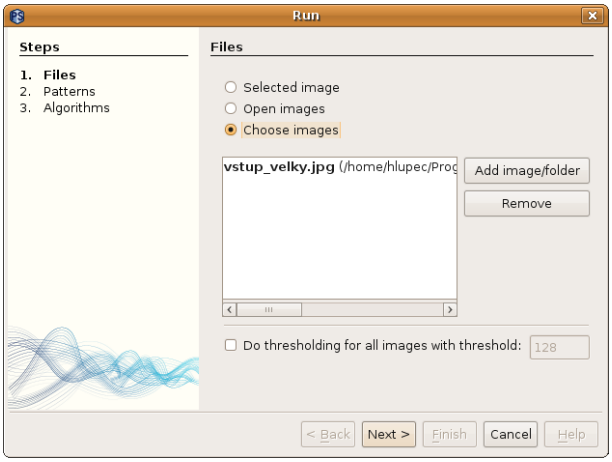
B.3.5.2 Rychlý start (Quick Run)

Metoda spuštění zvaná *Quick Run* provede vyhledávání na otevřeném obrázku s defaultním nastavením algoritmů (změnit nastavení je možné v *Hlavní menu -> Tools -> Options*) a právě vybranou skupinou vzorů v *Patterns Manager*.

B.3.5.3 Vyhledávání (Run)

Pomocí wizardu *Run*, který se spustí položkou *Hlavní menu -> Run -> Run*, je možné nastavit obrázky pro vyhledávání, sled vzorů a algoritmů a jejich nastavení.

První panel wizardu (obr. B.9) umožňuje specifikovat obrázky pro vyhledávání a také použití prahování. Tento panel byl popsán v sekci B.3.5.1 o prahování.



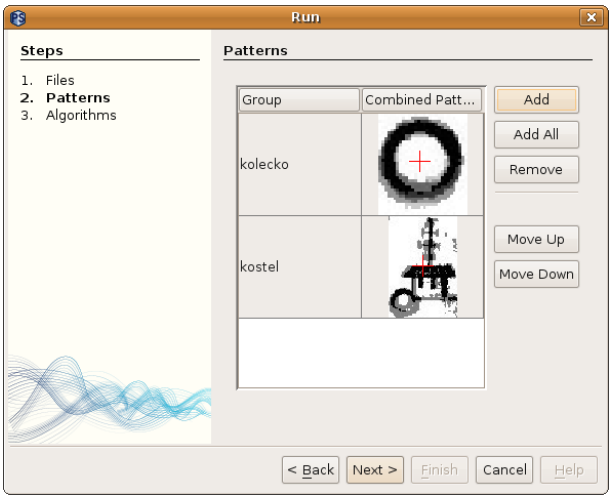
Obrázek B.9: Výběr souborů pro aplikaci vyhledávání

Následující panel je pro specifikaci vzorů, které se budou vyhledávat, a jejich pořadí. Doporučuje se vyhledávat nejprve větší vzory a nakonec menší. Pokud totiž nějaký velký vzor obsahuje také část podobnou vzoru malému, tímto postupem se eliminuje možnost že se bod označující polohu vzoru uloží do výsledku dvakrát (pro malý a velký vzor). V pravé části panelu se nacházejí tlačítka pro správu seznamu vzorů - přidání je možné pomocí *Add* (otevře dialog pro výběr vzoru) nebo *Add All* (přidá všechny existující vzory), odebrání stiskem *Remove* a pro změnu pořadí jsou *Move Up* a *Move Down*.

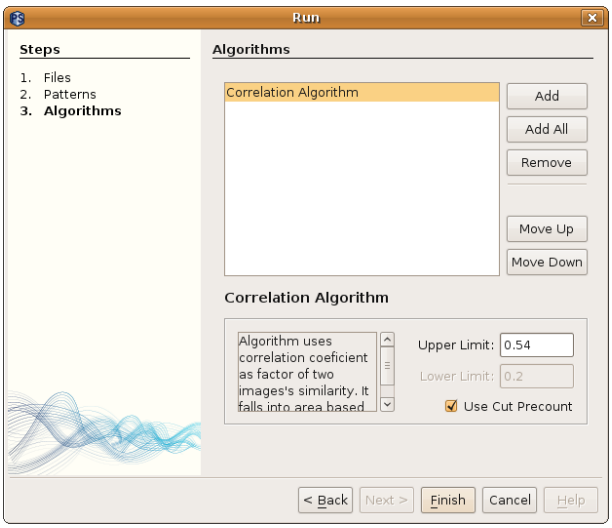
Poslední panel wizardu *Run* slouží k nastavení algoritmů používaných pro toto konkrétní vyhledávání. Je znázorněn na obr. B.11 a funkčně je shodný s levou částí záložky *Algorithms* v *Options* popsané v sekci B.3.1.

B.3.5.4 Kontrola výsledků (Check)

Kontrola výsledku pomáhá uživateli projít mapu s vyhledanými vzory a přidat či odebrat některá místa, která vyhledávací algoritmus nerozpoznal správně.



Obrázek B.10: Nastavení vzorů pro vyhledávání



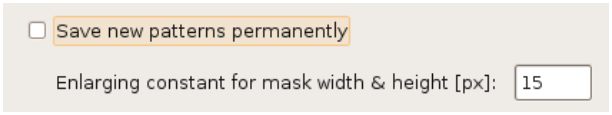
Obrázek B.11: Nastavení algoritmů pro vyhledávání

Wizard pro spuštění kontroly se nachází pod položkou *Hlavní menu -> Run -> Check* a obsahuje dva panely. První z nich je pro určení konkrétního výsledku ke kontrole a je zobrazen na obr. B.12. Výsledky jsou v aplikaci uchovávány ve formě stromu - pro každou mapu (určenou souborem) všechny její výsledky (určené názvem vzoru). Proto je třeba vybrat nejprve soubor a poté vzor ve dvou seznamech ve vrchní části prvního panelu. Pod nimi je ještě náhled velikosti mapy, která se dá měnit scrollováním nebo hodnotou vpravo od náhledu, ale nedá se již změnit jakmile se kontrola spustí.

Ve druhém panelu je pouze nastavení chování kontroly, které je možné ponechat beze změny. První položka, jeli zaškrtnutá, znamená, že se oblasti, které uživatel označí jako vzory (algoritmus je nesprávně označil za běžné oblasti), permanentně uloží na pevný disk a bude možné je při příštích vyhledáváních používat. Zde je nutné dodat, že více vzorů sice zpřesní vyhledávání, ale příliš mnoho vzorů určitou měrou sníží výkon například při

Obrázek B.12: Určení výsledků pro kontrolu

používání *Patterns Manageru*. Druhá položka v tomto panelu určuje, o kolik se zvětší vyhledávací okénko kolem kurzoru myši oproti kontrolovanému vzoru. Toto okénko je také vidět na obrázku B.14.



Obrázek B.13: Nastavení kontroly výsledků

Vlastní kontrola probíhá v jednom dialogovém okně, pomocí kterého se postupně projde celá mapa a umožní opravit chyby vzniklé automatickým vyhledáváním. Ihned po otevření dialogu je možné zvětšit okno tak, jak bude uživateli vyhovovat. To je sice možné i po prvním kroku, ale velikost kroku se již nezmění. Jak je vidět na obrázku B.14, dialogové okno obsahuje panel nástrojů ve vrchní části, náhled mapy s výsledky a spodní část s informací o množství kroků a tlačítky pro ukončení kontroly.

Postup kontroly je tedy následující:

1. nastavte velikost dialogového okna tak, jak vám to bude vyhovovat (poté to již není vhodné)

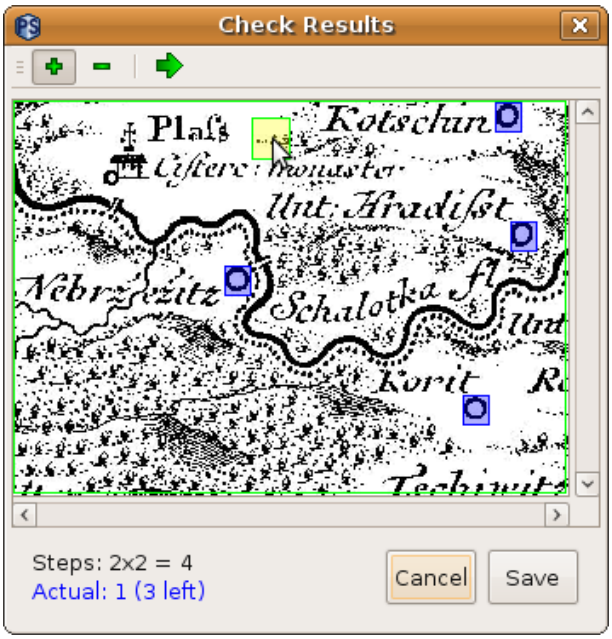
2. prohlédněte zobrazenou oblast a opravte chyby - v základním módu (v panelu nástrojů je vybrané tlačítko s ikonkou plus) se kliknutím levého tlačítka myši daná oblast pod vyhledávacím okénkem prohledá a místo s nejlepší shodou se přidá jako vzor a kliknutím pravého tlačítka na nějaký označený vzor se tento vzor odstraní; v módu s vybraným tlačítkem mínus fungují tlačítka obráceně a nezobrazuje se vyhledávací okénko

3. přejděte na další oblast mapy kliknutím prostředního tlačítka myši, nebo stisknutím tlačítka v toolbaru se šipkou směřující doprava

4. opakujte kroky 2 a 3 dokud neprojdete celou mapu

5. při zvolení dalšího kroku na poslední oblasti mapy se kontrola ukončí a zobrazí se dialog ptající se na uložení změn

Během kontroly je možné ji ukončit buď tlačítky v pravém spodním rohu, nebo zavřením okna, vždy se ale zobrazí dialog s otázkou, zda si je uživatel jist a chce-li změněné výsledky uložit.



Obrázek B.14: Kontrola výsledků

B.3.6 Správa výsledků

Výsledky jsou uchovávány pro každou mapu a vzor po dobu běhu aplikace. Mapy které již nějaké výsledky mají se objeví v rolovací nabídce toolbaru *Results* (obr. B.15). Tento panel nástrojů obsahuje zleva následující funkce:

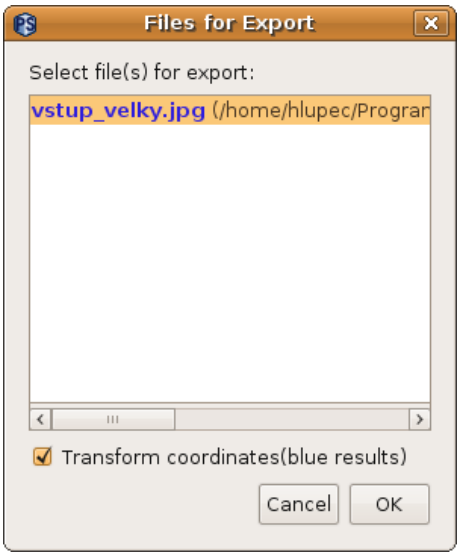
- *Results for file* - výběr z existujících výsledků popsaných názvem souboru (mapy)
- *Display* - otevře danou mapu v prohlížeči obrázků, kde je možné zobrazit výsledky pro určitý vzor jak bylo popsáno v kapitole B.3.2
- *Clear* - smaže veškeré výsledky pro vybranou mapu
- *Transformation* - otevře dialogové okno pro správu identických bodů (obsah okna je shodný s obsahem *Transformation Window* popsaném v sekci B.3.4)

Obrázek B.15: Panel nástrojů pro správu výsledků

B.3.6.1 Export souřadnic

Export souřadnic je umožněn díky nainstalovaným zásuvným modulům. Je tedy možné, že by některý modul obsahoval pro export nějaká další okna, to však musí být náplní uživatelské příručky příslušného modulu. Při exportování souřadnic se vždy objeví okno zobrazené na obrázku B.16. V tomto okně je možné vybrat jeden či více souborů (map) pro export. Modré položky v seznamu mají nastavené identické body pro transformaci a pokud je ve spodní části zaškrtnuto políčko *Transform coordinates*, tato transformace se na souřadnice aplikuje.

Po označení exportovaných výsledků se otevře dialog *Save* pro výběr souboru, do kterého se mají souřadnice uložit. Může záležet na konkrétním druhu exportu, ale je běžné, že pokud je vybráno více map, kterým se mají výsledky exportovat, vytvoří se více souborů se souřadnicemi - pro každou mapu jeden, a jejich jména se budou skládat z názvu zadaným uživatelem a čísla, které se automaticky generuje.



Obrázek B.16: Export souřadnic

B.3.7 Projekty

Správa projektů je v aplikace *Patterns Searching* spíše provizorní. Projekt lze pouze uložit a poté zase načíst, a to do XML souboru. Konkrétně se ukládá/načítá seznam otevřených obrázků a veškeré výsledky, které aplikace uchovává během běhu.

Příloha C

Obsah přiloženého CD

bin/ - adresář obsahující instalační soubory aplikace Patterns Searching

doc/ - javadoc k vytvořeným modulům a knihovně Utils

src/ - adresář s veškerými zdrojovými kódy

text/ - text bakalářské práce a také zvláštní soubor s uživatelskou příručkou

index.html - statická stránka sloužící jako rozcestník k obsahu CD