VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ONLINE SYSTÉM PRO VIZUÁLNÍ GEO-LOKALIZACI V PŘÍRODNÍM PROSTŘEDÍ

ONLINE SYSTEM FOR VISUAL GEO-LOCALIZATION IN NATURAL ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MIROSLAV POSPÍŠIL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN BREJCHA**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/21220/2017/xpospi73

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Pospíšil Miroslav, Bc.**
Obor: Informační systémy
Téma: **Online systém pro vizuální geo-lokalizaci v přírodním prostředí**
Online System for Visual Geo-Localization in Natural Environment
Kategorie: Zpracování obrazu

Pokyny:

1. Proveďte rešerši existujících algoritmů pro vizuální geo-lokalizaci, zahrnující algoritmy pro předzpracování obrazu, jako je např. detekce křivky horizontu.
2. Navrhněte architekturu webového systému pro vizuální geo-lokalizaci. Zaměřte se na interaktivitu, rychlost řešení a možnosti škálování.
3. Navrhněte jednoduché a snadno použitelné webové rozhraní pro vizuální geo-lokalizaci v přírodě. Zaměřte se na jednoduchost a snadnou použitelnost.
4. Implementujte nový, nebo vylepšete existující algoritmus pro vizuální lokalizaci tak, aby byl použitelný ve webovém systému.
5. S implementovaným systémem experimentujte a diskutujte výsledky.
6. Vytvořte plakát nebo video demonstrující implementovaný systém.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body zadání 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Brejcha Jan, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
L.S. 612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této diplomové práce je vytvořit online systém, který bude fungovat jako demonstrační aplikace pro prezentaci výsledků vizuální geo-lokalizace v přírodním a horském prostředí. Systém nabídne uživateli možnost vybrat si jednu z předdefinovaných fotografií nebo nahrát vlastní fotografii výběrem souboru nebo zadáním URL adresy. Systém bude hledat pozici kamery daného obrázku na základě vizuální geo-lokalizace. Geo-lokalizace využívá horizontu hor jako klíčovou charakteristiku pro vyhledávání podobných horizontů. Křivka horizontu je extrahována z fotografie plně automatickým algoritmem, založeným na strojovém učení s učitelem a dynamickém programování. Vizuální geo-lokalizace probíhá na serveru, který využívá nový inverzní index s cachovací politikou umožňující další škálování systému. Server zpracuje detekovanou křivku horizontu a vrátí nejlepší kandidáty na výsledky, které jsou pak vizualizovány uživateli formou klasické mapy, detailního satelitního pohledu a vykreslení nalezeného panoramatu.

Abstract

The goal of this master thesis is creation of an online system serving as a performing application for presentation results of visual geo-localization in nature and mountain environment. The system offers the users to choose one of the pre-defined photographs or to upload one's own photography while choosing a file or inserting an URL address. The system will localizate a camera of a given image based on a visual geo-localization. The geo-localization uses the mountain horizon as a key characteristic when searching for similar horizons. The curve line of the horizon is extracted by a fully automatic algorithm based on supervised learning and dynamic programming. Visual geo-localization running on the server which using new inversed index with cache politic. This allows further scaling of the system. The server processing detected horizon curve and respond with set of the best candidates on results. Results are visualised to the user in form of classic map, detailed sattelite view and rendering of found panorama.

Klíčová slova

Online systém, vizuální geo-lokalizace, vizuální geo-lokalizace v přírodním prostředí, horské prostředí, umělá inteligence, strojové učení s učitelem, lokalizace křivky horizontu, dynamické programování, projekt Locate, inverzní index, cachovací politiky, vizualizace geografický dat.

Keywords

Online system, visual geo-localization, visual geo-localization in natural environment, mountain environment, artificial intelligence, supervised learning, skyline localization, dynamic programming, project Locate, inversed index, cache politics, visualisation of geographic data.

Citace

POSPÍŠIL, Miroslav. *Online systém pro vizuální geo-lokalizaci v přírodním prostředí.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Brejcha

Online systém pro vizuální geo-lokalizaci v přírodním prostředí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Brejchy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Pospíšil
22. května 2018

Poděkování

Rád bych poděkoval panu Ing. Janu Brejchovi za poskytnuté konzultace a odborné vedení práce. Dále chci poděkovat překladatelům a korektorům jazykových verzí webové aplikace: Bc. Viera Vozárová, Bc. Tomáš Michalek, Mgr. Jakub Dušek a Ing. Carlos Filipe Do Lago Vieira.

Obsah

Kapitola 1

Úvod

Geo-lokalizace je problémem, jehož řešení má velmi užitečné a rozmanité uplatnění. Geo-lokalizaci můžeme definovat jako odhad geografické lokace zpracovávaného obrázku nalezením nejpodobnějších referenčních obrázků [47]. S možností zjistit polohu na základě pořízené fotografie se otevírají další možnosti jak dohledávat další informace o významu a obsahu fotografie. Vizuální geo-lokalizace a její cíl zjisit místo pořízeného obrazového záznamu má významné využití například pro historické a forenzní vědní obory, dokumentování, uspořádání a kategorizaci fotografických dokumentací světa, a také pro inteligentní aplikace [8]. Určení polohy může probíhat v civilizovaném prostředí (ulice, města), ale také v přírodním prostředí (lesy, řeky, hory). Tato práce se vymezuje na lokalizaci v přírodním prostředí, což je náročná disciplína spíše kvůli vegetaci, osvětlení, ročním obdobím, oblačnosti a podobně.

V této práci se zaměřuji na určování polohy, ve které se využívá horizontu hor, která s sebou nese významné sémantické informace. Křivka horizontu určuje profil hor. Můžeme z ní detekovat tvary jednotlivých hor, údolí, převisy nebo konkrétní vrcholy. Horizont hor můžeme porovnávat s jinými horizonty a najít tak podobné horizonty, u kterých známe zeměpisnou šířku a délku. Automatická detekce horizontu má velké využití právě ve vizuální geo-lokalizaci – například pro roboty a létající drony, které mohou na základě horizontu lépe odhadovat svou polohu [6].

První kapitola analyzuje bakalářskou práci [34], na kterou tato práce navazuje. V této kapitole upozorňuji na důležité problémy existujícího řešení a tím vytvářím motivaci pro tuto práci. V další kapitole jsou zpracovány rešerše nutné pro získání přehledu problematik algoritmů pro detekci horizontu, systémů pro vizuální geo-lokalizaci a možností vizuální prezentace geografických dat. Třetí kapitola zasvětí do implementace algoritmu pro lokalizaci křivky horizontu. Je zde vybrán vhodný algoritmus a postupně implementovány všechny jeho části. S implementovaným algoritmem je dále experimentováno a provedeno srovnání s dalšími konkurenčními algoritmy [4], [11], [26], [28]. Kapitola je zakončena profilací algoritmu a jeho následnou optimalizací za účelem co nejrychlejšího běhu, což je jeden z klíčových požadavků na nový systém. Následující kapitola líčí návrh webového systému a s tím související návrhy architektury a uživatelského rozhraní. Na návrhy přímo navazuje kapitola s implementací online systému a integrací lokalizace křivky horizontu. Práce je zakončena zhodnocením odvedené práce a náhledem na možný budoucí vývoj projektu.

1.1 Předchozí práce

Webová aplikace pro vizuální geo-lokalizaci již byla implementována v rámci mé bakalářské práce [34]. Cílem bylo vytvořit webovou aplikaci, která umožní uživatelsky přívětivou vizuální geo-lokalizaci fotografií na základě detekování křivky horizontu. Aplikace je zaměřena na uživatele, a proto návrh i výsledná aplikace byla podrobena uživatelskému testování. Aplikace mimo jiné nabízí také možnost výběru obrázku ze služby Flickr, která byla přímo integrována do aplikace. Dále je pro uživatele připravena možnost registrace a přihlášení, kde oproti neregistrovaným, získá uživatel výhody ukládání nahraných obrázků a zachování historie provedených vyhledávání.

Nejdůležitější částí v aplikaci je vyhledávání panoramat a s tím spojená detekce křivky horizontu. Proces extrakce křivky horizontu probíhá poloautomaticky ve dvou krocích. V práci je použit algoritmus GrabCut, který se sám pokusil odhadnout křivku horizontu. Tento algoritmus ale není na segmentaci oblohy vhodný kvůli okluzím v podobě vegetace, budovám, povětrnostním podmínkám a různým sezónním obdobím. Například je křivka horizontu špatně rozeznatelná od oblohy v případě, že jsou vrcholky pokryté sněhem. Proto je odhadnutá křivka zobrazena uživateli, který ji může interaktivně v aplikaci opravit pomocí pivotových bodů. Jednoduchým potáhnutím může uživatel body posunout do míst horizontu a tímto způsobem opravit chybně označenou křivku horizontu. V dalším zpracování se uživatelem opravená křivka použije pro novou detekci v GrabCut, který produkuje výsledný horizont. Z detekovaného horizontu je následně odhadnuta pozice kamery pomocí algoritmu Large Scale Visual Geo-Localization of Images in Mountainous Terrain [8]. Výsledky jsou komprimovány a zabaleny do ZIP archivu a jsou odeslány zpět do aplikace, která je zpracuje a vizualizuje uživateli společně s integrovanými Google mapami zaměřenými podle zeměpisných souřadnic u každého výsledku.

1.1.1 Analýza problémů předchozí implementace

Předchozí aplikace [34] splnila všechny své požadavky avšak výsledná aplikace a následné uživatelské testování odhalilo, že systém obsahuje problémy, které je třeba eliminovat. Základní funkcí aplikace je vyhledávání panoramat za pomoci detekované křivky horizontu. Celý proces určení křivky horizontu je však poloautomatický a vyžaduje časový interval v rozmezí 150–300 sekund. Dalším velkým problémem je časté nepochopení výsledků uživateli což ukazuje, že způsob prezentování výsledků není vhodný. Grafické uživatelské rozhraní aplikace má velké množství různých obrazovek, což může působit dojmem komplikovaného a rozsáhlého systému a odradit tak nově příchozího uživatele. Front-end aplikace má většinou kladné ohlasy, ale jeho vzhled neodpovídá současným moderním webovým aplikacím a nemůže konkurovat aplikacím[1] podobného zaměření.

Největšími nedostatky v aplikaci tedy jsou:

- Rychlost aplikace.
- Polo-automatická detekce horizontu vyžadující zásah uživatele.
- Nevhodná vizualizace nalezených výsledků ze systému Locate.

[1] Je pravděpodobné, že uživatel raději použije jinou aplikaci, která je lépe graficky zpracována a vzbudí tak u uživatele větší zájem.

- Velký počet jednotlivých obrazovek aplikace.
- Přiměřeně estetický a přívětivý design grafického uživatelského rozhraní, který ale nemůže konkurovat současným aplikacím podobného zaměření.

Těchto pět bodů, vycházejících z analýzy problémů předchozí implementace, shledávám jako nejvýznamnější důvody pro vytvoření nového systému, který bude explicitně zaměřen na eliminaci těchto překážek.

Kapitola 2

Rešerše existujících algoritmů

Tato kapitola představuje teoretickou bázi v této práci a rozebírá čtyři hlavní témata, která jsou v této práci využita. Každé téma je rešerší na konkrétní skupinu algoritmů, systémů či služeb. Vyhledány jsou *state-of-the-art* metody, které daná skupina nabízí. Rešerše byly hledány v rámci vědeckých publikací. Pro rešerši poslední skupiny jsou zde na místě také webové služby uznávaných společností nebo postupy uznávané širokým okruhem vývojářů a dalších uživatelů. Každá rešerše se vždy skládá z obecného pohledu na věc, ze sumarizace a krátkého uvedení jednotlivých zástupců, a na to navazující široké a podrobné pojednání každého zástupce včetně zvýraznění nejdůležitějších aspektů, výhod a nevýhod.

2.1 Rešerše algoritmů pro detekci horizontu

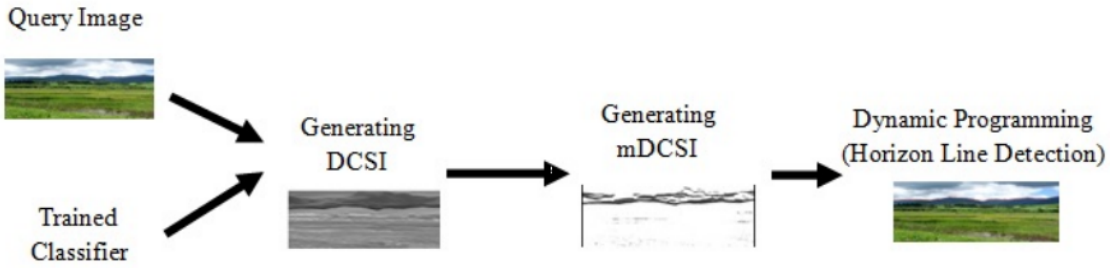
Pro možnost vytvoření plně automatické detekce křivky horizontu bylo potřeba uskutečnit podrobnou rešerši relevantních algoritmů. Výběr se soustředil výhradně na algoritmy, které se týkaly přímého detekování křivky horizontu v přírodním prostředí, ale také algoritmy, které se týkaly sémantické segmentace obrazu. Automatická detekce horizontu se v tomto systému zavádí za účelem časového zrychlení (avšak s dostatečnou přesností), a tím i zlepšení přívětivosti a použitelnosti pro uživatele. Většina takovýchto algoritmů [5], [11], [24], [26], [28] využívá strojového učení, které si v procesu sémantické segmentace obrazu nebo detekování objektů v obrazu našla své pevné místo, protože poskytuje solidní rychlost, vysokou přesnost a úspěšnost, čímž zastiňuje konkurenční algoritmy nevyužívající strojové učení.

Mezi vybrané algoritmy se řadí An Edge-less Horizon Line Detection [4], který využívá klasifikačních ohodnocení jednotlivých pixelů na rozdíl od většiny ostatních algoritmů, které používají detekované hrany pro extrakci křivky horizontu. Na základě klasifikace jsou získány vícestupňové grafy, ze kterých se dále extrahuje nejkratší cesta. Klasifikace společně s dynamickým programováním (viz. kap. 3.8) a počítáním velikostí gradientu využívá framework ALE - The Automatic Labelling Environment [26]. Metoda provádí sémantickou segmentaci do více tříd a využívá natrénovaný klasifikátor. Sémantické segmentaci se věnují i následující dva algoritmy. Fully Convolutional Neural Networks (FCNs) [28] také využívá strojového učení, a to konkrétně trénování plně konvolučních neuronových sítí. Využívá end-to-end a pixels-to-pixels přístupu takže pro každý vstupní pixel produkuje jeden výstupní podobně jako ALE. FCNs a SegNet [11] je hluboká enkodér-dekodér architektura pro segmentaci pixelů do tříd. Architektura je rozdělena do nelineárních vrstev - kodéry a korespondující množinou vrstev pro dekodéry. Poslední metody jsou zaměřeny na

přímou extrakci křivky horizontu s pomocí detekce hran a natrénovaného klasifikátoru pomocí SVM[1] (viz. kap. 3.3). První je Machine Learning Approach to Horizon Line Detection [5], která využívá pouze hrany, které projdou filtrem rozsáhlého prahování[2] (wide range thresholding). Metoda dále využívá dynamické programování pro nalezení nejkratší cesty. Skyline Localization for Mountain Images [24] z detekovaných hran vytváří segmenty jejichž informace o okolí jsou použity pro trénování i predikci. Horizont se pak získává z gradientní mapy a je opět využito dynamického programování pro nalezení nejkratší cesty.

2.1.1 An Edge-less Horizon Line Detection

Většina algoritmů sémantické segmentace pracuje na extrakci hran a získávání informací z jejich okolí. Metoda Edge-less Horizon Line Detection [4] využívá strojového učení a dynamického programování pro extrahování horizontu z klasifikační mapy namísto mapy hranové. Každému pixelu je přiřazeno klasifikační skóre, které udává míru pravděpodobnosti, že pixel patří do horizontu. Klasifikační mapa je reprezentována vícestupňovým grafem[3] (multi–stage graph), ve kterém je pomocí dynamického programování extrahována cesta s nejmenší cenou (suma všech ohodnocení v cestě). Algoritmy využívající hrany jsou většinou binární a obsahují mezery. Tato metoda je spojitá a mezery neobsahuje.



Obrázek 2.1: Proces zpracování v Edge-less Horizon Line Detection
převzato z [4]

Pro natrénování klasifikátoru je užito strojového učení s učitelem, kde bylo použito SVM a konvolučních neuronových sítí. Trénování probíhá pro každý obrázek z trénovací sady, ze kterého se extrahuje N bodů uniformě z ground truth a stejný počet je vybrán náhodně z oblastí, které nejsou součástí horizontu. Pro trénování i následnou predikci se generuje DCSI (Dense Classifier Score Image) pro daný obrázek. Pro každý vybraný pixel (při predikci - pro každý pixel v obrázku) je generován 16×16 px oblast, kde se počítají intenzity. Normalizované intenzity jsou použity pro vytvoření 256 dimenzionálního vektoru, který je předán klasifikátoru. Výsledek predikce je normalizován do intervalu [0..1] a asociován s lokací pixelu. Z DCSI se dále vytváří redukovaná varianta (mDCSI), kde se ponechává pouze m nejvyšších skóre pro každý sloupec. Nejvyšší skóre jsou typicky koncentrovány poblíž horizontu. Korespondující vícestupňový graf s mDCSI obsahuje méně vrcholů a tím pádem

[1] Support Vector Machines - Strojové učení s učitelem

[2] Metoda segmentace obrazu založená na jasovém ohodnocení každého pixelu. Z hodnot je vytvořen histogram a hledá se taková hodnota (prahu), pro kterou bude platit, že nižší hodnoty patří pozadí / vyšší patří popředí. [21]

[3] Vícestupňový graf G = (V, E) je orientovaný graf, ve kterém jsou jednotlivé uzly rozděleny do k stupňů, kde k ≥ 2. Problém ve vícestupňovém grafu je nalezení nejkratší cesty ze Source do Sink uzlu. [35]

výsledek obsahuje méně cest na výběr při hledání nejkratší cesty což má značný dopad na rychlost zpracování.

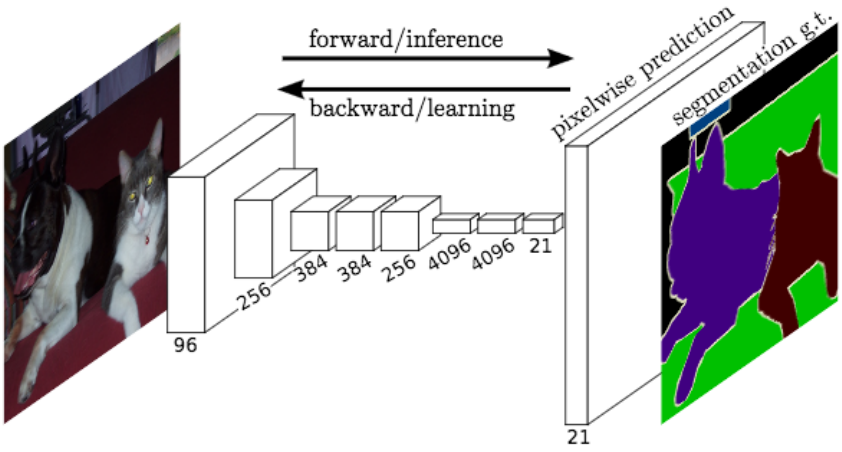
V porovnání k [24] je nejvýznamnějším rozdílem, že tento algoritmus nevyužívá hranové mapy. Vektory jsou zde extrahovány pro každý pixel a jeho okolí o velikosti 16×16 px, zatímco v [24] se extrahují vektory pro celý hranový segement o velikosti 8×8 px. S tím jsou spojeny také různé velikosti vektorů. Důležitým rozdílem je, že v tomto algoritmu jsou čerpány pouze jasové informace z oblastí okolo pixelů, kdežto v [24] jsou z oblasti extrahovány hodnoty barevných složek RGB modelu pro každý pixel oblasti, průměrné jasové hodnoty a souřadnice pixelů.

2.1.2 The Automatic Labelling Environment (ALE)

Lubor Ladický a Philip H.S. Torr společně se skupinou na univerzitě Oxford Brookes[4] vytvořili nástroj The Automatic Labelling Environment [26], který provádí sémantickou segmentaci založenou na rozpoznání objektů ve scéně a získání jejich hloubek. Základem ALE je metoda používající P^N model, která každý pixel přiřazuje ke konkrétnímu objektu. Kód umožňuje trénovat vlastní kasifikátory na vlastních datech či předtrénovaných klasifikátorech. V rámci [6] a [33] bylo však zjištěno, že v porovnání s metodami [4], [11] a [28] je tento nástroj je pomalý a dosahuje menší přesnosti než konkurence.

2.1.3 Fully Convolutional Neural Networks (FCNs)

Sémantická segmentace s pomocí Fully Convolutional Neural Networks [28] je implementována pomocí trénovaných end-to-end, pixels-to-pixels plně konvolučních neuronových sítí (viz. obr. 2.3). Přístup však používá plně konvoluční sítě, aby mohl zpracovat obrázek původní velikosti a produkovat výstup korespondující velikosti s efektivním odvozením a učením. Je zde využíváno před-trénovaných modelů s pomocí trénování s učitelem. Autoři přizpůsobili stávající moderní klasifikační sítě[5] na plně konvoluční sítě a přenesli jejich reprezentaci učení na segmentační úkol.



Obrázek 2.3: FCNs se mohou efektivně učit pro vytváření hustých predikcí pro *per-pixel* úkoly jako je sémantická segmentace.

převzato z [28]

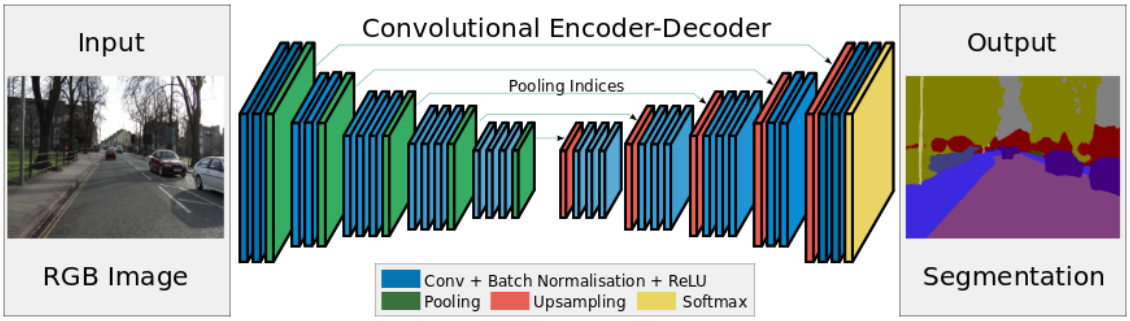
[4]The Oxford Brookes University Computer Vision (http://cms.brookes.ac.uk/research/visiongroup/)
[5]AlexNet, the VGG net, GoogLeNet

Jádrem je architektura s možností přeskočit vrstvu, která kombinuje sémantické informace z hlubokých hrubých vrstev s výskytem informací v mělkých jemných vrstvách pro produkování přesných a detailních segmentací. Sítě využívají vrstev nadvzorkování, kde probíhá pixelová predikce. A vrstev podvzorkování, kde probíhá učení. Plně konvoluční neuronové sítě byly testovány na datových sadách[6], na kterých dosahovaly state-of-the-art výsledků pro sémantickou segmentaci.

2.1.4 SegNet

Architektura SegNet [11] byla vytvořena skupinou Computer Vision and Robotics[7] na Univesity of Cambridge a provádí sémantickou segmentaci a je založena na hluboké plně konvoluční neuronové síti. Segnet obsahuje sekvenci nelineárních vrstev (kodérů a dekodérů) společně s natrénovaným pixelovým klasifikátorem. Schéma těchto vrstev je znázorněno na obrázku 2.5. Kodéry se skládají ze třinácti konvolučních vrstev navržených pro objektovou klasifikaci. Inicializace trénovacího procesu může proběhnout s pomocí váh trénovaných pro klasifikátor z objemných datasetů.

Každá vrstva kodéru má korespondující vrstvu v dekodéru. Téměř jako ve všech konvolučních neuronových sítích je v poslední vrstvě dekodéru výstup odeslán do vícestupňového „soft-max“ klasifikátoru, který predikuje pravděpodobnost nezávisle pro každý pixel. V každé vrstvě sítě kodéru probíhá konvoluce s bankou filtrů, která produkuje množinu „feature“ map. Tyto mapy jsou dále dávkově normalizovány a upraveny pomocí ReLU (Rectified-linear non-linearity). Na konci každé vrstvy kodéru je vrstva poskytující max-pooling jehož výsledek je podvzorkován. Ve vrstvách dekodéru jsou vždy vstupní „feature“ mapy nejdříve nadvzorkovány. Následují podvrstvy konvoluce, normalizace a ReLU.



Obrázek 2.5: Schéma SegNet
převzato z [11]

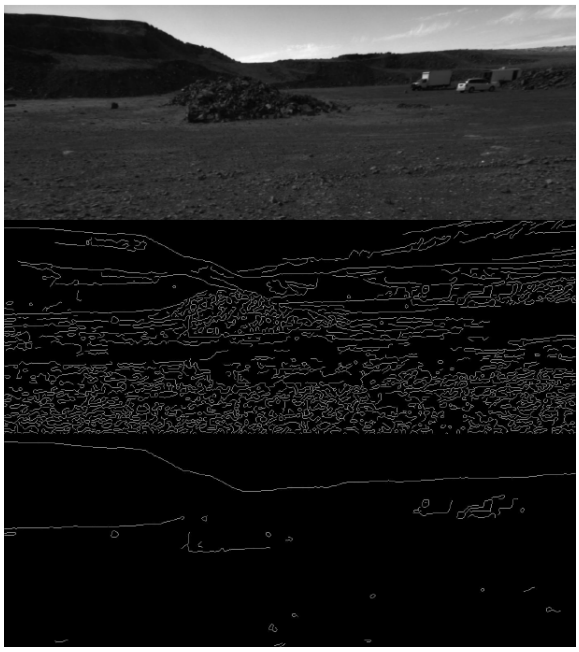
2.1.5 Machine Learning Approach to Horizon Line Detection

V Machine Learning Approach to Horizon Line Detection [5] je také využito strojového učení s učitelem, a to pro detekci křivky horizontu s pomocí hran v obrázku a lokálních vlastností. Na daný šedotónový obrázek je nejdříve použit Cannyho detektor hran [14]

[6] PASCAL VOC, NYUDv2, SIFT Flow
[7] Machine Intelligence Laboratory, University of Cambridge Department of Engineering (http://mi.eng.cam.ac.uk/Main/CVR)

s použitím různých parametrů, ale ponechány jsou pouze hrany, které projdou filtrací rozsáhlého prahování (wide range thresholding [21]). Takto vyfiltrované hrany jsou označeny jako MSEEs (Maximally Stable Extremal Edges). Postup získání MSEE hran je znázorněn na obrázku 2.7. S použitím informací z ground truth je natrénován klasifikátor pomocí SVM pro klasifikaci MSEE do pixelů dvou kategorií - horizont a ostatní. Každý MSEE pixel a jeho okolí je popsáno pomocí SIFT deskriptorů [29], které jsou vstupem pro SVM klasifikátor. Horizontové MSEE lokace jsou vybrány pro každý čtvrtý pixel křivky horizontu. Pro nehorizontové lokace je vybrán náhodně stejný počet z nehorizontových oblastí obrázku. Pro každý MSEE pixel je extrahován 128 dimenzionální vektor z 16×16 px čtvercové oblasti okolo tohoto pixelu.

Pro predikci je použit stejný postup a každý pixel je tak predikován zda-li je součástí křivky horizontu nebo ne. Pozitivně predikované pixely jsou použity pro dynamické programování, kde algoritmus hledá nejkratší cestu, která vrací konzistentní křivku horizontu. Dynamické programování vrací různé výsledky pro hledání zleva do prava nebo naopak. Z toho důvodu je výsledná křivka kompozicí křivek z obou směrů.



Obrázek 2.7: (top) Šedotónový vstupní obrázek. (middle) Výstup z Cannyho detektoru hran. (bottom) Extrahované MSEE.

převzato z [5]

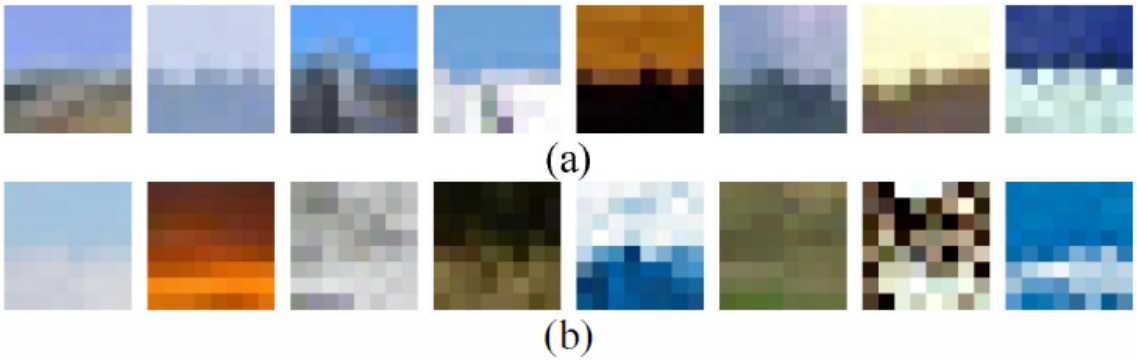
2.1.6 Skyline Localization for Mountain Images

Metoda Skyline Localization for Mountain Images [24] je velmi podobná předchozí metodě [5]. Metoda je založená na určení horizontu s využitím hranové extrakce, strojového učení s učitelem a dynamickém programování. Obrázek je nejdříve převeden do odstínů šedé a jsou z něj extrahovány hrany za pomocí Cannyho algoritmu [14]. Detekované hrany jsou oproti [5] rozděleny na hranové segmenty, které jsou definované jako spojité úseky o velikosti 8 px. Pro každý hranový segment je vybrána čtvercová oblast 8×8 px (narozdíl od

16×16 px oblasti použité v [5]). Vektory pro trénování jsou zde vytvářeny pro celý hranový segment, zatímco v [5] jsou vytvářeny vektory pro každý pixel zvlášť.

Pro trénování klasifikátoru se používá 210 dimenzionální vektor, který se skládá:

- RGB hodnoty každého pixelu oblasti hranového segmentu ($3 * 8 * 8$)
- Průměrné hodnoty jasu horní a dolní poloviny oblasti hranového segmentu (2)
- Pozice X a Y každého pixelu hranového segmentu ($2 * 8$)



Obrázek 2.9: (a) Pozitivní trénovací vzorky. (b) Negativní trénovací vzorky.
převzato z [24]

Klasifikátor je natrénován s pomocí těchto vektorů pro každý hranový segment. Vektory segmentů křivky horizontu jsou použity všechny a jsou získány s pomocí ground truth. Negativní vektory jsou vybrány náhodným způsobem ve stejném počtu jako pozitivní vektory. Oblasti pozitivních a negativních hranových segmentů jsou znázorněny na obrázku 2.9.

Pro predikci je využíván stejný postup získání vektoru, kde SVM predikuje zda-li je hranový segment pozitivní/negativní. Vrácené pozitivní segmenty jsou využity pro dynamické programování. S pomocí energetické funkce je vytvořena mapa energií pro každý pixel, ve kterém jsou pozitivní segmenty označeny hodnotou 0. V této mapě je pak hledána nejkratší cesta s co nejnižší kumulativní energií.

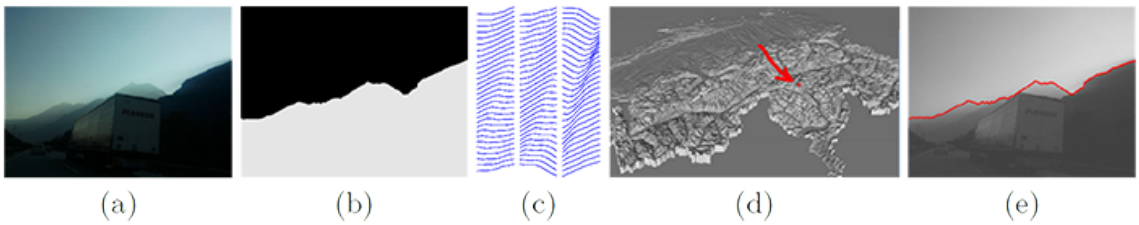
2.2 Rešerše přístupů pro vizuální geo-lokalizaci

Vizuální geo-lokalizace je netriviální problém kvůli různorodým podmínkám zahrnujícím například proměnlivou vegetaci, počasí, sluneční osvětlení nebo roční období. Narozdíl od vizuální geo-lokalizace v urbanistickém prostředí, kde budovy většinou svou vizuální stránku často nemění, je použití vizuálně založených metod mnohem náročnější právě v přírodním prostředí. Vizuální geo-lokalizace je navíc velmi užitečný nástroj pro historické a forenzní vědní obory, inteligentní aplikace, dokumentační techniky, organizování fotografických kolekcí nebo vyhledávání v archivech. Většina přístupů pro vizuální geo-lokalizaci používá datové sady s digitálními 3D modely a fotografiemi s již známými informacemi o zeměpisných souřadnicích a úhly pořízení fotografie.

Následující podkapitoly popisují algoritmické přístupy [8], [27], [39] pro vizuální geo-lokalizaci v přírodním prostředí. Dále je zde zmíněn projekt Locate [13], který vychází z [10] a implementuje několik algoritmů (např. [39]).

2.2.1 Large Scale Visual Geo-localization of Images in Mountainous Terrain

Přístup popsaný v článku Large Scale Visual Geo-localization of Images in Moutainous Terrain [8] se zaměřuje na kamerovou geo-lokalizaci. Využívá se rozpoznání křivky horizontu v lokalizovaném obrázku pro daný digitální elevační model (DEM[8]) dané země. Agregují se zde informace o tvaru napříč celou křivkou horizontu a hledá se podobná konfigurace základních tvarů v rozsáhlé databázi, která je organizována pro možnost vyhledávání různých úhlů pohledu - FOV[9]. Různé stavy celého procesu jsou znázorněny na obr. 2.11. Tento článek byl dále rozšířen v Image Base Geo-localization in the Alps [39], kde autoři uvádějí více detailnější analýzu a evaluaci systému a vylepšují algoritmus v segmentaci oblohy. Obsahuje novou metodu pro robustní kódování kontur jako dvě různé hlasovací schémata. První schéma operuje pouze v prostoru deskriptorů zatímco druhý kombinuje hlasování v prostoru deskriptorů a rotací.



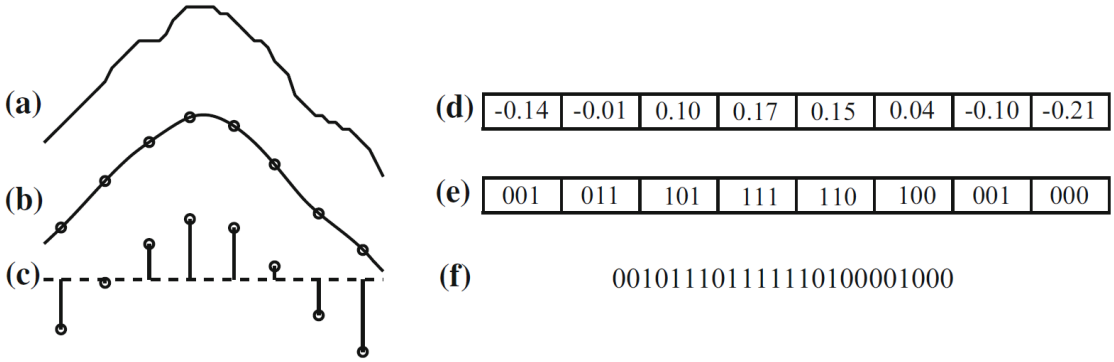
Obrázek 2.11: (a) Lokalizovaný obrázek. (b) Segmentace oblohy. (c) Vzorek množiny extrahovaných 10° contourletů. (d) Rozpoznaná geo-lokalizace v DEM. (e) Překryv viditelného horizontu odhadnuté pozice.

převzato z [8]

Problém rozpoznání lokace je v obecném rámci šesti-dimenzionální kvůli zpracování tří dimenzí pozice a tří dimenzí orientace. V tomto článku se předpokládá, že k pořízení fotografie došlo v malé výšce nad povrchem země. Využívá se vlastností viditelného horizontu. Viditelný horizont digitálního elevačního modelu je extrahován ve všech pozicích (360° v každé pozici) a reprezentován kolekcí vektorově kvantifikovaných lokálních *contourletů*[10]. Subsekvenčně je extrahovaný obrys robustně popsán množinou lokálních contourletů s relativní úhlovou vzdáleností α_q s ohledem na optické osy kamery. Následně je využíván invertovaný souborový systém pro obrysová slova pro nalezení nejlepších lokací a postupně se hlasuje pro různé směry pohledu. To je provedeno v integrované geometrické verifikaci během *bag-of-words*[11] hledání.

Zpracování obrázku se skládá z nalezení křivky horizontu, což je zde chápáno jako extrakce popředí / pozadí. Hledá se nejvyšší pixel popředí a počítá se suma všech cen pixelů popředí. Dále pak všechny ceny nad kandidátem křivky horizontu. Segmentace je pak popsána energetickou funkcí. Algoritmus je automatický, ale umožňuje i zásah uživatele pro manuální označení popředí. Dalším krokem je extrakce contourletů (viz. obr. 2.13). Kontury jsou překrývající se křivky šířky w. Tyto křivky jsou navzorkovány s pevným odstupem

[8] Digital Elevation Model
[9] Field of View
[10] Contourlets: obrysová slova, podobná v duchu vizuálních slov získaných z kvantifikování obrázku plošnými deskriptory.
[11] Bag-of-words: je v počítačovém vidění vektor počtu výskytů slovníku lokálních vlastností obrázku.



Obrázek 2.13: Výpočet konturových slov: (a) Základní kontura. (b) Vyhlazená kontura s n vzorkovacími body. (c) Navzorkované body po normalizaci. (d) Contourlety jako numerické vektory. (e) Každá dimenze kvantifikovaná do tří bitů. (f) Konturové slovo jako 24-bitové celé číslo.

převzato z [39]

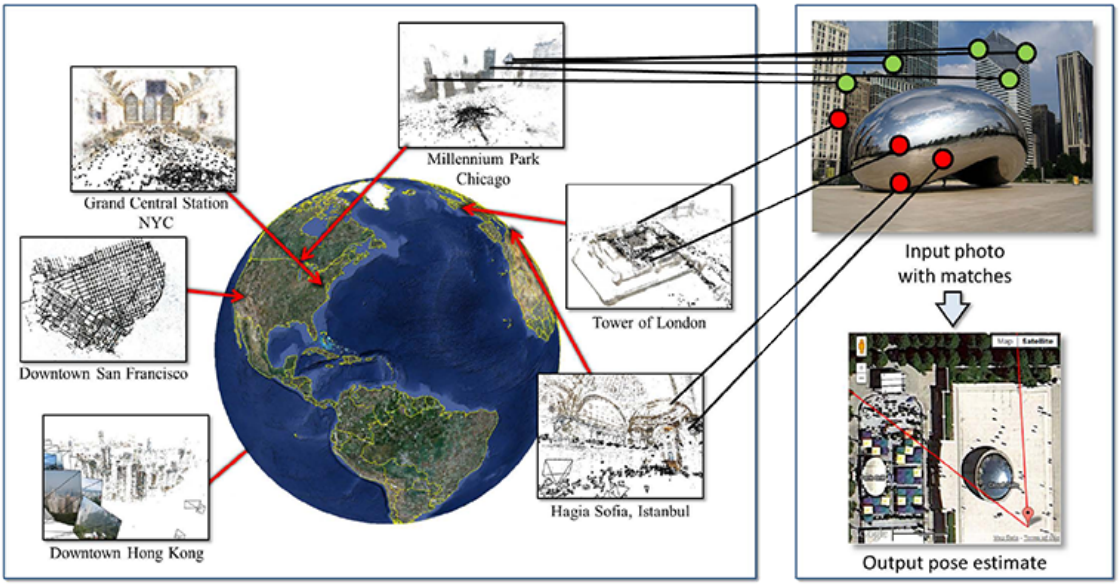
a tím jsou získány n-dimenzionální vektory hodnot. Dále se přímo kvantifikuje separátně každá dimenze deskriptoru. Navíc je nalezen nejlepší *bin*[12]. Z každé dimenze je získán bin, ze kterých se pak konkatenací získá jedno celé číslo - konturové slovo. Jedna featura (contourleta) se kóduje na n bitů. Počet featur je pak roven 2^n. Při vyhledávání se používají invertované soubory a celý proces je ještě rozšířen o hrubou geometrickou verifikaci. Celková velikost invertovaného indexu je dána velikostí a hustotou vzorkování prohledávané geografické oblasti. Pro každé slovo je vytvořen seznam, která uchovává ID panoramatu a azimut α_d pro každý výskyt contourletu. Algoritmus porovnává 10°–70° pohledy s 360° panoramaty. V ideálním případě je skóre 0 získána z databázového obrázku, který obsahuje každé konturové slovo nejméně tak často jako lokalizovaný obrázek. Během hlasovací fáze se získá geometrická informace, která se dále použije pro geometrickou verifikaci po obdržení 1000 nejlepších kandidátů. Geometrická verifikace se skládá z výpočtu optimálního zarovnání dvou viditelných křivek horizontu použitím iterativních nejbližších bodů (ICP[13]) což určuje kompletní 3D rotaci. Průměrná chyba zarovnání je použita jako skóre pro nové seřazení kandidátů.

2.2.2 Worldwide Pose Estimation Using 3D Point Clouds

Worldwide Pose Estimation Using 3D Point Clouds [27] popisuje vizuální lokalizaci jako celosvětové odhadnutí pozice. V daném obrázku probíhá automatické určení matice kamery (pozice, orientace a kamerové hodnoty) v geo-označeném koordinačním systému. Práce se zaměřuje na precizní geometrii kamery, jejíž správný odhad může instantně poskytnout silná vodítka pro určení, která část obrázku může být obloha, silnice nebo budovy. Metoda pracuje převážně s lokalizací v urbanistickém prostředí, ale lze ji použít i na přírodních oblastech s dostatečným množstvím charakteristických a známých vizuálních bodů.

Metoda přímo zpracovává korespondenci mezi 2D vlastnostmi v obrázku a 3D body ve velkém mračnu bodů (point cloud) zahrnující mnoho míst na světě. Je využíváno 3D bodového shluku vytvořeného během procesu *Structure from Motion (SfM)*[14] nad obrázky se

[12] Binární číselná hodnota.
[13] Iterative Closest Points
[14] Technika fotogrametrického zobrazování rozsahu pro odhad 3D struktur z 2D sekvencí obrázků.



Obrázek 2.15: Celosvětová databáze bodových shluků. Obrázek je porovnáván s databází geo-označených struktur. Databáze (vlevo) zahrnuje obrázky po celém světě (zde je znázorněno jen několik bodových shluků). Vpravo je počítána pozice nového obrázku porovnáváním na tuto celosvětovou databázi. Přímé porovnávání vlastností je nepřesné a může produkovat špatné výsledky (červené body). Proto jsou v tomto článku použity robustnější techniky.

převzato z [27]

zeměpisnými souřadnicemi. Hlavní částí je škálovatelná metoda pro přesné získání 3D pozice kamery z jedné fotografie pořízené z neznámé lokace. Dalším úkolem je nalezení dobrých korespondencí mezi vlastnostmi obrázku v masivní databázi 3D bodů (viz. obr. 2.15).

2.2.3 Locate - Visual Localization in Natural Environments

Locate [13] je geo-lokalizační projekt, který (mimo jiné) implementuje algoritmus [39]. Cílem je lokalizovat fotografii a určit tak její pozici a orientaci pořizující kamery. Je tedy využito digitálních elevačních modelů, které jsou v dobrém rozlišení dostupné téměř pro celou planetu. Metoda je však extrémně výpočetně náročná kvůli velmi rozsáhlému prostoru, který se prohledává. Systém byl natrénován na prostor švýcarských Alp. Ilustrace správné geo-lokalizace fotografie je znázorněna na obr. 2.17.

2.3 Rešerše algoritmů pro inverzní indexy

Přístupy pro vizuální geo-lokalizaci uvedené v kapitole 2.2 jsou velmi paměťově náročné z důvodu velkého množství dat reprezentujících například známé horizonty. Tato data jsou reprezentována formou binů extrahovaných z detekovaných contourletů (viz. 2.2.1). V praxi se pak obdržený horizont segmentuje na několik částí[15], které představují contourlety. Každý je pak zpracován na biny, a každý z těchto binů je dále použit pro vyhledání známých dokumentů (horizontů), ve kterých se daný contourlet nachází. Výsledky ve formě

[15] Features

Obrázek 2.17: Vizualizace geo-lokalizace
převzato z [13]

dokumentů jsou pak vráceny v závislosti na počtu obsahujících contourletů hledaného horizontu.

Pro zmíněné vyhledávání dokumentů se využívá invertovaných indexů. Jednotlivé známé contourlety jsou uloženy v paměti a ke každému vždy přísluší seznam dokumentů, ve kterých se daný contourlet nalézá. V rámci naší implementace algoritmu [39] je pro možnost vyhledávání nahrán do paměti celý invertovaný index. Tento invertovaný index může obsahovat velké množství featur (contourletů) a dokumentů. V případě naší implementace se jedna featura zakóduje do 24 bitů, takže počet featur je 2^{24}. Pro implementaci [39], zabírá invertovaný index pro plochu přibližně 1/3 Alp, v paměti prostor o velikosti zhruba 120 GB. To má kritický dopad na výpočetní nároky hardwaru, na kterém systém poběží. V tomto případě by bylo zcela nereálné, že by mohl být systém dále škálován na celé Alpy, kompletní pohoří Evropy nebo pro celý svět.

Tuto problematiku je třeba vyřešit vhodným přístupem organizace a řízení paměti. Funkcionalita invertovaného indexu by měla zůstat zachována, protože je pro tento typ vyhledání ideální. Musel jsem však nalézt optimální řešení, které by se postaralo o správu indexu v paměti. V nejlepším případě by knihovna měla rezervovat pro index jen malou omezenou oblast paměti, ve kterých by zanechával například jen 100 posledních hledaných contourletů nebo 100 nejčastěji hledaných. Pokud by v paměti daný contourlet nebyl proběhlo by načtení contourletu z disku do paměti a vhodným stránkovacím algoritmem by nahradil jiný v paměti. Případné vlastní řízení procesů mezi pamětí a diskem konkrétní knihovnou není překážkou, pokud splní podmínku jen omezeného množství používané paměti pro index.

2.3.1 HDF5

Knihovna HDF5 [44] poskytuje možnost pro práci s různorodými datovými modely a komplexními datovými objekty s podporou metadat. Nabízí platformně nezávislý datový formát, který nemá žádné limity na velikost obsažených dat. Knihovna nabízí API v jazycích C, C++, Java a Fortran 90. Výhodou je velké množství vlastních implementovaných metod, které dovolují optimalizace přístupového času a úložného prostoru na disku. HDF5 je podobný jako XML v tom smyslu, že HDF5 soubory popisují svou strukturu a umožňují uživateli specifikovat vztahy a závislosti komplexních dat. Na rozdíl od XML může HDF5 pracovat i s binárními daty a umožňuje přímý přístup k částem binárního souboru.

HDF5 se skládá:

- Souborový formát pro uložení HDF5 dat.
- Datový model pro logickou organizaci a přístup k HDF5 datům z aplikace.
- Software (knihovny, jazyková rozhraní a nástroje) pro práci s formátem.

Základním objektem HDF5 datového modelu jsou datasety a skupiny (groups). Skupiny seskupují více datasetů. Datasety jsou multidimenzionální pole datových elementů společně s metadaty. Vytvoření a použití datasetu vyžaduje otevření HDF5 souboru, kde je dataset fyzicky uložen. Dále je třeba definovat souborový a paměťový datový prostor (dataspace). Až když je veškeré zázemí připraveno je možno dataset načíst ze souboru. Knihovna umožňuje mít různé velikosti datového prostoru pro soubor i paměť. Je tedy možné mít definován datový prostor pro celý soubor a v paměti jen omezenou velikost. Problém však nastává při používání knihovny. Z daného datasetu je zapotřebí číst pouze úseky dat[16]. Tuto možnost samozřejmě knihovna umožňuje.

Pro čtení všech dat na jednom konkrétním indexu je zapotřebí mít alokovaný prostor pro úplně celý dataset. Vybraná data v alokovaném poli budou k dispozici na daném indexu, zbytek hodnot bude vyplněn nulami. Povinnost mít však v paměti vymezený blok pro kompletní invertovaný index zůstává. Tato vlastnost knihovny ji však činí nepoužitelnou pro potřeby uchovávat v paměti jen omezenou část invertovaného indexu.

2.3.2 MeTA: ModErn Text Analysis

MeTA: ModErn Text Analysis [31] je knihovnou zaměřující se na práci s textovým korpusem. Tato modulární knihovna nabízí tokenizaci textu včetně funkcí jako například *parse trees*[17]. MeTA využívá invertovaných a dopředných indexů s kompresí a různými cachovacími strategiemi. V knihovně je dostupná kolekce rankovacích funkcí pro hledání v indexu.

Všechna zpracovaná data jsou uložena v diskovém indexu. Každá MeTA aplikace má na vstupu index a všechna zpracovávaná data jsou zaměnitelná mezi všemi komponentami. V hlavním konfiguračním souboru se definuje typ indexační strategie a maximální velikost prostoru v RAM paměti. To umožňuje definovat jen omezenou a přesně definovanou velikost obsazené paměti indexem. Konfigurace obsahuje také název datasetu a korpusu. Název datasetu se použije jako prefix pro hledání korpusových dat. Definovaný korpus se použije pro cestu ke konfiguračnímu souboru korpusu. Při indexaci jsou nejdříve spuštěny tokenizéry, které definují jak segmentovat vstupní soubor na tokeny. Následují filtry, které mohou být zřetězeny. Filtry definují jakým způsobem bude text modifikován nebo transformován. Posledním článkem ve zpracování jsou analyzátory, které operují nad výstupem filtrů a produkují počty výskytů tokenů v dokumentu.

Knihovna však neumožňuje dynamické změny indexu za běhu. Dále knihovna vyžaduje použití rankeru[18] definovaného v rámci knihovny. Rankovací funkce v naší implementaci již implementovaná je a její přesunutí do třídy knihovny by bylo velmi náročné. Knihovna je navíc určená pro zpracování přirozeného jazyka. Její adaptování na zpracování konturových

[16] Ve dvoudimenzionálním prostoru pro invertovaný index chceme číst pouze konkrétní řádek.

[17] Parse tree nebo také derivační strom je seřazený kořenový strom, který reprezentuje syntaktickou strukturu řetězců na základě bezkontextové gramatiky.

[18] Funkce, která hodnotí výsledky dotazů v invertovaném indexu. Ranker vrací seznam dokumentů seřazených dle relevance.

slov by znamenala nutnost, vyhradit si velkou porci času na experimentování s knihovnou a nalezení funkčního řešení. Navíc není jisté, že by adaptace byla vůbec možná.

2.3.3 MIFLUZ

Knihovna MIFLUZ [17] je určena pro používání a řízení fulltextových invertovaných indexů. MIFLUZ umožňuje dynamické aktualizace indexu a podporuje škálování (až na úroveň 1 TB indexů). V knihovně je zajištěno řízení a kontrolování množství alokované paměti a sdílení souborů indexu a cache paměti napříč procesy a vlákny. Velkou výhodou je implicitní komprese indexu až na 50 % objemu surových dat. Struktura indexu je konfigurovatelná za běhu a umožňuje inkluzi relevantních rankovacích informací. Funkce pro dotazy nevyžadují nahrání všech výskytů hledaného termu.

V prvním kroce se musí vytvořit textový indexační algoritmus pro spočítání počtu výskytů slov v textu. Toho lze dosáhnout determinováním datového modelu a pravidel pro filtrování znaků. Dalším krokem je algoritmus pro segmentaci textu. MIFLUZ potřebuje datový model pro uložení dat. Prvním elementem datového modelu je slovo, které je uloženo ve slovníku, a které je možno vyhledávat. V posledním kroce je třeba definovat prostředí indexu a vyhledávací funkci.

Knihovna však sdílí problémy s knihovnou MeTA. Další nevýhodou může být velmi slabá a omezená dokumentace nebo téměř žádné oficiální tutoriály či návody k použití. Pro neznámé uživatele tedy může být překážkou strmá křivka učení při použití této knihovny.

2.4 Rešerše služeb pro vizualizaci geografických dat

Jedním z hlavních problémů specifikovaných v kapitole 1.1.1 byla špatná vizualizace výsledků nalezených horizontů. V práci [34] byly pak výsledky většinou špatně pochopeny uživateli. Tento problém bylo třeba eliminovat novou vizualizací výsledků, která bude maximálně jednoduchá a pochopitelná uživateli. V této kapitole je rešerše zaměřena na poslední skupinu algoritmů, kterými jsou vizualizační knihovny a nástroje. Analýzou dostupných nástrojů, schopných vizualizovat geografická data, můžeme zjistit, který nástroj by měl být pro použití nejvhodnější. Jedním z hlavních kritérií pro výběr, aby služba běžela ideálně na serverech jejich autorů. Virtuální server, na kterém webová aplikace poběží nemá k dispozici fyzické GPU, a proto jsou z rešerše vyloučeny vizualizační nástroje[19], které by jinak mohly být nainstalovány na serveru a mohly by provádět renderování výsledků v reálném čase.

2.4.1 Google Maps Platform

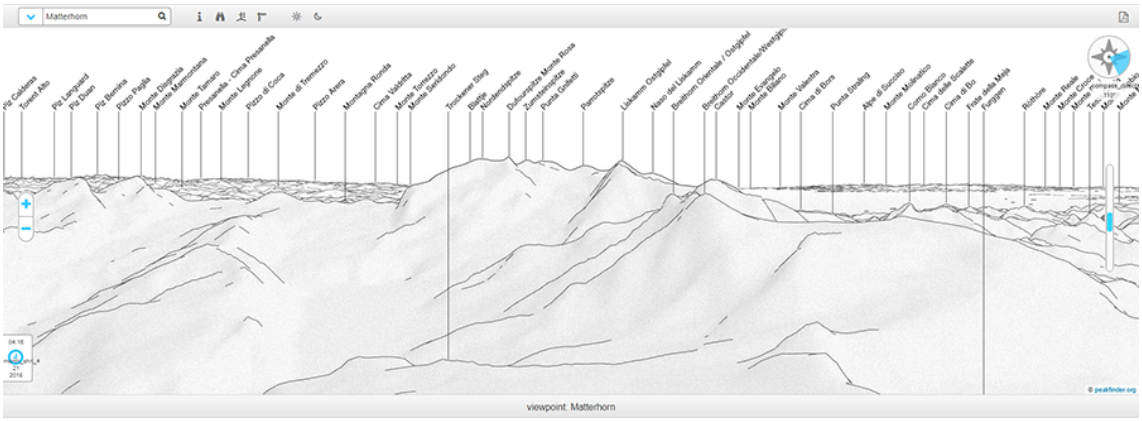
Google Maps [3] od společnosti Google je k dispozici již velmi dlouho a je inovován každým rokem. Disponuje velmi spolehlivou službou s rozsáhlým a robustním API. Pro všechny své služby poskytuje také velkou řadu tutoriálů a návodů včetně plné dokumentace. Své služby nabízí ve většině použití zdarma. Využitelnou službou Google Map je JavaScriptové rozhraní díky čemuž lze rychle a pohotově zavést služby map do jakékoliv webové aplikace. Stačí jen připojit odkaz na knihovnu Google Maps JS a v JavaScriptovém kódu aplikace již jen definovat mapu, její typ, pozici, přiblížení a případně zadefinovat ukazatel polohy, který se zobrazí uprostřed mapy. Google Mapy umožňují zobrazit klasické orientační mapy, ale také satelitní snímky nebo i Google Street View (pokud je pro danou k dispozici). Google

[19] Např.: Virtual Terrain Project (http://vterrain.org/), AutoDEM (http://www.autodem.com/).

Maps Platform by tedy měl být více než nápomocný ke vhodné vizualizaci výsledků vizuální geo-lokalizace.

2.4.2 PeakFinder

PeakFinder [40] od autora Fabio Soldati je aplikace pro vyhledávání panoramat, která zobrazuje panoramata hornatých oblastí a vykresluje k vrcholkům jejich názvy. V aplikaci je možné si vyhledat konkrétní místo podle názvu nebo souřadnic. Z nalezeného místa lze pak dokola procházet kompletní horizont (viz. obr. 2.19). Aplikace také umožňuje vykreslit trasu slunce a měsíce v čase. PeakFinder funguje primárně jako webová aplikace, ale je již také k dispozici jako mobilní aplikace s možností augmentované reality. V rámci této aplikace je definováno malé API poskytující základní službu aplikace. V rámci API je však možné ovlivnit pozici zobrazeného panoramatu, nadmořskou výšku pohledu, přiblížení nebo azimut. Nástroj je možné zavést do aplikace například pouhým vložením HTML tagu <iframe>, kde se v URL adrese definují všechny parametry. PeakFinder je rovněž velmi silným kandidátem pro případné vizualizace pozic na základě zeměpisných souřadnic.



Obrázek 2.19: Ukázka aplikace PeakFinder.

2.4.3 CesiumJS

CesiumJS [43] je open-source JavaScriptová knihovna pro vizualizace map, 3D objektů, modelů a časově dynamických vizualizací apod. CesiumJS využívá služeb Google Earth a dokáže tak vizualizovat i celou planetu Zemi. Nevýhodou této knihovny může být její velmi vysoké nároky na výpočetní výkon a internetové připojení. Samotná manipulace s ukázkou mapy na webových stránkách aplikace je velmi kostrbatá, trhaná a relativně pomalé načítání textur může být pro běžného uživatele značně nepřívětivé. Tato knihovna nabízí širokou škálu použití, včetně definovaného API. Prerekvizitou pro spuštění aplikace je však zprovoznění vlastní instance Cesium webového serveru poskytující výpočetní výkon. To může činit pro naše potřeby potíže, protože virtuální server příliš výkonu nabídnout nemůže. Celkově je CesiumJS velmi bohatá knihovna, avšak nesplňuje podmínky maximální jednoduchosti a přívětivosti pro uživatele.

Kapitola 3

Implementace detekce křivky horizontu

V této kapitole je detailní popis implementace algoritmu pro detekci křivky horizontu v obrázku. Z provedené rešerše algoritmů z kapitoly 2.1 jsem získal 7 kandidátů, ze kterých bylo třeba vybrat nejvhodnější algoritmus. Kandidáti [4], [11], [26], [28] byli již implementováni ve srovnávací publikaci [6]. V rámci [6] byly všechny tyto metody trénovány na datové sadě CH1 [9] a testovány na datové sadě GeoPose3K [12]. Metody [11], [26], [28] byly také implementovány v rámci diplomové práce [33] Ing. Jakuba Pelikána zabývající se sémantickou segmentací v horském prostředí. Práce taktéž obsahuje část, ve které jednotlivé algoritmy srovnává na stejných datových sadách jako [6]. Pro eliminaci problémů předchozích jmenovaných algoritmů jsem se rozhodl pro implementaci algoritmu Skyline Localization for Mountain Images [24] jejímž autorem je Yao-Ling Hung a další. Implementovanou metodu mohu natrénovat i testovat na stejných datových sadách [9], [12] a výsledky testování přímo porovnávat s výsledky z prací [6] a [33].

Implementace algoritmu se skládá z popisu struktury aplikace, ve které bude vybraný algoritmus implementován. Dále následuje podrobný teoretický popis algoritmu, na který následují podkapitoly s již konkrétními informacemi o realizaci implementace. V rámci implementace jsou zde uvedeny experimenty při výběru různých parametrů pro Cannyho detektor hran a pro trénování modelu pomocí SVM. Součástí kapitoly jsou také experimenty s finální implementací algoritmu. Následující podkapitola porovnává naměřené výsledky s pracemi [6] a [33]. Kapitola je uzavřena profilací algoritmu a optimalizací z hlediska času.

3.1 Aplikace lokalizace křivky horizontu

Aplikace běží ve dvou režimech: trénování a lokalizace. Režim trénování je určen primárně pro zpracování obrázků a jejich ground truth a získání tak „Feature vektorů“[1], a k němu korespondující trénovací vektor[2]. Režim lokalizace je připraven pro přímé použití detekce křivky horizontu.

[1] Vektory s informacemi o jednotlivých hranových segmentech a jejich oblastech (viz. kap. 2.1.6).
[2] S označením pozitivních/negativních Feature vektorů (viz. kap. 3.5).

3.2 Extrakce hran

Základem pro algoritmus extrakce křivky horizontu je extrakce hran z obrázku a získání hranové mapy. Pro extrakci hran je v algoritmu použit Cannyho hranový detektor [14], který je založený na aproximování maxim prvních derivací a detekuje hrany ve dvourozměrném diskrétním obraze. Pro detekci hran jsem použil implementovanou verzi Cannyho hranového detektoru z knihovny OpenCV [2]. Ovlivňováním parametrů[3] Cannyho detektoru lze získávat různě husté hranové výstupy.

3.2.1 Experiment s hodnotami prahování

S hodnotami prahování jsem provedl experiment, kde jsem měnil hodnoty prahování. Cílem experimentu bylo zjistit optimální hodnotu práhu, aby obrázek s detekovanými hranami obsahoval co největší počet hran náležících křivce horizontu. A zároveň aby tento obrázek neobsahoval zbytečně příliš velký počet hran, které křivce horizontu nenáleží. Výsledky byly subjektivně hodnoceny vizuálním zhodnocením. Kritériem pro nejlepší výsledky byla vlastnost, aby křivka horizontu byla ideálně kompletní. Avšak zároveň aby křivek, které nepatří horizontu, bylo co nejméně (aby v obrázku bylo co nejméně *vaty*). Experiment byl proveden na fotografiích z datové sady GeoPose3K [12]. Vizuální hodnocení však proběhlo pouze na přibližně 50 fotografií. Více obrázků nebylo třeba vizuálně měřit z důvodu opakujících se výsledků, které nepřinášely žádné nové informace.

Experiment byl proveden na různé hodnoty prahování[4]. V ukázce výsledků 3.1 je možné vidět původní obrázky převedené do odstínů šedi a dále výsledky po prahování. Pro nízkou hodnotu prahování (≤ 15) je detekovaných hran příliš mnoho a jejich zpracování by stálo zbytečně mnoho výpočetních i paměťových prostředků. Při velkých hodnotách prahování ($45 \geq$) je počet detekovaných křivek optimální, ale v častých případech na úkor hran křivky horizontu. Při těchto hodnotách již chyběly i celé kusy hran křivky horizontu, což by mělo kritický dopad na úspěšnost celého algoritmu lokalizace křivky horizontu. Při hodnotě prahování (30) byl počet nadbytečných křivek v některých případech poměrně vysoký, ale na druhou stranu hrany křivky horizontu chyběly pouze v minimální míře. Mohu proto tvrdit, že je tato hodnota prahování nelepší variantou.

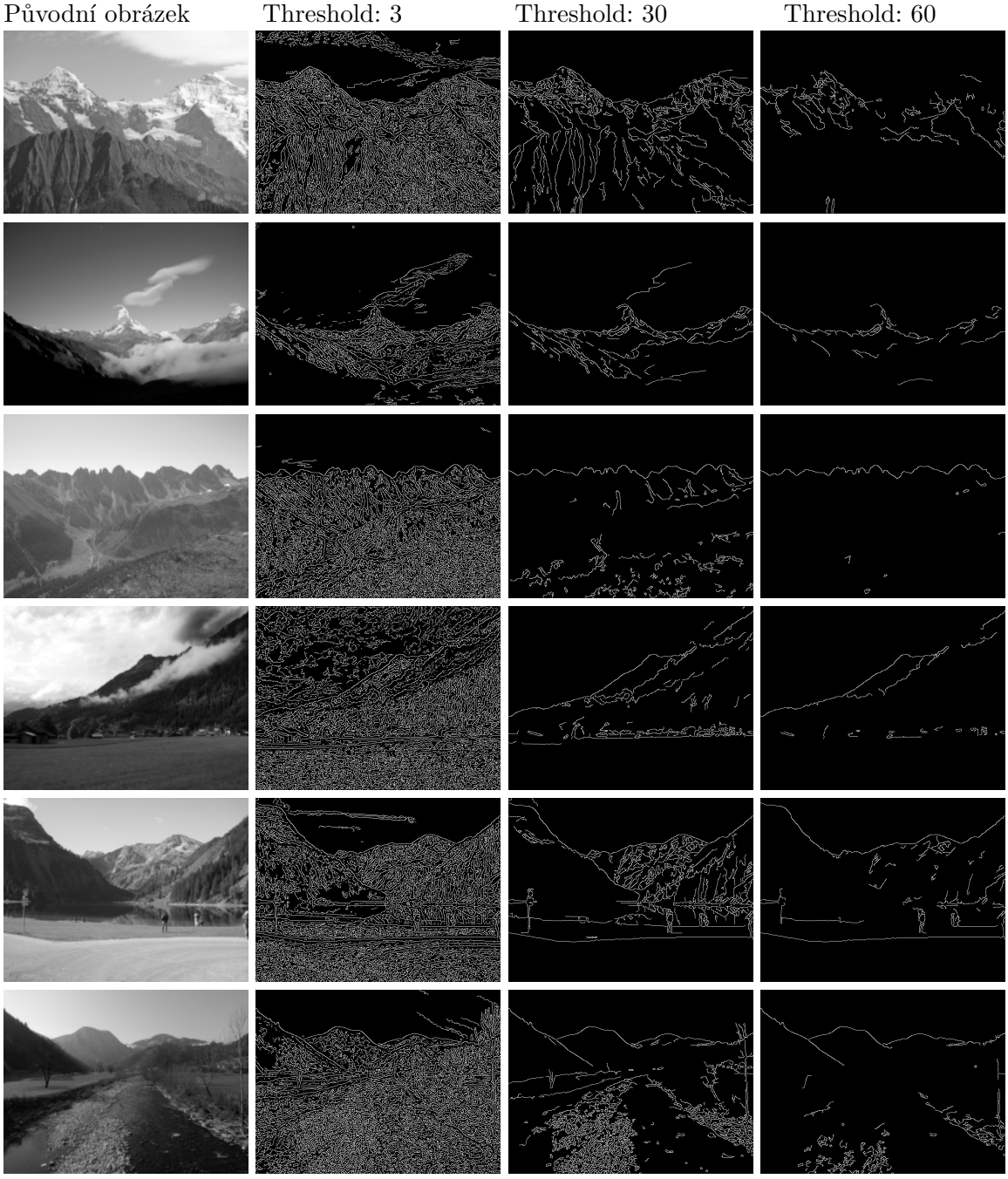
3.2.2 Extrakce hranových segmentů

Po získání hran z obrázku je třeba hrany rozdělit na úseky pevně dané velikosti, se kterými se bude nadále pracovat. Jednotlivé úseky jsou v algoritmu [24] získány o velikosti 8 px a je jim přiřazen název *hranové segmenty*. Tyto hranové segmenty jsou získány algoritmem, kdy je vždy nejlevější obrazový bod získané hrany z Cannyho detektoru hran označen jako počáteční pixel. Od tohoto pixelu se algoritmus pokouší najít přímého následníka hranového segmentu na pěti předdefinovaných pozicích[5]. Pokud je takový pixel nalezen, pak je počáteční pixel uložen a hledá se další následník od právě nalezeného. Tento proces hledání sousedů se opakuje dokud není takto nalezený spojitý úsek dlouhý 8 px. V tom případě je extrakce hranového segmentu kompletní a dotyčné pixely jsou vymazány z hranové mapy, aby se tak odstranil možný výskyt duplicit nebo nedeterminismu. Pokud sousední pixel není nalezen na definovaných pozicích, pak je celý hranový segment zahozen. Proces se

[3] Velikost prahování pro histerezi.

[4] Hodnoty: 3, 15, 30, 45, 60.

[5] Možné pozice následníka: nad, vpravo nad, vpravo, vpravo pod, pod (aktuálním pixelem).



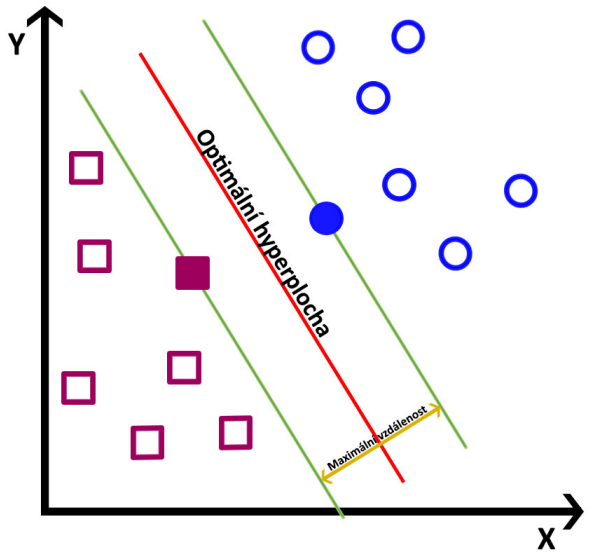
Obrázek 3.1: Hledání optimální hodnoty prahování.

opakuje tak dlouho, dokud v hranové mapě existují nějaké detekované hrany. Výstupem tohoto procesu je množina hranových segmentů.

3.3 Detekce segmentů horizontu

Vybraná metoda lokalizace křivky horizontu [24] využívá SVM - Support Vector Machines [42], což je metoda strojového učení s učitelem a je využívána pro klasifikaci[6] nebo regresní analýzu[7]. Základním kamenem SVM je lineární klasifikátor do dvou tříd. Cílem v této úloze je najít nadrovinu (viz. obr. 3.2), která rozdělí obě třídy trénovacích dat, kde hodnota minima vzdáleností bodů od nadroviny by měla být co největší.

V praxi je SVM využito k vytvoření modelu, který při trénování dělí vstupní data právě do dvou tříd. Vstupní data jsou proto rozdělována díky informacím, které vzorky jsou pozitivní nebo negativní. Vstupní data mohou být typicky multidimenzionální. Vstupem SVM je trénovací sada dat skládající se z vektorů hodnot. SVM pak vektor přiřadí k jedné či druhé skupině. Po každém vektoru si SVM upravuje nadrovinu tak, aby byla co nejoptimálnější (tedy aby vzdálenost obou skupin byl co největší - viz. obr. 3.2). Při predikci je testovaný vektor porovnán klasifikátorem s natrénovaným modelem a SVM rozhodne, do které skupině vektor odpovídá. Výstupem predikce je odpověď pozitivní (1) nebo negativní (-1).



Obrázek 3.2: Optimální nadrovina
inspirováno z [42]

Pro implementaci SVM do algoritmu jsem zvolil knihovnu LIBSVM [15]. LIBSVM je integrovaný software, pro support vector klasifikaci, regresi a distribuované odhady. Podporuje multidimenzionální klasifikaci. LIBSVM je implementováno do řady známých jazyků včetně Javy. Tato knihovna mimo jiné poskytuje velmi snadno použitelné nástroje pro kontrolu zda jsou trénovací data v pořádku, nástroj pro selekci optimálních parametrů a nástroj, který automaticky vybere nejlepší parametry a natrénuje spolu s testovacími daty a trénovacím vektorem finální model.

[6] Problém kde se určuje, do které z kategorií dat je dané pozorování přiřazeno.

[7] Statistické metody, pomocí kterých se odhaduje hodnota jisté náhodné veličiny na základě znalosti jiných veličin.

3.4 Extrakce Feature vektoru

Reprezentací jednotlivých vzorků trénovací sady dat je Feature vektor. Feature vektor je v našem konkrétním případě této metody 210 dimenzionální vektor hodnot, nesoucích informace o jednotlivých hranových segmentech. Význam jednotlivých dimenzí již byl vysvětlen v kapitole 2.1.6. Pro každý hranový segment se Feature vektor počítá pro jeho nejbližší okolí, kterým je 8×8 px čtvercová oblast, kterou hranový segment prochází. Prvních 192 dimenzí je rezervováno pro informace o barvách jednotlivých pixelů okolí hranového segmentu, kde se jako pro jednu dimenzi počítá každá barevná složka RGB modelu. Další 2 dimenze jsou určeny pro jasy, které se také počítají z okolí hranového segmentu. Okolí se rozdělí horizontálně na dvě poloviny a vypočítává se jas pro každou polovinu zvlášť. V každé této polovině se vypočítává jas pro každý pixel zvlášť, a to pomocí vzorce 3.1. Jas celé poloviny je spočten jako aritmetický průměr jasů všech pixelů dané poloviny okolí. Zbylých 16 dimenzí zaplní X a Y souřadnice jednotlivých členů hranového segmentu. S pomocí tohoto zpracování je pro každý hranový segment získán jeho Feature vektor.

$$luminance = (red * 0.2126) + (green * 0.7152) + (blue * 0.0722) \tag{3.1}$$

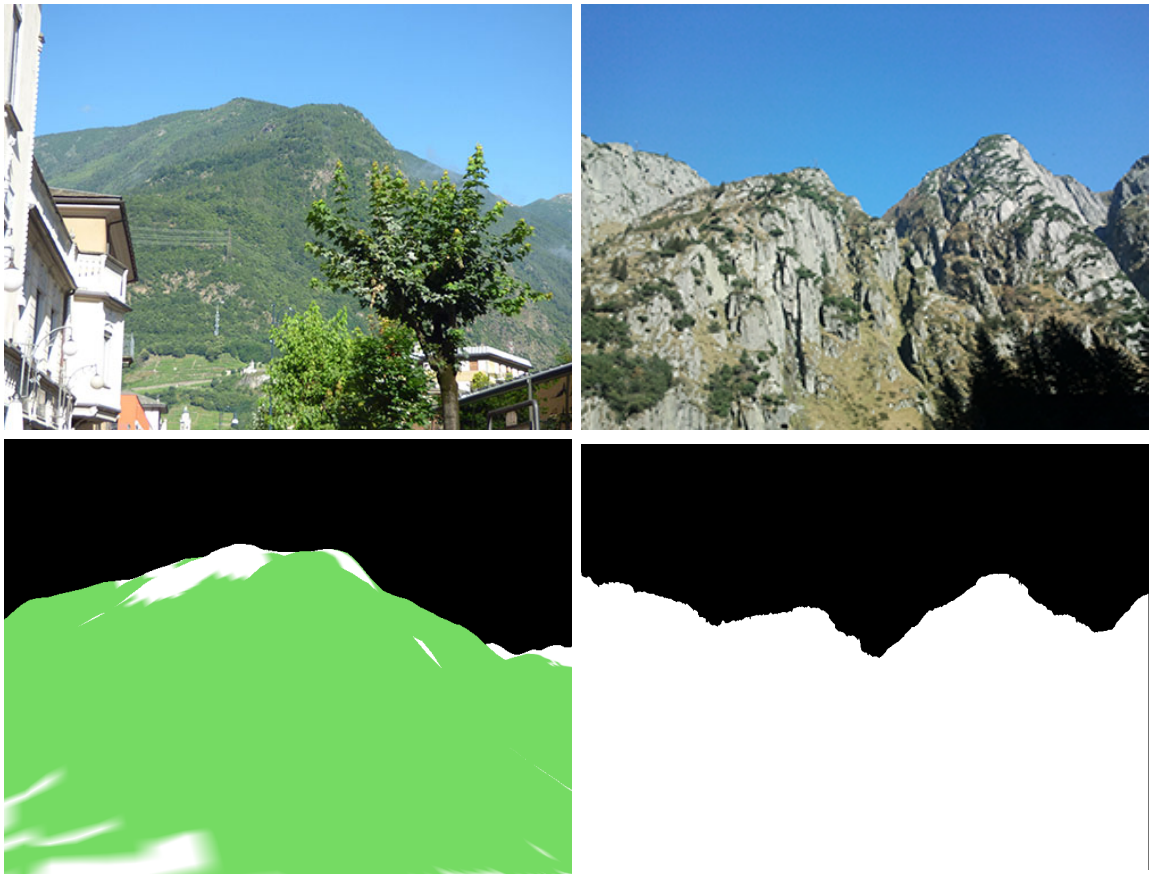
Implementace extrakce Feature vektoru probíhá společně se získáním hranové mapy a extrakcí hranových segmentů v balíčku *processImage* ve třídě *FeatureVector*. Pro reprezentaci a jednoduché ukládání informací o hranových segmentech a jejich Feature vektorech je připravena třída *Edge*, ve které jsou připravené datové struktury pro následné reprezentace okolí hranových segmentů a barev pixelů a pro vypočtené hodnoty jasů. Výstupem po získání Feature vektoru je kolekce objektů třídy *Edge*. Pro extrakci barevných složek pixelů okolí hranového segmentu jsou nejdříve zjištěny souřadnice levého horního pixelu okolí. Následně od této pozice probíhá získání barev postupně po sloupcích a řádcích. Při výpočtech jasu je opět iterativně po sloupcích a řádcích procházena horní / spodní polovina okolí, kde se pro každý pixel nejdříve získají jednotlivé barevné složky až následně vypočítává jas pixelu. Podle pozice pixelu se pak jas připočítává k sumě všech jasů daného okolí. V posledním kroku je jas aritmeticky zprůměrován.

3.5 Extrakce trénovacího vektoru

Trénovací vektor reprezentuje označení pozitivních a negativních hranových segmentů, respektive Feature vektorů. Informaci o tom, zda-li je Feature vektor pozitivní nebo negativní, získáme z ground truth nebo masek obrázků, které musí být připravené předem. Příklad ground truth a masky je vyobrazen na obrázku 3.4.

Nejdříve se daný obrázek načte a je z něj extrahována skutečná křivka horizontu pomocí jednoduchého procesu. Obrázek je prohledáván po pixelech zleva–doprava a shora–dolů. Při zjištění změny barvy je pixel zaznačen jako součást křivky horizontu. Po extrakci celé křivky jsou postupně všechny Feature vektory pixel po pixelu zkontrolovány podle X-ové souřadnice, zda-li spadají do intervalu tolerance skutečné křivky horizontu. Každý pixel Feature vektoru je tedy porovnán podle Y-ové souřadnice. Pokud výšky pixelů skutečné křivky horizontu a hranového segmentu sedí, je daný Feature vektor označen jako pozitivní. Interval tolerance je zaveden kvůli možným odchylkám Cannyho hranového detektoru.

Ground truth v GeoPose3K [12] Maska v CH1 [9]



Obrázek 3.4: Ukázky fotografií a jejich ground truth a masky použitých datových sad.

3.6 Trénování modelu

Trénování modelu probíhalo na datové sadě CH1 [9], která obsahuje 203 obrázků společně s jejich maskami. Z každého obrázku se postupně extrahovaly Feature vektory a korespondující trénovací vektory. Aby pro účely trénování byl vyvážený počet pozitivních a negativních Feature vektorů, byly ponechány všechny nalezené pozitivní Feature vektory. Negativních Feature vektorů bylo vždy mnohem větší množství, a proto se negativní vektory navzorkovaly takovým způsobem, aby byl počet roven pozitivním vektorům. Způsob selekce negativních vektorů je řízen podle podle podílu počtu negativních a počtu pozitivních Feature vektorů[8]. Celkem bylo z datové sady CH1 extrahováno 12 684 Feature vektorů. Hodnoty ve Feature vektorech byly následně ještě po jednotlivých dimenzích přeškálovány do intervalu [0..1]. Trénovací datová sada s Feature vektory byla exportována do LIBSVM formátu (příklad takového formátu je znázorněn v kódu 3.1). První hodnotou je označení pozitivního / negativního vektoru. Následují dvojice hodnot, kde číslo před dvojtečkou označuje dimenzi. Číslo za dvojtečkou hodnotu v dané dimenzi.

[8] Např. pro 12.000 negativních a 4.000 pozitivních je jejich podíl roven 3. Z negativních Feature vektorů se tedy vybere každý třetí.

```
0 1:-0.513725 2:-0.488189 3:-0.629921 4:-0.498039 5:-0.458824
1 1:0.0509804 2:0.023622 3:-0.0944882 4:0.0431373 5:0.0117647
0 1:-0.827451 2:-0.84252 3:-0.88189 4:-0.772549 5:-0.772549
1 1:0.013642 2:0.19846 3:-0.0944882 4:0.0118373 5:0.04537
```

Kód 3.1: Ukázka formátu LIBSVM.

Implementovaný program v Javě vygeneruje soubor s Feature vektory a jejich ohodnoceními. Avšak pro samotné natrénování modelu byl použit nástroj *easy.py*, který je k dispozici spolu se základní distribucí LIBSVM (kompletní knihovna LIBSVM včetně tohoto nástroje je k dispozici v adresáři projektu). Tento skript napsaný v pythonu provádí zcela automaticky:

- Přeškálování jednotlivých vektorů.
- Provedení „cross“ validace[9].
- Nalezení nejoptimálnějších parametrů.
- Samotné natrénování klasifikátoru a uložení do modelového souboru.
- Testování přesnosti klasifikace na stejné sadě vektorů.

Nástroj jsem spustil na exportované datové sadě obsahujícími Feature vektory včetně informace zda jsou pozitivní nebo negativní. Výsledkem bylo nalezení optimálních hodnot: c = 32.0; g = 0.03125; CV rate = 94.4182. Natrénovaný model měl přesnost klasifikace na trénovací datové sadě 99.0697 % (12566/12684).

3.7 Predikce pozitivních segmentů

Algoritmus Skyline Localization [24] využívá natrénového modelu k predikci pozitivních hranových segmentů. Při procesu predikce pozitivních segmentů se ze vstupního obrázku extrahují hranové segmenty a jejich příslušící Feature vektory. Tyto vektory jsou následně po jednom posílány do SVM, který má načtený model, a zde probíhá predikce. Výstupem predikce může být pouze zda-li je vektor pozitivní či nikoliv.

Implementace predikce spočívá v použití stejné extrakce hranových segmentů a Feature vektorů jako pro trénování modelu. Predikce samotná je implementována ve třídě *Localization*, kde je přímo využit java kód LIBSVM pro predikci. Před samotnou predikcí je ale nutné vektor naškálovat podobně jako to provedl nástroj *easy.py* předtím, než spustil cross validaci. Pro naškálování jsem upravil zdrojový kód ve třídě *svm_scale* LIBSVM. Feature vektor je tedy nejdříve naškálován do intervalu [-1..1] a až následně je poslán na predikci do SVM. Tento proces proběhne postupně pro všechny extrahované hranové segmenty a výsledky všech predikcí jsou zaznamenávány pro další zpracování.

[9] Cross validace je funkce, která napomáhá nalezení co nejlepších parametrů pro trénování v LIBSVM.

3.8 Spojování segmentů pomocí dynamického programování

Poslední částí algoritmu detekce křivky horizontu je spojování segmentů za asistence dynamického programování popsaného v [24]. Na základě pozitivních hranových segmentů a ignorování negativních je možné spojit pozitivní hrany do kompletní křivky horizontu. Originální vstupní obrázek je převeden na energetickou mapu váhováním gradient obrázku za pomoci rovnice 3.2.

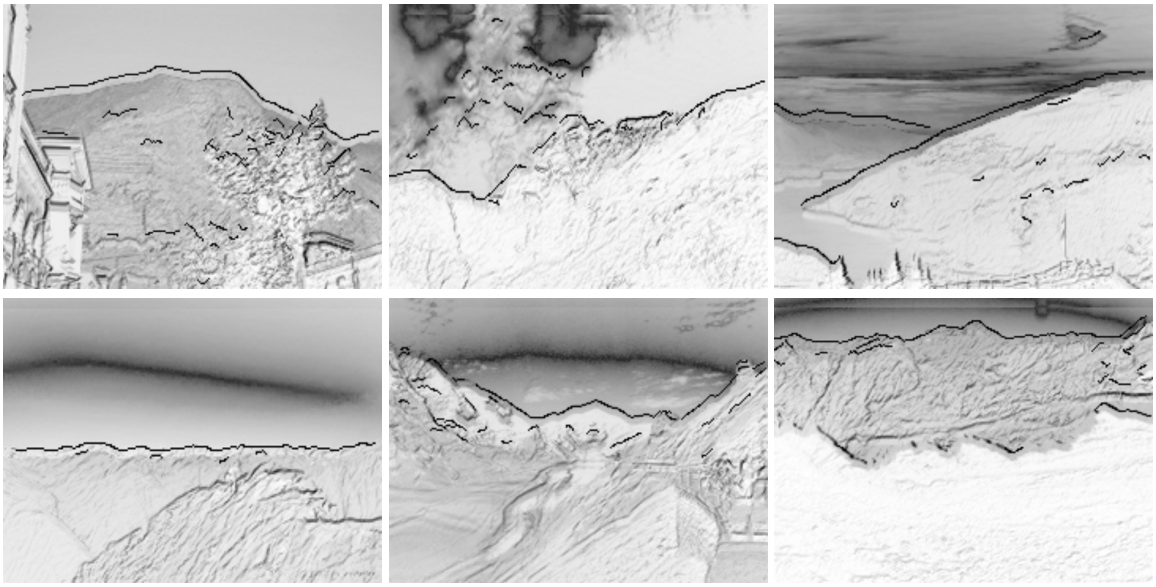
$$e_p = \begin{cases} 0 & if \quad p \in S \\ \alpha_p(1 - G_p) & otherwise \end{cases} \tag{3.2}$$

Kde e_p je energie pro pixel p. S označuje množinu pixelů křivky horizontu a G_p označuje normalizovanou gradientní hodnotu p. α_p je váha energetické funkce pixelů nepatřících do křivky horizontu.

Pixely křivky horizontu p jsou výsledky predikce SVM. Pro pozitivní hranové segmenty je velmi pravděpodobné, že jsou součástí křivky horizontu a proto jsou jejich pixely v energetické mapě zaznačeny nulovou hodnotou. Rovnice 3.3 je váhovaný parametr pro zvýšení rozdílu mezi pixely, které se podobají křivce horizontu a ostatními pixely.

$$\alpha_p = log(\parallel c_p^d - c_{sky}^d \parallel + \epsilon) \tag{3.3}$$

Kde C_p^d označuje barvu pixelu na souřadnicích *(x, y-d)*, kde d je pozice *(x,y)*. C_{sky}^d označuje medián barev pixelů nad pixelem křivky horizontu s malou distancí d. Barevná distance - norma je popsána v RGB barevném prostoru. ϵ je hodnota větší než 1 pro zajištění, že váha α_p bude vždy větší než 0.



Obrázek 3.5: Ukázka generovaných energetických map.

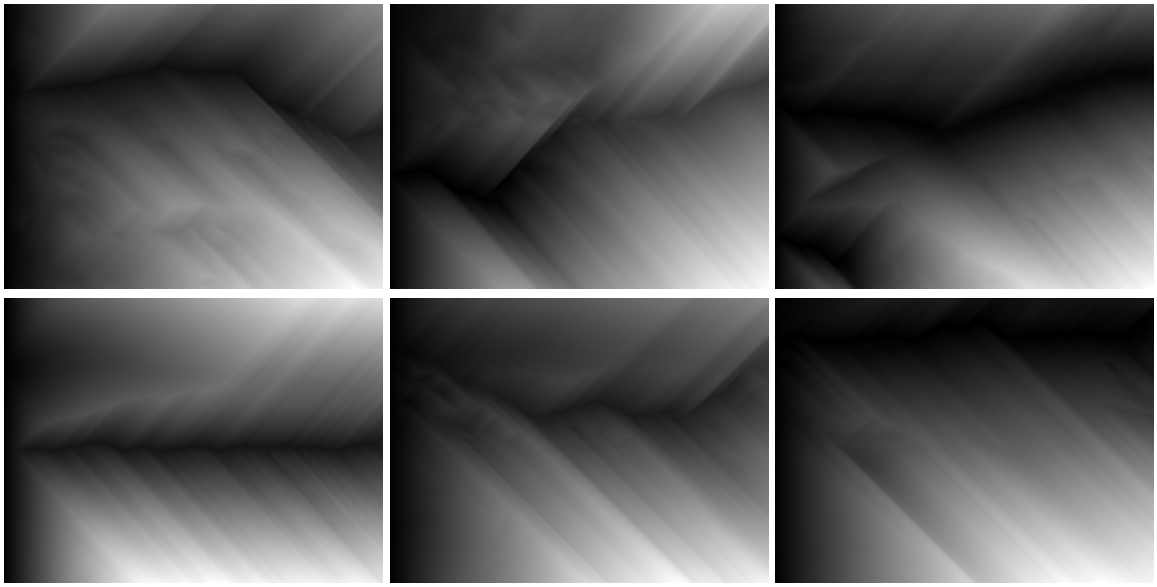
Po získání energetické mapy (viz. 3.5) přichází na řadu vyřešení problému minimalizace kumulativní energie. Předpokladem je, že křivka horizontu může být extrahována jako horizontální cesta zleva do prava. V případě tohoto algoritmu je proces dynamického programování *Seam Carving* [7] dobrá volba, protože řeší rychle a přesně nalezení cesty s nejmenší kumulativní energií. Pixel s menší energií bude mít větší příležitost stát se součástí cesty, pokud již nebyl vybrán pomocí SVM. Algoritmus Seam Carving však selhává při hledání horizontální cesty pro prudké kolmé svahy vzhledem k tomu, že další kandidáty hledá pouze mezi sousedy, které jsou na pozicích: vpravo nad, vpravo a vpravo pod. Proto byl algoritmus upraven a byly definované rekurzivní funkce 3.4, 3.5, 3.8 pro vyřešení problému s kolmými svahy.

$$E^0_{p(x,y)} = \begin{cases} e_{p(x,y)} & if \quad x = 1 \\ min(E^f_{p(x-1,y-1)}, E^f_{p(x-1,y)}, E^f_{p(x-1,y+1)}) + \epsilon_{p(x,y)} & otherwise \end{cases} \tag{3.4}$$

$$E^i_{p(x,y)} = min(E^{i-1}_{p(x,y-1)} + \epsilon_{p(x,y)}, E^{i-1}_{p(x,y)}, E^{i-1}_{p(x,y+1)} + \epsilon_{p(x,y)}) \tag{3.5}$$

$$E^f_{p(x,y)} = E^j_{p(x,y)}, \ if \ E^j_{p(x,y)} = E^{j-1}_{p(x,y)} \ for \ all \ x \tag{3.6}$$

Pro pixely v nejlevějším sloupci obrázku platí $E^0_p = e_p$. Jinak je E^0_p kumulativní energií nejkratší cety z pixelů nejlevějšího sloupce do pixelu p. Cesta nikdy neprochází skrz jakékoliv pixely ve stejném sloupci jako p. Narozdíl od E^0_p je nejkratší cesta skrz další pixely ve stejném sloupci jako p bude označena v E^i_p, kde i označuje i-tou iteraci. Ve chvíli kdy se hodnoty E^j_p ustálí pro všechny pixely sloupce j-té iterace, pak E^j_p bude nastavena jako E^f_p, která je finální nejmenší kumulativní energií z pixelů nejlevějšího sloupce do aktuálního pixelu p.



Obrázek 3.6: Ukázka kumulovaných hodnot původní energetické mapy po první iteraci.

Největší rozdíl oproti Seam Carving je zavedení možnosti hledání dalšího kandidáta s nejnižší energií také nad a pod aktuálně zpracovávaným pixelem v rovnici 3.5. Tímto je proto možné detekovat horizonty obsahující kolmé strmé srázy. Ve chvíli, kdy se nalezne minimum v rovnicích 3.4 a 3.5, můžeme zaznamenat cestu sledovat nejkratší cestu pro každý pixel. Po výpočtu finální kumulativní energie pro všechny pixely v obrázku můžeme zpětným průchodem získat cestu s nejmenší kumulativní energií, která začíná v nejpravějším sloupci v pixelu s nejmenší energií.

Implementace tohoto algoritmu kopíruje teoretický základ [24]. Nejdříve jsou do energetické mapy zaneseny pixely pozitivně predikovaných hranových segmentů. Nad těmito pixely jsou rovnou spočítány mediány potřebné pro vypočtení α. V dalším průběhu je vytvořen horizontální a vertikální kernel pro výpočet váhovaných gradientů. Následuje iterování pro jednotlivých sloupcích a řádcích, kde je pro každý pixel (vyjma těch s hodnotou 0) spočítána energie podle 3.2. Následuje cyklus pro postupné distribuce kumulativních hodnot (viz. 3.6) dle definovaných rovnic 3.4, 3.5, 3.8. Ve chvíli, kdy se hodnoty ustálí, je provedeno zpětné hledání nejkratší cesty (viz. 3.7).

Obrázek 3.7: Ukázka extrahovaných křivek horizontu zpětným hledáním nejkratší cesty v kumulovaných hodnotách původní energetické mapy.

3.9 Testování na datové sadě GeoPose3K

V rámci tohoto algoritmu byl natrénován model na datové sadě CH1 [9]. Získaný model byl podroben evaluaci na datové sadě GeoPose3K [12]. Tato datová sada obsahuje více než 3000 fotografií v hornatém prostředí včetně jejich ground truth a dalších anotací. Implementovaný algoritmus musel být nepatrně upraven pro možnost automatické evaluace modelu na všech fotografiích GeoPose3K. Vzhledem k tomu, že datová sada CH1 je podmnožinou také datové sady GeoPose3K, musely tyto fotografie být z evaluace vyloučeny, aby nezkreslovaly a neovlivňovaly naměřené výsledky. Celkem tedy měření probíhalo na 2916 fotografiích. Testování probíhalo na virtuálním stroji s těmito parametry: OS: Linux Ubuntu 16.04.4

LTS (64-bit) xenial; RAM: 4 GB; Procesor: 3 jádra; Video paměť: 64 MB; Pevný disk: SATA 120 GB; Další nastavení[10]. Virtuální počítač běžel v rámci Oracle VM VirtualBox[11] na fyzickém stroji s parametry: OS: Windows 10 Home (64-bit), verze 1709, číslo sestavení 16299.371; RAM: 16 GB, Procesor: Intel(R) Core(TM) i7-4710HQ s taktem 2,50 GHz, Grafická karta: NVIDIA GeForce GTX 860M, Pevný disk: KINGSTON SSD 128 GB, WDC WD10JPVX-22JC3T0 1 TB.

3.9.1 Definice metrik

Pro účely automatické evaluace byly definovány ekvivalentní metriky jako například v [6] nebo [33]. Definovanými metrikami tedy jsou:

Střední přesnost (Mean accuracy)

Pro každý obrázek je počítáno jaké pixely byly korektně klasifikovány. Formálně můžeme definovat metriku jako:

$$DC = \frac{1}{N_{set}} \sum_{i=1}^{N_{set}} \frac{N_c^i}{N_t^i} \tag{3.7}$$

Kde N_{set} je celkový počet obrázků datové sady. N_i^c a N_i^t jsou počty korektně klasifikovaných pixelů a celkový počet pixelů v obrázku i. V perfektním případě 100 % přesnosti klasifikace bude metrika dávat nejlepší hodnotu 1.

Průměrná absolutní vzdálenost pixelů (Average absolute pixel distance)

V této metrice se měří absolutní vertikální vzdálenost pixelů mezi výslednou segmentací a ground truth. Správná segmentace by se měla co nejvíce shodovat s ground truth a proto by vertikální rozdíl detekovaného a ground truth horizontu měl být minimální.

$$S = \frac{1}{N_{set}} \sum_{i=1}^{N_{set}} \left(\frac{1}{N} \sum_{j=1}^{N} |P_{d(j)}^i - P_{g(j)}^i| \right) \tag{3.8}$$

Kde $P_{d(j)}$ a $P_{g(j)}$ jsou pozice (řádky) detekovaného a ground truth horizontu ve sloupci j. N je počet sloupců obrázku. Rozdíly jsou dále zprůměrovány. Perfektní detekovaný horizont by v této metrice měl vracet hodnotu 0.

3.9.2 Experimenty

Implementovaný algoritmus pro extrakci křivky horizontu z fotografie byl podroben třem experimentům pro zjištění zda-li existuje nějaká možnost algoritmus vylepšit. Zlepšení je

[10] Dodatečné nastavení: Povolené IO APIC, HW čas v UTC, Omezení procesoru: 0 %, 3D Akcelerace, Povolené VT-x/AMD-V, Používání přímé komunikace s HW.

[11] Grafické uživatelské prostředí pro VirtualBox, verze 5.2.4r119785 (Qt5.6.2).

žádoucí z hlediska hlavně rychlosti, ale také přesnosti algoritmu. Systém dokumentace postupu byl inspirován z [6] a [33]. Všechny experimenty probíhaly s určitou modifikací stávajícího algoritmu. Vstupní datavou sadou bylo opět 2916 fotografií včetně jejich ground truth z GeoPose3K [12], abych tak mohl výsledky navzájem porovnávat i s výsledky z měření na původní verzi algoritmu. Výsledky experimentů byly měřeny definovanými metrikami z kap. 3.9.1 na virtuálním stroji specifikovaném v kap. 3.9.

Algoritmus	Střední přesnost	Průměrná absolutní vzdálenost pixelů
Originální implementace	95,0563 %	14,6260 px

Tabulka 3.1: Výsledky měření na originální implementaci algoritmu.

V prvním kroku provádění experimentů bylo naměření hodnot pro implementovaný algoritmus bez žádných úprav. Získal jsem tak referenční hodnoty znázorněné v tabulce 3.1, které mohu dále porovnávat s novými hodnotami získanými z měření dále uvedených experimentů.

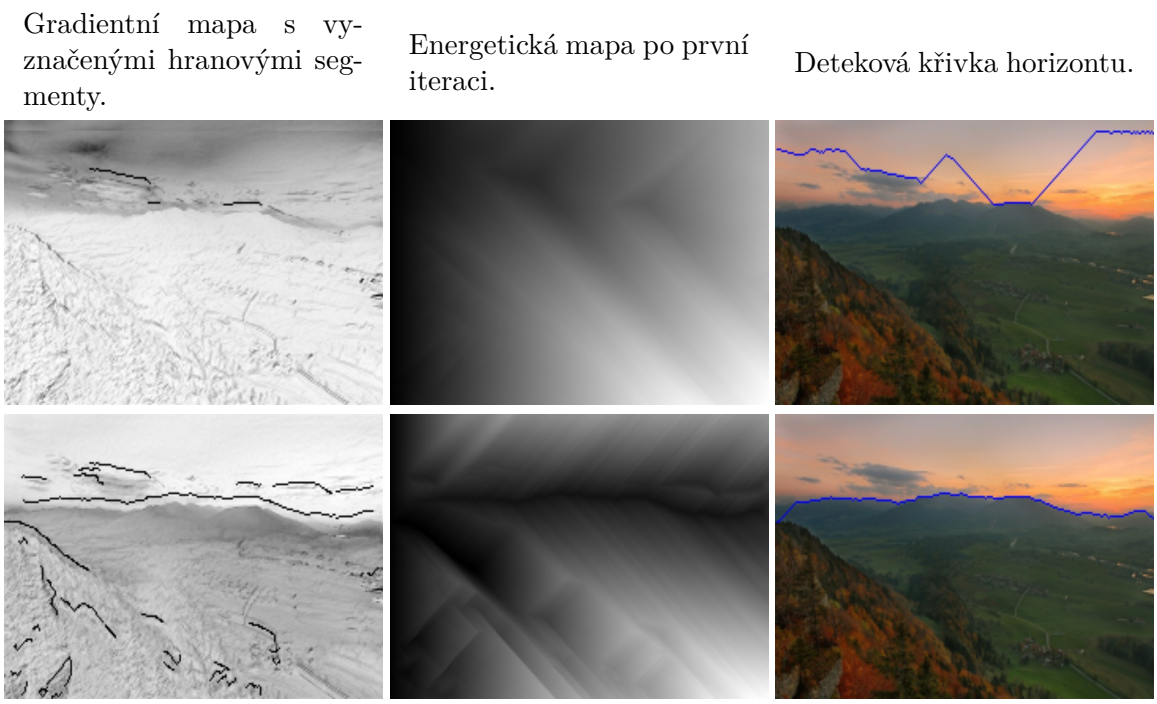
Experiment pro ignorování nízkého počtu predikovaných segmentů

Základní premisou pro uskutečnění tohoto experimentu byla namátkou vypozorovaná negativní vlastnost, která se projevila na výsledných detekovaných křivkách horizontu. Při procházení výsledků bylo možné si všimnout, že u některých obrázků byla detekovaná křivky určena zcela špatně i přesto, že na fotografii byl zcela zřetelně oddělený horizont, se kterým by si bez problémů poradily i jednoduché algoritmy na segmentaci popředí / pozadí. Po prozkoumání příčiny tohoto problému jsem zjistil, že příčinou chybně extrahované křivky byl nízký počet detekovaných pozitivních hranových segmentů z SVM. Často tak základem pro zpracování algoritmem pro distribuci kumulativních hodnot a hledání nejkratší cesty (viz. kap. 3.8) bylo jen málo pozitivních segmentů. Pokud z těchto segmentů je navíc některý predikován špatně, pak to má dramatický dopad na úspěšnost výsledné detekované křivky.

Základní otázkou v tomto experimentu tedy bylo: zda-li je možnost zvýšit úspěšnost lokalizace horizontu v případě, že byl predikován malý počet hranových segmentů. Algoritmus byl upraven, aby nižší detekovaný počet segmentů než definovaná hranice, byl zahozen. Namísto zahozených segmentů se do energetické mapy vyplní všechny hrany z Cannyho detektoru hran. V tomto experimentu byla testována hranice 10, 20, 30, 40 a 50. Do dalšího zpracování s pomocí dynamického programování tak nebyly zaneseny žádné predikované hranové segmenty z SVM, které by v opačném případě byly pevně zadefinované do gradientní mapy hodnotou 0. Celou detekci křivky pak má na starosti pouze algoritmus z kapitoly 3.8.

Výsledky experimentu jsou znázorněny v tabulce 3.2. Výsledky ukazují, že již hranice s hodnotou 10 zlepšuje průměrnou absolutní vzdálenost pixelů za cenu zanedbatelného zhoršení střední přesnosti v řádu setin procent. Strojově neměřitelnou, avšak člověkem vizuálně pozorovatelnou metrikou bylo optické zlepšení přesnosti detekce křivky horizontu právě pro obrázky, na které měla tato úprava algoritmu dopad (viz. 3.8). Samozřejmě s narůstající hodnotou hranice zahazování přibýval i počet dotčených obrázků. V tabulce 3.2 jsou také uvedeny počty obrázků na něž měla konkrétní hodnota hranice vliv. Je tak možné vypozorovat, že dramatické zhoršování výsledků od hranice 40 a výš je v důsledku toho, že se hranice dotkla i obrázků, kde byl již dostatečný počet správně predikovaných segmetů, které měly

vliv na správné zpracování v hledání nejkratší cesty. Tyto hranice měly celkově negativní vliv, protože působily na příliš velké množství obrázků.



Obrázek 3.8: Ukázka přínosu ignorování predikce: První řádek (originální implementace) - v gradientní mapě je vyznačený malý počet (a navíc nevhodně) predikovaných hranových segmetů. Energetická mapa je pak počítána špatně, což se odrazí na extrahové křivce horizontu. Druhý řádek (úprava s ignorací pro méně než 10 predikovaných segmentů) - gradientní mapa obsahuje všechny hranové segmenty. Energentická mapa je pak schopna lépe kumulovat energii a výsledná křivka horizontu je nesrovnatelně lepší než v originální implementaci.

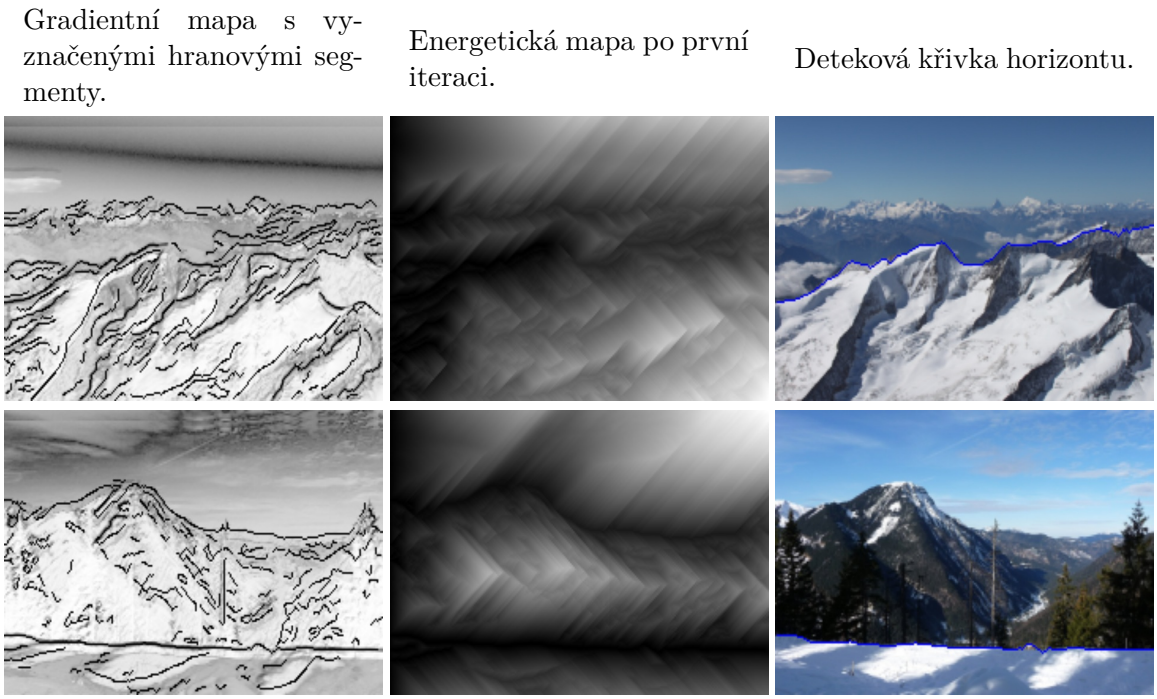
Hranice	Střední přesnost	Průměrná absolutní vzdálenost pixelů	Počet dotčených obrázků
10	95,0472 %	13,9748 px	66
20	94,8409 %	13,5312 px	164
30	94,2755 %	13,7678 px	342
40	84,8694 %	26,6737 px	677
50	85,6568 %	37,1541 px	1190

Tabulka 3.2: Výsledky experimentu s ignorováním nízkého počtu predikovaných segmentů

Experiment běhu bez predikce SVM

Predikování pozitivních a negativních hranových segmentů má velký dopad na čas a rychlost zpracování. Cílem tohoto experimentu bylo zjistit jak velký dopad na úspěšnost algoritmu by mělo úplné odstavení SVM z algoritmu a ponechání úkolu detekce křivky horizontu jen

na algoritmu s dynamickým programováním (viz. kap. 3.8. Pokud by dopad na úspěšnost byl malý a SVM bylo možné z procesu úplně vyloučit, došlo by tak k velmi výraznému zrychlení detekce. Predikce v SVM je největším konzumentem strojového času (viz. kap. 3.10). Jedná se vlastně o podobný pokus jako předchozí experiment. Avšak počet dotčených obrázků není omezen žádnou hranicí a vzhledem k nezávislosti na SVM je celá predikce SVM vynechána.



Obrázek 3.9: Ukázka špatných výsledků při vyloučení predikce SVM z procesu. První sloupec: použity jsou všechny hranové segmenty. Druhý sloupec: kumulace hodnot pro velké množství segmentů neprobíhá ideálně a často se vytváří několik kumulovaných cest. Třetí sloupec: z více možností cest je nalezena nejkratší cesta, která však neodpovídá skutečné křivce horizontu.

Algoritmus	Střední přesnost	Průměrná absolutní vzdálenost pixelů
Implementace bez SVM	84,8693 %	26,6737 px

Tabulka 3.3: Výsledky měření na implementaci algoritmu bez predikování SVM.

Výsledky měření jsou zobrazeny v tabulce 3.3. Naměřené hodnoty ukazují, že úplné vyčlenění predikce hranových segmentů pomocí SVM má fatální vliv na úspěšnost algoritmu (viz. 3.9). S tak zásadním zhoršením úspěšnosti se ukazuje, že SVM má v procesu velmi důležitou roli a nelze jej vyloučit i za cenu jeho náročnosti na strojový čas.

Experiment použití LIBLINEAR SVM

Projekt LIBSVM má svou větev LIBLINEAR [20] věnující se čistě rozsáhlé lineární klasifikaci. Klasický LIBSVM má klasifikaci implicitně nelineární. LIBLINEAR však používá čistě

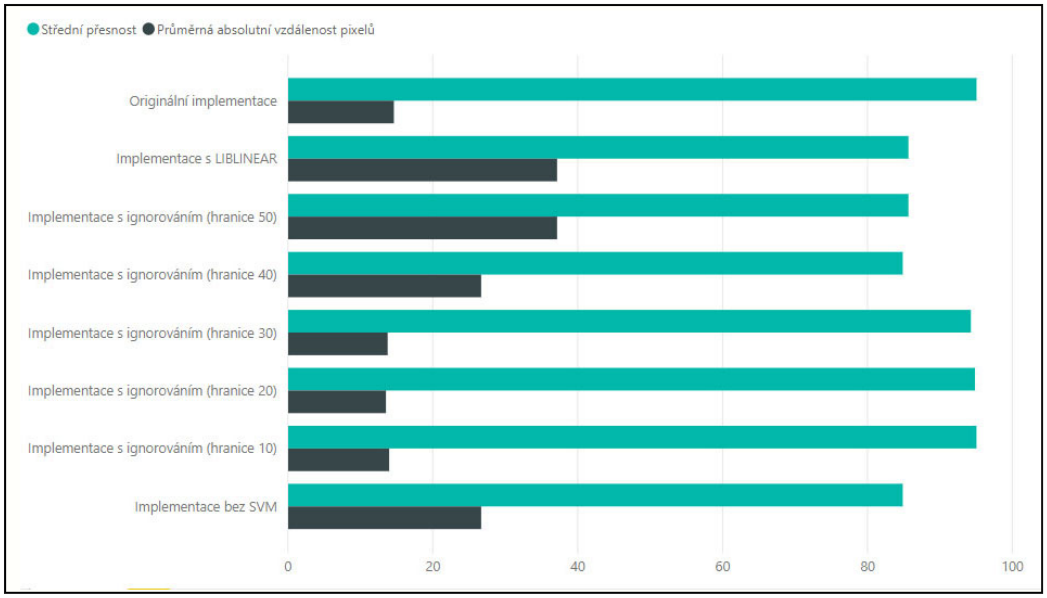
lineární klasifikaci. Pro určité řešené problémy tak může mít LIBLINEAR větší úspěšnost a zároveň nižší nároky na čas výpočtu než LIBSVM. Cílem experimentu bylo vyzkoušet LIBLINEAR namísto současného LIBSVM a zjistit, zda-li by klasifikace hranových segmentů mohla být rychlejší a úspěšnější pouhou změnou klasifikátoru. Kvůli změně klasifikátoru bylo třeba natrénovat nový model za pomocí LIBLINEAR. Vstupem byla opět datová sada CH1 [9].

Algoritmus	Střední přesnost	Průměrná absolutní vzdálenost pixelů
Implementace s LIBLINEAR	85,6568 %	37,1541 px

Tabulka 3.4: Výsledky měření na implementaci algoritmu bez predikování SVM.

Tabulka 3.4 obsahuje naměřené hodnoty za využití LIBLINEAR namísto LIBSVM. Úspěšnost klasifikace LIBLINEAR je však velmi nízká a má ještě horší výsledky než, kdybychom křivky horizontu detekovali úplně bez použití natrénovaného klasifikátoru. Jeho použití není pro tento konkrétní problém vhodným řešením.

3.9.3 Srovnání výsledků experimentů



Obrázek 3.10: Srovnání výsledků experimentů

V rámci experimentování byly provedeny tři pokusy, ze kterých byly naměřeny hodnoty definovaných metrik. Výsledky byly zpracovány ve formě pruhového grafu 3.10. Tabulka se všemi výsledky experimentů je v příloze C.1. Z výsledků experimentů lze vypozorovat, že oproti naměřeným výsledkům originální implementace dosahuje implementace s ignorováním na základě hranice 10 a 20 shodných výsledků v rámci metriky střední přesnosti. Pro průměrnou absolutní vzdálenost pixelů však originální implementace má horší výsledky než implementace s ignorováním. Zbylé experimenty dosahují mnohem horších výsledků než

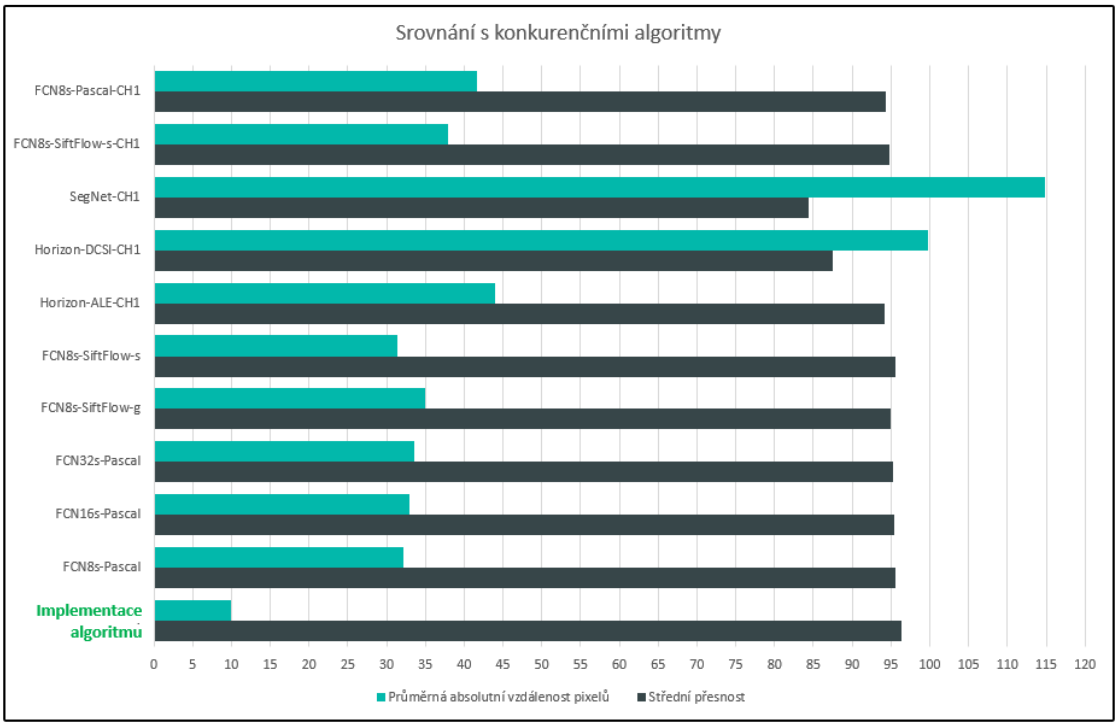
původní implementace. Nejlepších výsledků v rámci celého testování dosahovala implementace s ignorováním pod hranici 10 hranových segmentů. Na základě tohoto zjištění byla tato úprava nasazena trvale jako vylepšení původního algoritmu.

3.9.4 Srovnání s konkurenčními algoritmy

Algoritmus pro detekci křivky horizontu jsem dále srovnával s výsledky konkurenčních algoritmů [6] a [33]. Od autorů [6] jsem získal upravenou datovou sadu GeoPose3K [12], na které byly testovány algoritmy implementované právě v tomto dokumentu srovnávající aktuální nejlepší algoritmy pro detekci křivky horizontu. Na této upravené datové sadě jsem spustil algoritmus adaptovaný na zpracování upravené datové sady s již nasazenou úpravou 3.9.3 pro získání co nejlepších výsledků na srovnání. Cílem tohoto srovnání bylo zjistit jak si stojí SVM v porovnání s neuronovými sítěmi použitými v [6]. Obecně jsou neuronové sítě daleko přesnější než klasifikace pomocí strojového učení s učitelem. Počet obrázků, na kterých se testovalo byl shodný s předchozími experimenty.

Algoritmus	Střední přesnost	Průměrná absolutní vzdálenost pixelů
Implementace algoritmu	96,3709 %	9,9730 px

Tabulka 3.5: Výsledky měření implementace algoritmu na upravené datové sadě.



Obrázek 3.11: Srovnání s konkurenčními algoritmy

Výsledky měření na upravené datové sadě jsou v tabulce 3.5 a dosahují ještě lepších výsledků než na původní datové sadě. Výsledky byly společně s výsledky z [6] zpracovány do grafu 3.11. Kompletní tabulku se všemi daty tohoto srovnání jsou k dispozici v příloze C.2.

Data pro srovnání byla z [6] použita z tabulky po druhém post-zpracování „*postprocessing*“. Naměřené hodnoty testovaného algoritmu dosahují nejlepších výsledků střední přesnosti a pro průměrnou absolutní vzdálenost je zlepšení velmi vysoké. Nejlepších výsledků pro střední přesnost dosahuje, v rámci algotitmů porovnávaných v [6], algoritmus [28] ve verzi FCN8s-SiftFlow-s s hodnotou 95,63 %. Pro průměrnou absolutní vzdálenost pixelů je, opět v rámci [6], nejlepší opět [28]. Implementovaný algoritmus, jehož základem je SVM, dosahuje v tomto případě lepších výsledků než všechny algoritmy a jejich variace, které běží na konvolučních neuronových sítích.

Srovnání s výsledky [33] může proběhnout pouze porovnáním hodnot naměřených pro střední přesnost. Nejlepší hodnoty, v rámci [33], dosáhl autor v diplomové práci 93,73 % pro konvoluční neuronové sítě v rámci DeepLab v1 pro segmentaci na 11ti třídách. V porovnání na mé naměřené výsledky tyto hodnoty nestačí. Je však třeba vzít v úvahu, že zde se jedná o segmentaci do 11ti tříd. Pokud by autor zkusil segmentovat obrázky na 2 třídy, mohly by být výsledky i přesnější než mé naměřené hodnoty.

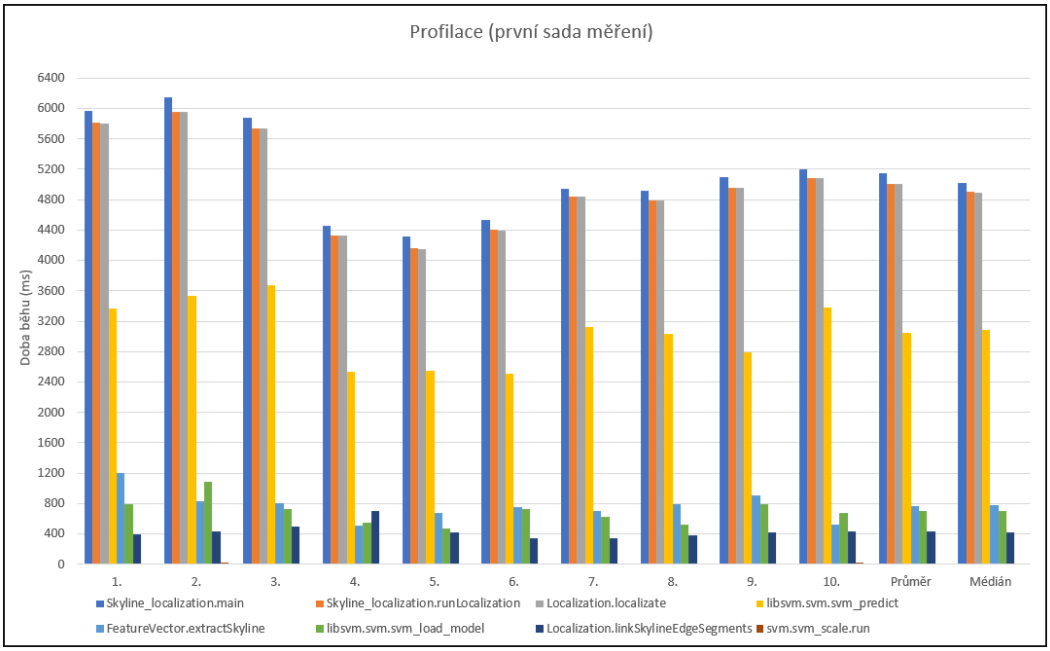
3.10 Profilace a měření

Poslední fází implementace algoritmu pro detekci křivky horizontu byla profilace algoritmu a následná optimalizace. Jedním z klíčových požadavků na nový systém je rychlost zpracování, proto bylo nutné algoritmus a všechny jeho funkční podčásti změřit a získat tak výchozí data. Na základě výchozích dat je pak možné určit hlavní problémy, které by se daly řešit a také je lze později využít například pro srovnání s dalším měřením po optimalizačních úpravách algoritmu. Měření v rámci profilace bylo zaměřeno na čas strávený zpracováním jednotlivých funkcí. Vzhledem k tomu, že spuštění a zpracování programu a všech jeho částí může být afektováno mnoha rozličnými faktory, jak z hlediska software tak hardware, bylo měření opakováno desetkrát. Z deseti měření pak je možné hodnoty aproximovat a zjistit tak aritmetický průměr nebo medián naměřených hodnot. Měření probíhalo v rámci funkce profilace platformy NetBeans[12] na virtuálním stroji specifikovaném v kap. 3.9. V rámci prvního spuštění proběhla nejdříve kalibrace[13] prostředí pro adaptování profilace na měřený algoritmus.

Výsledky prvního měření jsou znázorněny v grafu 3.12. Kompletní naměřené hodnoty jsou k dipozici v příloze D. Ve výsledcích je jako první uvedeno měření na metodě *Skyline_localization.main*, která je hlavní funkcí a řídí zpracování celého programu. Tato metoda kaskádově volá funkci *Skyline_localization.runLocalization* a *Localization.localizate*, které jsou na druhém a třetím místě. Všechny další metody jsou volány v rámci *Localization.localizate*. Z výsledků lze vypozorovat, že obecně program (v rámci profilace) spotřebuje na výpočet průměrně 5144,9 ms. Největší porci času si vezme zpracování metody *libsvm.svm.svm_predict*, jedná se o metodu knihovny LIBSVM, která provádí klasifikaci vstupní hodnoty. Tato metoda se stará o určení, zda-li je daný hranový segment součástí křivky horizontu. Průměrný čas predikování všech hranových segmentů v obrázku je 3051 ms. Dalším zásadním spotřebitelem času je metoda *FeatureVector.extractSkyline*, která má za úkol vstupní obrázek zpracovat a extrahovat z něj hranové segmenty. Ze seg-

[12] NetBeans IDE 8.1 (Build 201510222201) - https://netbeans.org/

[13] Výsledky kalibrace: Aproximovaný čas v jednom volání páru methodEntry()/methodExit(): Pro absolutní čas: *0,1211 μm*. Pro CPU čas vlákna: *0,43 μm*. Pro oba časy: *0,5446 μm*. Aproximovaný čas v jednom volání páru methodEntry()/methodExit() je navzorkován přístrojovým módem za: *0,0985 μm*.



Obrázek 3.12: Výsledky prvního měření

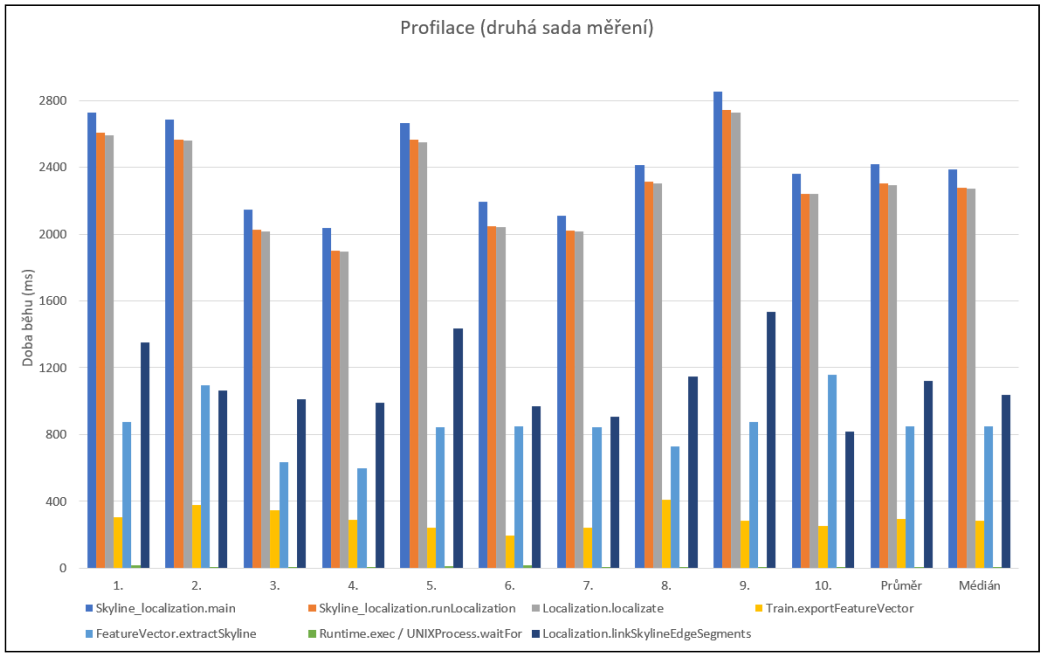
mentů pak metoda extrahuje hodnoty pro feature vektor. V této metodě zpracování vyžaduje v průměru 770,6 ms. Následující metodou je opět LIBSVM knihovní funkce *libsvm.svm.svm_load_model*, která se má starat pouze o načtení modelu ze souboru. Celý algoritmus pouze na načtení modelu SVM potřebuje průměrně 696,9 ms času. Poslední metodou, která stojí za zmínku je *Localization.linkSkylineEdgeSegments*, která následně zpracovává predikované hranové segmenty a s pomocí dynamického programování extrahuje křivku horizontu hledáním nejkratší cesty. Přesto, že zpracování zde probíhá rekurzivně, je třeba na průběh metody v průměru 437,4 ms času.

3.11 Optimalizace z hlediska času

Na základě zjištěných výsledků proběhla základní optimalizace [25] zdrojových kódů algoritmu eliminující hlavně nevhodné programovací praktiky při programování v jazyce Java. Avšak úprav ve zdrojovém kódu bylo nakonec minimum, vzhledem k tomu, že kód již byl programován s ohledem na vhodné programovací techniky a rychlé zpracování.

Při analýze výsledků měření bylo zjištěno, že algoritmus stráví velké množství času v knihovně LIBSVM. Metody knihovny jsou volány přímo z kódu, což může způsobovat delší časové trvání zpracování v jazyce Java oproti možnému spuštění binárního souboru, kompilovaného z původního jazyka knihovny C++. Pouhé načítání modelu trvá dlouhou dobu. V rámci algoritmu tedy proběhla úprava, kdy se extrahované feature vektory exportují do textového souboru. Predikce v LIBSVM pak probíhá za využití nachystaného spouštěcího binárního souboru svm-predict, který se spustí externě ze zdrojového kódu. Výsledky jsou uloženy do dalšího textového souboru, který je pak ze zdrojového kódu načten do programu a předán metodě *Localization.linkSkylineEdgeSegments*.

Samotný výpočet algoritmu v metodách pro získání feature vektorů a extrakce křivky horizontu[14] bohužel nebylo možné hlouběji optimalizovat. Metody vykonávají jen to nejnutnější, aby poskytly přesný výsledek v co nejkratším časovém intervalu.



Obrázek 3.13: Výsledky druhého měření

Po optimalizačních úpravách proběhla nová sada měření, jejichž kompletní výsledky jsou dostupné v příloze D. Vizualizace výsledků je k nahlédnutí v grafu 3.13. Ve vzájemném porovnání vypadla funkce *libsvm.svm.svm_predict* jelikož predikce probíhá externě přímo v binárním souboru svm-predict. Ze stejného důvodu už nebylo nutné načítat model, proto také vypadla funkce *libsvm.svm.svm_load_model*. Naopak zde přibyla funkce *Train.exportFeatureVector*, která exportuje extrahované Feature vektory do souboru. Pro zpracování predikce přibylo měření funkcí *Runtime.exec* a *UNIXProcess.waitFor*.

Porovnání výsledků druhého a prvního měření ukazuje na vysoké zlepšení rychlosti zpracování. Průměrné zpracování celého algoritmu v rámci *Skyline_localization.main* bylo v průměrném čase sraženo na poloviční hodnotu oproti původní implementaci. Zpracování *FeatureVector.extractSkyline* bylo v průměru navýšeno o 80 ms a v případě *Localization.linkSkylineEdgeSegments* dokonce o 700 ms z důvodu nově potřebného exportu ze souboru a následného načtení hodnot ze souboru. Tato navýšení zpracování konkrétních metod však zastiňuje uspořený čas v rámci predikce v SVM. V původní implementaci trvala průměrná predikce v součtu[15] 3747,9 ms. Po přesunutí predikce mimo kód do externí aplikace svm-predict je průměrná časová spotřeba na výpočet v metodách *Train.exportFeatureVector*, *Runtime.exec* a *UNIXProcess.waitFor* v součtu pouze přes 300 ms. Tato úspora i s připočítaným navýšením zpracování metod *FeatureVector.extractSkyline* a *Localization.linkSkylineEdgeSegments* znamená celkové průměrné zrychlení výpočtu algo-

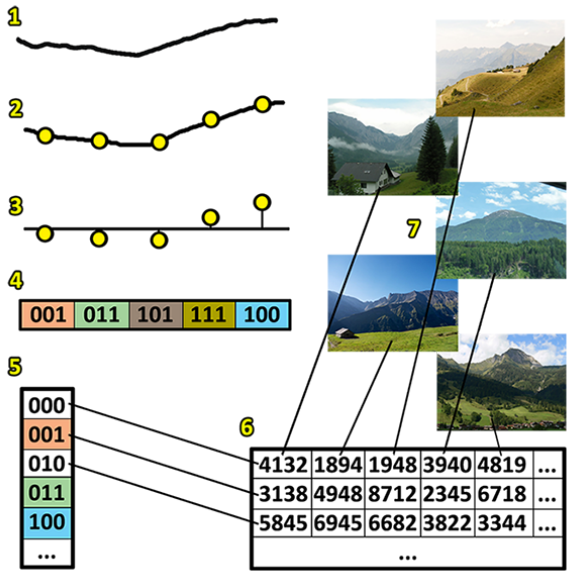
[14] *FeatureVector.extractSkyline* a *Localization.linkSkylineEdgeSegments*
[15] Načtení modelu *libsvm.svm.svm_load_model* a samotná predikce *libsvm.svm.svm_predict*.

ritmu z 5144,9 ms na 2419,8 ms. Optimalizace kódu pravděpodobně ušetřila setiny milisekund. Avšak optimalizace zpracování predikce v SVM měla majoritní vliv a je tedy velmi účelná. Tyto změny v rámci optimalizace byly v systému zavedeny na trvalo, protože citelně vylepšují spotřebu času pro práci algoritmu.

Kapitola 4

Implementace inverzního indexu

Jak již bylo popsáno v 2.3 je pro vyhledávání a porovnávání contourlettů využíváno inverzního indexu. Implementace v rámci projektu Locate načítá celý inverzní index do operační paměti. To však limituje ze spodní hranice minimální hardwarové nároky a z horní hranice možnost dalšího škálování. Řešením může být ponechat inverzní index uložen na pevném disku s použitím vhodné cachovací politiky. Tím bychom získali v paměti omezený prostor pro index a obě uvedené hranice bychom eliminovali za cenu určitého zpoždění[1]. Z tohoto důvodu byla provedena rešerše v rámci podkapitoly 2.3, ze které byla vyvozena konkluze - nejvhodnější řešení je implementovat vlastní inverzní index s cachovací politikou.



Obrázek 4.1: Průběh vyhledávání v inverzním indexu: 1. Detekovaná křivka horizontu je vyhlazena. 2. Křivka je navzorkována s pevným odstupem. 3. Vzorky jsou normalizovány. 4. Z normalizovaných vzorků jsou vypočítány biny (contourletty). 5. Contourletty jsou vyhledány v hlavním poli inverzního indexu. 6. Contourletta z hlavního pole indexu odkazuje na sekundární pole se seznamem ID dokumentů. 7. ID dokumentů odkazují na známé horizonty, ve kterých se daná contourletta nachází.

[1] Čtení z operační paměti je řádově rychlejší než čtení z disku.

4.1 Inverzní index

Proces vyhledávání v inverzním indexu je znázorněn v obrázku 4.1. Základní prerekvizitou inverzního indexu uloženého na pevném disku je systém či hierarchie v jakém bude index uložen. Sekundární pole budou uloženy jako samostatné soubory obsahující ID dokumentů. Primární pole ale bude uloženo jako hierarchie složek na jejímž konci bude právě soubor obsahující sekundární pole. Vzhledem k tomu, že systém poběží na Linuxu a tedy na souborovém systému Ext4, bylo potřeba analyzovat souborový systém a zjistit případná omezení. Podrobná analýza tohoto souborového systému byla provedena v [18], avšak zde není provedena analýza testující možnou závislost mezi poklesem výkonu a vzrůstajícím množstvím podsložek v jedné složce. Tuto a mnoho dalších závislostí testovali na Kalifornské univerzitě[2] a výsledky vystavili online [45]. Na základě této analýzy můžeme zjistit, že souborový systém Ext4 nabývá malého poklesu výkonu pokud je v jedné složce více než 1.000 podsložek. Tento pokles výkonu se navíc týká čtení i zápisu. Drastický pokles výkonu pak nastává pokud je v jedné složce více než 1.000.000 podsložek. Z tohoto důvodu byla zvolena horní hranice 1.000 podsložek pro hierarchii inverzního indexu. Prvky primárního pole inverzního indexu jsou celá nezáporná čísla, které můžeme hierarchicky rozdělit podle řádů. Inverzní index by tak měl být na disku uložen hierarchicky rozdělen tak, aby v každé podsložce obsahoval maximálně 1.000 podsložek. Toho můžeme docílit například tak, že číslo rozdělíme vždy po třech řádech (kvůli limitu 1.000).

Číslo 12.345.678 můžeme rozdělit na hierarchii složek: 12/345/12345678[3].

Inverzní index byl implementován v jazyce C++, aby bylo možné jej přímo zakomponovat do nástroje HLOC[4] projektu Locate, který je v tomto jazyce napsán. Pro práci se složkami a souborovým systémem byla použita knihovna Boost[5]. Hlavní funkcí inverzního indexu je možnost vložit hodnotu do sekundárního pole podle indexu primárního pole. Druhou hlavní funkcí je načtení a vrácení celého sekundárního pole podle indexu primárního pole. Obě tyto metody potřebují podle ID featury (contourletty) získat cestu k souboru, ve kterém je uložen seznam ID dokumentů. Metoda generující cestu přečte číslo po znacích zprava (od nejmenšího řádu) a postupně je konkatenuje do řetězce za sebe. Navíc za každé tři řády vloží lomítko (oddělovač identifikující složky v cestě souboru). Tento řetězec s extrahovanou cestou je následně reverzován do správné podoby a případně je doplněn hodnotami nula, pokud je číslo menšího řádu než řád maximálního čísla feature ID. Funkce pro zápis ID dokumentu do souboru feature ID si nechá extrahovat cestu k souboru. Na základě cesty se vytvoří příslušné složky a podsložky, pokud již nebyly vytvořeny dříve. Do souboru feature ID se na konec souboru připojí zapisované ID dokumentu.

Implementace inverzního indexu se týká také jeho reprezentace v operační paměti. Omezený prostor pro obsah právě načtených dokumentů featur je reprezentován jako vektor struktur obsahujících číslo featury, seznam dokumentů a případně informací sloužící pro cachovací politiky (viz. 4.2). Aby byla zachována konstantní složitost vyhledání, zda-li je daná featura načtena v paměti, byl vytvořen hlavní vektor o velikosti počtu featur. Načtené featury v paměti budou mít na svém příslušném indexu v hlavním vektoru uložen index

[2] University of California (https://www.universityofcalifornia.edu/)

[3] Poslední číslo v cestě obsahuje kompletní číslo protože se jedná o koncový soubor obsahující sekundární pole.

[4] Implementující algoritmus [39].

[5] Boost je volně dostupná C++ knihovna jejíž primární autoři jsou Berman Dawes, David Abrahams a Rene Rivera (https://www.boost.org/).

do sekundárního vektoru[6], ve kterém jsou načteny. Pokud featura načtená není, bude mít na svém indexu v hlavním vektoru hodnotu (-1). Metoda pro přečtení všech ID dokumentů uložených pod danou feature ID tedy nejdříve zjistí z hlavního vektoru jestli je featura k dispozci v paměti. Pokud ano, pak metoda vrátí vektor ID dokumentů dané featury. V opačném případě metoda přistoupí k načtení featury do paměti. Nejdříve je opět extrahována cesta k souboru featury. Ze souboru jsou pak načteny ID dokumentů, které se uloží do dočasného vektoru. Pokud je featura načtena správně tak je zvolenou cachovací politikou (viz. 4.2) vybrána featura, která má být v paměti nahrazena právě načtenou featurou. Proběhne nahrazení v sekundárním vektoru a metoda vrátí vektor ID dokumentů.

4.2 Cachovací politiky

Správa omezeného prostoru inverzního indexu v paměti musí být řízena vhodnou cachovací politikou[7]. Cachovací politiku můžeme chápat jako optimalizační instrukce či algoritmus, který řídí cachování informací uložených v počítači. Pokud je cache zaplněná, algoritmus musí vybrat, kterou položku zahodí pro vytvoření prostoru pro novou položku. V rámci této implementace inverzního indexu byly vybrány tři cachovací politiky: LRU, Random a FIFO. V následujících podkapitolách je rozebrán princip politiky a její implementace. Důvod výběru a implementace více politik bylo všechny politiky naměřit pro různě velké prostory v paměti. Na základě experimentů je pak možné vysledovat možné závislosti a vybrat nejvhodnější politiku pro použití v nástroji HLOC.

4.2.1 LRU - Least Recently Used

LRU je deterministická strategie, kdy je z cache pro nahrazení vybrán vždy prvek, jež byl ze všech prvků cache nejdéle nevyužíván. Strategie výběru tedy předpokládá, že ke všem prvkům musí být udržován kontext stárnutí. Při použití konkrétního prvku je třeba jeho kontext resetovat a všem ostatním prvkům kontext stárnutí zvýšit. Politika LRU je spravedlivá ve smyslu, že nejstarší vybraný prvek je pravděpodobně málo využíván (nebo méně často využíván) oproti ostatním prvkům, a proto je zahozen. Problémem však může být režie udržování kontextu stárnutí. LRU může být však považována za velmi spravedlivou politiku, právě proto, že afektuje reálnou využívanost načtených prvků a zahazuje vždy ten nejstarší / nejméně využívaný prvek.

Kontext stárnutí je v implementaci této politiky koncipován jako inkrementující celá nezáporná čísla (integery), která se uchovávají ve struktuře každé featury v paměti. Struktura tedy kromě ID featury a seznamu ID dokumentů obsahuje LRU čítač, který simuluje stárnutí prvku v paměti. Při čtení prvku z paměti je mu nastavena hodnota stárnutí na (0) a všem zbylým prvkům v paměti je hodnota stárnutí inkrementována o hodnotu (1). Stejný proces je aplikován i při nahrazování v paměti, kdy je novému prvku nastaven kontext stárnutí na (0) a stávajícím prvkům je kontext inkrementován. Při prvotním spuštění inverzního indexu je prázdným prvkům v cache nastaven kontext stárnutí na maximální hodnotu[8].

[6] Reprezentující omezený prostor v paměti.
[7] Cache replacement policies.
[8] Maximální hodnota je UINT_MAX ($2^{16} = 65.535$).

4.2.2 RANDOM

Tato strategie se neřadí mezi běžné konvenční cachovací politiky. Je to z důvodu, že za cenu absolutně nulové režie (a tedy ušetření času), je z paměti zahozen zcela náhodný prvek. Vůbec se však nebere v potaz, zda-li je zahození spravedlivé či nikoliv. Zahozen může být prvek, který je v paměti přítomen nejkratší dobu nebo je nejčastěji využíván. Výhodou politiky je velmi jednoduchá implementace a absolutně žádná režie. Algoritmus je tak poměrně rychlý, narozdíl od LRU, který má časově náročnou režii pro čtení, udržování a aktualizaci kontextu stárnutí. Velkou nevýhodou však může být právě nepředvídatelné chování výběru prvku k zahození. Pokud nedeterministický náhodný výběr nevhodně vybere v několika iteracích mladé nebo často využívané prvky, může být provoz inverzního indexu s touto politikou zbytečně časově náročný.

Implementace náhodného výběru byla realizována pomocí funkce, která vrací náhodnou hodnotu z uniformního rozložení.

4.2.3 FIFO - First In First Out

Politika FIFO je založena na deterministické strategii, kdy první prvek uložený do paměti je také jako první vybrán k zahození. Můžeme si proces představit jako frontu. Prvek jako první vstoupí do fronty a další prvky se staví za něj. Prvek, který do fronty vstoupil jako první, jej také jako první opustí. Celkově je strategie dost podobná LRU. FIFO nemusí udržovat kontext stárnutí jako LRU, avšak si musí udržovat kontext o pořadí prvku ve frontě. Oproti LRU navíc politika FIFO nebere vůbec v úvahu využívanost daného prvku. Pokud byl konkrétní prvek předchozí iterací čten z paměti, ale nachází se na konci fronty, je jeho pravděpodobně častá využívanost zcela ignorována a prvek je přesto zahozen. FIFO můžeme prohlásit jako částečně spravedlivý, protože sice simuluje stárnutí jako LRU, ale již nebere ohled na frekventovanost využití konkrétního prvku.

FIFO byla implementována jako fronta (*queue*), kterou nabízí C++. Manipulace s frontou je omezená, ale v rámci FIFO stačí mít k dispozici pouze funkce na vložení prvku na začátek fronty (*pop_front*) a vyjmutí prvku z konce fronty (*push_back*). Veškerou další režii fronty obstarává její nativní implementace v C++. Prvek, který má být zahozen, je tedy získán pouze invokací funkce pro vrácení prvku na konci fronty. Nový prvek je vložen na začátek fronty a jeho další postup ve frontě je ovlivňován automaticky na základě příchozích dalších prvků.

4.3 Experimenty s inverzním indexem

Implementovaný inverzní index byl podroben testování, pro zjištění možných závislostí nebo případných anomálií. Primárním důvodem testování však bylo odhalit, která z implementovaných cachovacích politik je nejvhodnější. Sledovanými metrikami v experimentech byly CPU čas zpracování a počet přepisů v paměti (*cache miss*). Testování probíhalo na virtuálním stroji specifikovaném v kapitole 3.9. Všechny experimenty byly provedeny pro každou cachovací politiku pro různé velikosti vyhrazeného prostoru pro index v paměti. Zvolené hodnoty měřených velikostí indexu v paměti byly 1 %, 2 %, 5 %, 10 %, 15 % a 20 % z celkového počtu featur. Kromě měření cachovacích politik a různých velikostí indexu v paměti, byla také naměřena původní implementace indexu z HLOC. Původní implementace nepoužívá žádnou cachovací politiku, protože má celý index načten v paměti. Cílem měření původní implementace bylo zjistit jak moc se inverzní index časově zpozdí vlivem cachova-

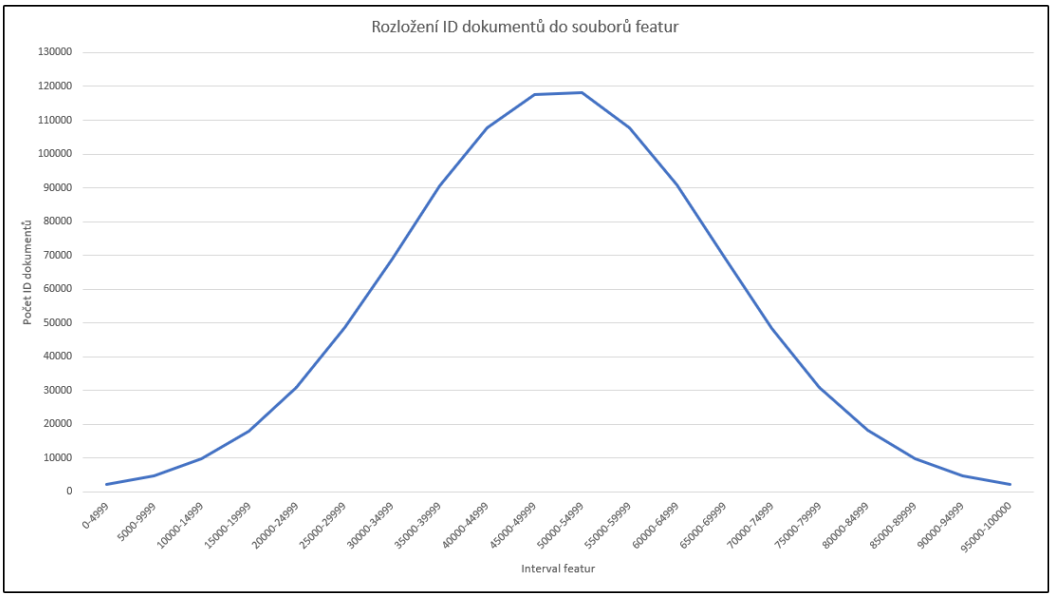
cích politik a omezených prostorů v paměti, a tedy zjistit jaká je cena rychlosti za možnost dalšího škálování inverzního indexu. V následujících podkapitolách jsou rozebrány jednotlivé experimenty včetně jejich vyhodnocení. Poslední podkapitolou je závěrečné zhodnocení naměřených výsledků a selekce optimální politiky pro využití v novém inverzním indexu.

V rámci těchto experimentů byly ověřovány tyto premisy:

- S rostoucí velikostí indexu v paměti by měly klesat časové nároky na zpracování.
- S rostoucí velikostí indexu v paměti by měl klesat počet přepisů prvků v paměti.

4.3.1 Experiment pro generovaná data

Důvodem provedení tohoto experimentu bylo vysledovat, jakým způsobem se budou politiky chovat na generovaných datech a získat tak základní představu o tom, jak se pravděpodobně budou politiky chovat na reálných datech. Výsledky tohoto experimentu je pak možné porovnat s výsledky na reálných datech, které by měly kopírovat stejné závislosti jako výsledky z generovaných dat.

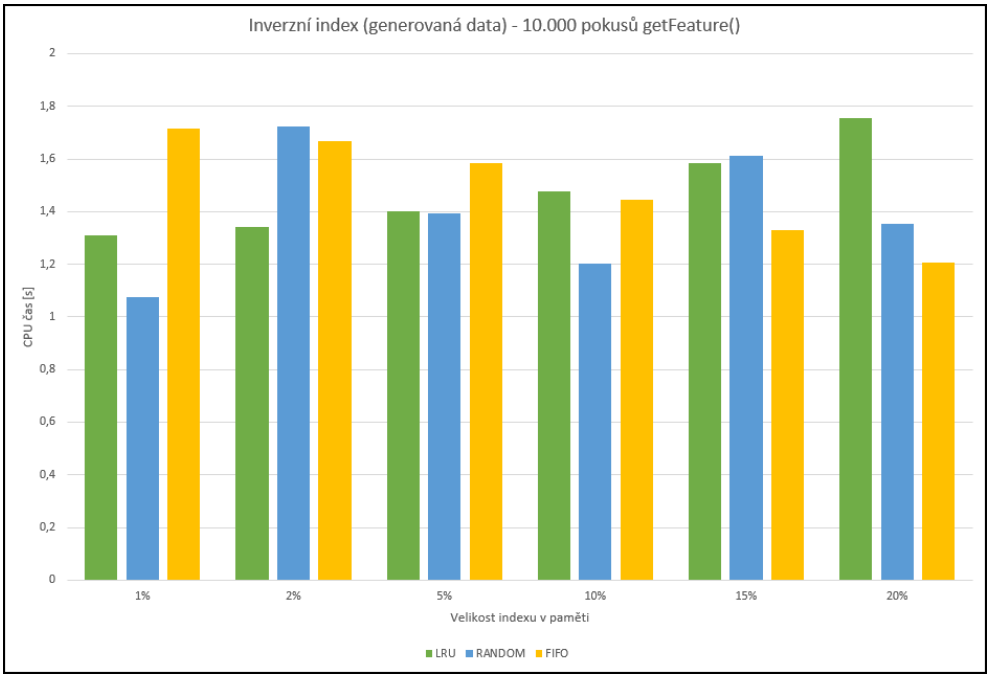


Obrázek 4.2: Distribuce ID dokumentů mezi featury na základě normálního rozložení.

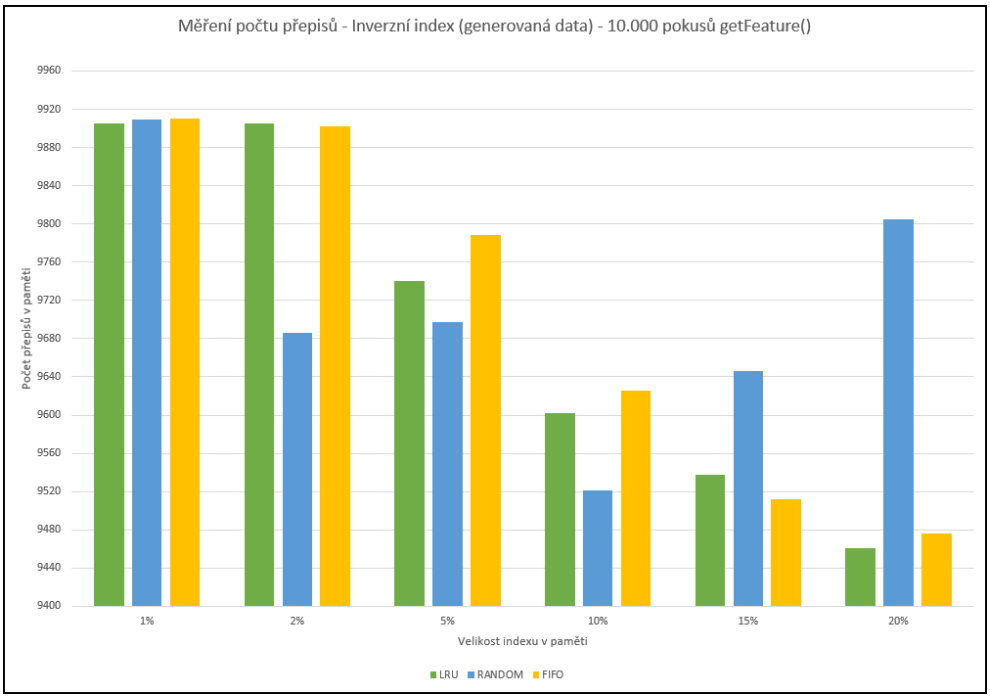
Příprava experimentu spočívala v automatickém vygenerování dat inverzního indexu. Pro tento experiment byl určen počet featur na 100.000. Generování probíhalo iterativní invokací metody pro zápis ID dokumentu dokud nebylo dosaženo hodnoty 1.000.000. Tato hodnota byla zvolena jako počet dokumentů rozložených mezi všechny featury. Distribuce ID dokumentů do featur bylo řízeno na základě náhodné hodnoty z normálního rozložení (viz. obr. 4.2). Tento proces vygeneroval potřebnou hierarchii na pevném disku a do featur nageneroval náhodné ID dokumentů. Pro každou politiku a každou již definovanou velikost indexu (viz. 4.3) bylo provedeno 10 měření. V jednom měření se testovalo 10.000 pokusů načtení náhodně[9] vybrané featury, přičemž prostor v indexu byl v počátečním stavu prázdný.

[9]Na základě uniformního rozložení.

Referenční průměrná hodnota času zpracování 10.000 požadavků na získání featury měřená na původní implementaci je 0,030212 s (výsledky měření všech pokusů je k dispozici v příloze E).



Obrázek 4.3: Výsledky měření času.



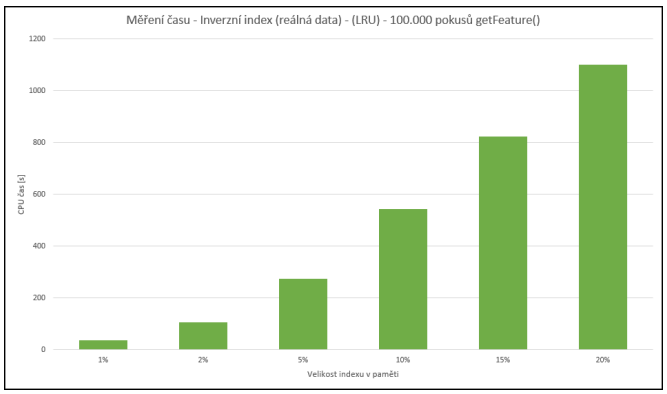
Obrázek 4.4: Výsledky měření počtu přepisů.

Průměrné hodnoty výsledků pro měření CPU času jsou znázorněny v grafu 4.3. Kompletní naměřené výsledky jsou k dispozici v příloze E. Z obdržených výsledků lze pro LRU politiku sledovat trend, který se projevuje jako vzrůstající funkce v závislosti na zvětšování prostoru indexu v paměti. To může být způsobeno právě dopadem režie LRU na aktualizaci kontextu stárnutí. Se zvětšujícím se prostorem roste i náročnost na údržbu kontextu stárnutí pro všechny prvky. K trendu LRU je kontrastní detekovatelný trend pro FIFO, který může být popsán jako klesající funkce v závislosti na rostoucím prostoru indexu v paměti. Tento trend podporuje první premisu (viz. 4.3). Pro politiku RANDOM nelze vysledovat žádný charakteristický trend, jelikož průměrné hodnoty pro různě velké indexy v paměti kolísají zcela náhodně.

Pro měření počtu přepisů prvků inverzního indexu v paměti jsou průměrné hodnoty dostupné v grafu 4.4. Tabulka obsahující všechny naměřené hodnoty všech pokusů je zobrazena v příloze F. Průměrné hodnoty počtu přepisů pro LRU s rostoucí velikostí paměti klesá i přesto, že časová náročnost na zpracování stoupá. Zde je vidět paradox, že i přes snížení počtu přepisů v paměti (a tím i ušetření času), CPU čas roste. To ukazuje, že ušetřený čas je zanedbatelný v porovnání se vzrůstajícím časem, který vzniká rostoucí režií politiky LRU (spolu s rostoucí velikostí indexu). Politika FIFO zobrazuje chování, které bychom čekali v rámci korespondence s časovými výsledky. Se zvětšujícím se prostorem indexu v paměti klesá počet přepisů (častěji je právě hledaná featura již přítomna v paměti), a tím klesá i náročnost na CPU čas (čtení z operační paměti je časově řádově rychlejší než čtení z pevného disku). U politiky RANDOM jsou výsledky opět náhodné a ukazuje se, že zvyšující paměť má minimální vliv na počet přepisů.

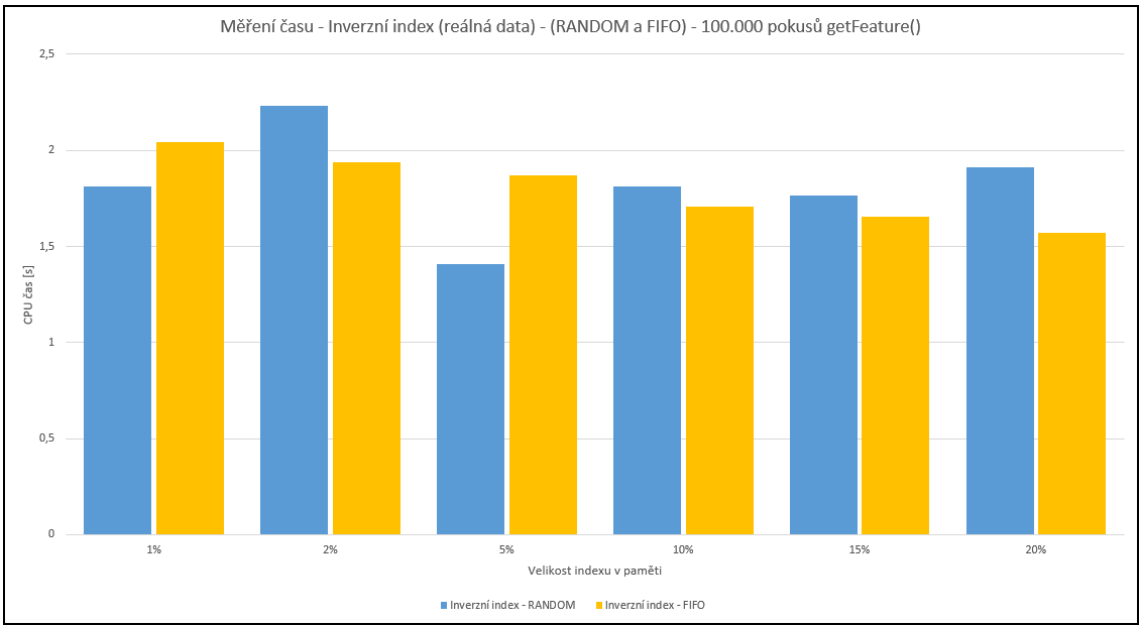
4.3.2 Experimenty pro reálná data

Nový inverzní index byl zakomponován do nástroje HLOC a nahradil tak původní implementaci. Nástroj musel být adaptován pro nový index a optimalizován pro lepší využití služeb nového indexu. Tento experiment byl tedy proveden přímo v nástroji HLOC na reálných datech, které jsem dostal k dispozici od vedoucího této práce. Datová sada reálných dat se skládá z 1.000.000 pozic a obsahuje 16.777.216 featur. Experimentování na reálných datech bylo opět provedeno pro všechny cachovací politiky včetně všech velikostí indexu. Měření probíhalo na 100.000 požadavků na čtení featury, přičemž každé měření proběhlo 10×. Referenční průměrná hodnota času na původní implementaci s kompletním inverzním indexem načteným v paměti bylo 0,0167928 s (výsledky měření všech pokusů je k dispozici v příloze G).



Obrázek 4.5: Výsledky měření času pro LRU.

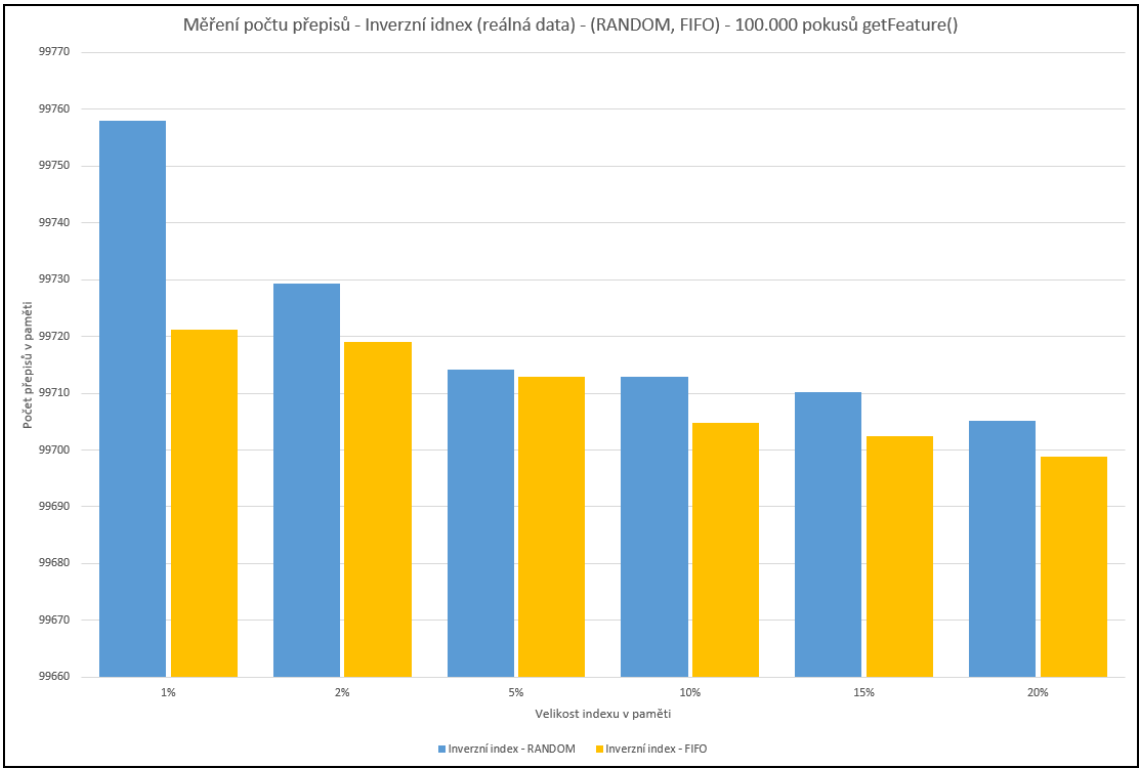
Výsledky měření času pro LRU byly umístěny do vlastního grafu 4.5 kvůli až příliš překvapivým hodnotám výsledků. Naměřené hodnoty jsou příliš vysoké, než aby je bylo možné srovnat s naměřenými výsledky politik RANDOM a FIFO. Vzhledem k výsledkům měření na náhodně generovaných datech se dal očekávat podobný průběh, ovšem například průměrný čas zpracování pro velikost indexu 1 % je 38,10295 s. Pro 5 % velikosti je to již 274,4822 s, a pro 20 % je průměrná hodnota neuvěřitelných 1101,5701 s (což je zhruba 18,3 minuty). Spolu s velikostí indexu v paměti se zvyšuje i režie udržování a aktualizace kontextu stárnutí pro všechny prvky v paměti. Každé čtení prvku z paměti a každé načtení prvku ze souboru, znamená aktualizaci kontextu pro všechny prvky. To znamená, že pro 100.000× získání featury se musí provést stejný počet i pro aktualizaci kontextu stárnutí.



Obrázek 4.6: Výsledky měření času pro RANDOM a FIFO.

Průměrné hodnoty výsledků pro měření času zpracování jsou znázorněny v grafu 4.6. Kompletní výsledky všech pokusů měření jsou dále k dispozici v příloze G. Výsledky ukazují, že náhodný výběr zahazovaného prvku politikou RANDOM je skutečně náhodný, protože z průměrných hodnot pro různé velikosti nelze vyčíst žádný trend nebo charakteristika, která by blíže popisovala konkrétní chování. Pro politiku FIFO naopak lze detekovat trend, kde se vzrůstající velikosti indexu v paměti, klesá časová náročnost zpracování. Čímž přímo potvrzuje první premisu uvedenou v úvodu kapitoly (viz. 4.3).

Graf 4.7 ukazuje vývoj počtu přepisů v paměti pro politiky RANDOM a FIFO. Politika LRU nebyla do grafu a ani do tabulky kompletních naměřených výsledků zanesena, protože počet přepisů byl pro všechny velikosti paměti a všechna měření stejný - 100.000 ze 100.000 pokusů. Kompletní naměřené hodnoty pro RANDOM a FIFO jsou k nahlédnutí v příloze H. Průměrné výsledky naměřených hodnot ukazují, že pro reálná data se počet přepisů v paměti nijak razantně nemění. Obě politiky vykazují klesající tendenci. Největším skokem je zde pro politiku RANDOM porovnání měření pro 1 % a pro 20 %, které tvoří rozdíl více než 50 přepisů. Zajímavý faktem je poměrně plynulé klesání právě pro politiku RANDOM, zatímco u měření na generovaných datech byly hodnoty zcela náhodné. I v tomto případě



Obrázek 4.7: Výsledky měření počtu přepisů pro RANDOM a FIFO.

měření na reálných datech se jedná o náhodné hodnoty. Jediným vhodným vysvětlením je, že se jedná pouze o „*šťastnou souhru náhod*“. FIFO má velmi jemné klesání které sice potvrzuje druhou premisu v úvodu kapitoly (viz. 4.3), ale na výkonnost inverzního indexu nemá zásadní vliv. Charakteristika pouze potvrzuje podobně jemné klesání v rámci měření času.

4.3.3 Celkové zhodnocení experimentů

V rámci závěrečného hodnocení lze upozornit na korespondenci chování politik LRU a FIFO v měřeních mezi generovanými a reálnými daty. Vysledovaná charakteristika z měření na generovaných datech je pozorovatelná i na měření na reálných datech. Z měřených politik splňuje původní dvě premisy z úvodu kapitoly (viz. 4.3) pouze FIFO. Pro politiku LRU je cena za férový přístup k výběru nejméně používaného prvku bohužel jeho hlavním problémem, protože režie udržování kontextu stárnutí je s rostoucí velikostí indexu v paměti příliš vysoká, než aby bylo možné jeho použití v praxi. Politika RANDOM vykazuje poměrně dobré hodnoty, ale jeho nepředvídatelnost je zároveň majoritní slabinou, což se může ve špatných chvílích negativně projevit. FIFO se zdá být z těchto tří kandidátů jako nejvhodnější volba, protože jeho chování je ustálené a vykazuje nejlepší výsledky za cenu (pouze) částečné férovosti výběru nahrazovaného prvku. Pro potřeby nástroje HLOC a demonstrační webové aplikace více než postačí.

Nejdůležitějšími důvody implementace nového inverzního indexu byly podpora škálování a lepšího hospodaření s pamětí. Původní myšlenka načteného celého inverzního indexu v paměti sice přispívá rychlosti, ale již téměř vylučuje možnosti dalšího škálování. Pokud by měl být systém dále rozšiřován na větší území, s tímto novým indexem by to již bylo

možné. Pokud porovnáme referenční hodnotu původního indexu experimentu pro reálná data a nejlepší průměrný výsledek nového inverzního indexu s FIFO politikou, zjistíme, že nový inverzní index je téměř 100× pomalejší než původní. Z přibližné hodnoty 0,016 s je CPU čas pro zpracování navýšen na zhruba 1,5 s. To je však se získanou možností systém dále škálovat zanedbatelné zpoždění.

Kapitola 5

Návrh systému

Kapitola rozvádí problematiku návrhu celého online systému pro vizuální geo-lokalizaci. Před samotným návrhem bylo třeba definice požadavků na systém, které budou vycházet z předchozí práce [34] a jejích největších problémů popsaných v kapitole 1.1.1. Navrhovaný systém musí splňovat všechny definované požadavky. Celý online systém se skládá z klientské a serverové části webové aplikace, systému Locate a vzájemným propojením. První podkapitola představuje návrh architektury celého systému včetně všech jeho komponent. Následuje podkapitola s návrhem architektury webové aplikace a bližším popisem vybrané architektury. Další část se týká návrhem grafického uživatelského rozhraní, kde jsou vyzdviženy konkrétní požadavky na frontend. Proces návrhu GUI se skládá z navrhnutí kostry frontendu a rozmístění jednotlivých dílčích částí aplikace. Poslední kapitolou je výběr a specifikace vybraných technologií.

5.1 Definice požadavků

Definice požadavků proběhla vytyčením funkčních a nefunkčních požadavků [30]. Výčet funkčních požadavků je zobrazen v tabulce 5.1. Nefunkční požadavky jsou uvedeny v tabulce 5.2. Definování obou skupin doprovázely zejména východiska kapitoly 1.1.1 a zaměření na eliminaci těchto problémů. Velmi důležité je, aby výsledný systém byl co nejrychlejší a aby polo-automatickou detekci horizontu nahradil jiný algoritmus, který by obstojně zvládal plně automatickou extrakci křivky horizontu bez zásahu uživatele. Dále také vybrat prezentaci výsledků s možnostmi účelových stručných vysvětlivek nebo nabídkou s FAQ (nejčastěji kladené otázky). Velký počet obrazovek by mohl být v nejlepším případě omezen pouze na jednostránkovou aplikaci, po které by se uživatel přesouval interaktivně a měl tak vše přímo dostupné bez zbytečných načítání celé webové stránky v návaznosti na průchod různými okny.

Funkční požadavek	**Typ**	**Popis**
Nahrát obrázek	Případ použití	Uživatel může do aplikace nahrát vlastní obrázek z lokálního úložiště nebo z URL adresy.
Výběr obrázku	Případ použití	Uživatel si může vybrat jeden z předpřipravených obrázků.
Detekce křivky horizontu	Systémový	Systém automaticky detekuje křivku horizontu.
Odeslání detekované křivky	Systémový	Systém automaticky odešle detekovanou hranu do podsystému pro geo-lokalizaci křivky horizontu.
Lokalizace detekované křivky	Systémový	Podsystém lokalizuje křivku horizontu a odešle výsledky do systému.
Vizualizace výsledků geo-lokalizace	Systémový	Systém vizualizuje výsledky geo-lokalizace uživateli.
Informace o projektu	Systémový	Systém musí nabízet zobrazení informací o projektu (informace *„About“*).
Informace „Jak to funguje“	Systémový	Systém musí nabízet zobrazení informací o tom, jak funguje detekce a geo-lokalizace křivky horizontu.
Zobrazení nápovědy	Systémový	Systém musí nabízet nápovědu.
Jazykové mutace	Systémový	Systém musí nabízet zobrazení v různých jazycích.

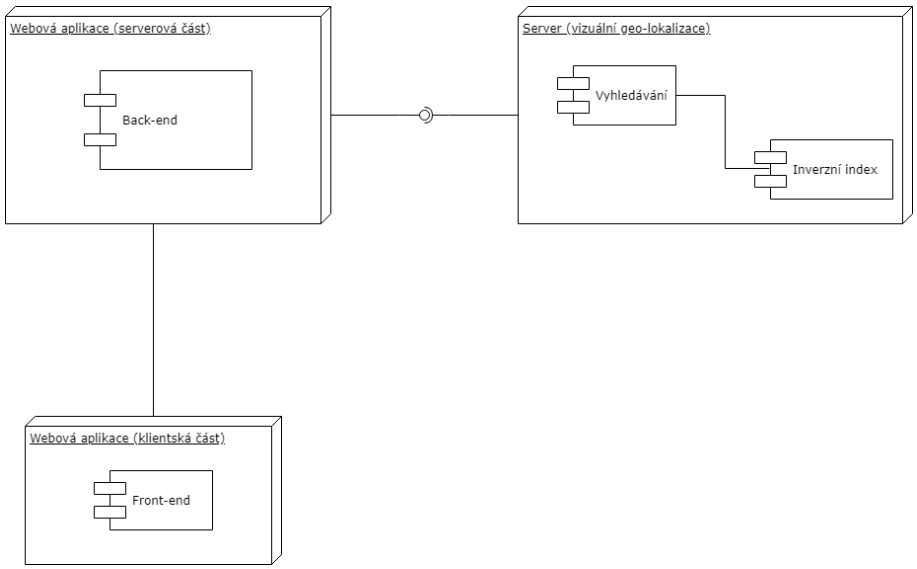
Tabulka 5.1: Funkční požadavky

Kategorie	**Požadavek**	**Popis**
Výkon	Rychlost	Systém musí detekovat křivku horizontu a provádět geo-lokalizaci v co nejrychlejším čase.
Použitelnost	Automatizace	Systém musí detekovat křivku horizontu plně automaticky.
Použitelnost	Přístupnost	Celý systém včetně vizuální stránky musí být maximálně jednoduchý, aby jej dokázal použít i méně zdatný uživatel.
Použitelnost	Srozumitelnost výsledků	Výsledky geo-lokalizace musí být maximálně srozumitelné.
Použitelnost	Přívětivost	Vzhled systému by měl být moderní, přívětivý, minimalistický a schopen konkurovat systémům podobného zaměření.
Spolehlivost	Přesnost	Systém musí detekovat křivku horizontu s co největší přesností.
Spolehlivost	Zabezpečení	Systém musí být ošetřen proti běžným útokům na webové aplikace.
Podporovatelnost	Škálovatelnost	Systém musí být škálovatelný z frontendu i backendu.
Vývojové omezení	Podpora platformy	Systém musí být dostupný ze všech zařízení využívajících internet.
Vývojové omezení	Moderní technologie a přístupy	Systém by měl být vyvíjen na základě nejmodernějších technologií a přístupů, které jsou v současné době k dispozici.

Tabulka 5.2: Nefunkční požadavky

5.2 Návrh architektury systému

Architektura je základní organizace systému, daná jeho komponentami[1], vzájemnými vztahy těchto komponent a jejich vztahy k okolnímu prostředí a principy řídícími její návrh a evoluci [1]. V rámci aspektů architektury [19] pak samotná architektura definuje strukturu, její prvky, vztahy a rozhraní. Dále architektura definuje chování a interakci mezi prvky. Zaměření architektury je na zásadní prvky (s dlouhodobým a trvalým efektem), které mohou ovlivňovat stabilitu a škálovatelnost.



Obrázek 5.1: Struktura architektury online systému

Online systém se skládá ze tří částí: klientská a serverová část webové aplikace a server samotný, poskytující zpracování vizuální geo-lokalizace. Kompletní schéma architektury systému včetně jejich vazeb a komponent je znázorněn na obr. 5.1. Klientská část poskytuje front-end grafické uživatelské rozhraní pro webové aplikace. Tato vrstva obsluhuje veškerou komunikaci s uživatelem. Front-end přijímá interakce uživatele a dále interpretuje a prezentuje výsledky získané ze serverové části webové aplikace, která tvoří back-end. Serverová část aplikace obsahuje logiku a zpracování dat. Back-end aplikace je napojen na implementovaný algoritmus detekce křivky horizontu. Webová aplikace je také svou serverovou částí napojena na server samotný s pomocí definovaného rozhraní. Server má na starosti samotnou geo-lokalizaci na základě detekované křivky horizontu. Vyhledávání podobných horizontů probíhá v rámci inverzního indexu, který má server k dispozici.

5.3 Návrh architektury webové aplikace

S ohledem na IEEE standard 1471-2000 [1] definici o softwarové architektuře, může být webová architektura definována jako základní organizace systému vestavěných komponent

[1] Komponenta představuje modulární část systému, jež zapouzdřuje svůj obsah a v rámci daného prostředí jsou její projevy nenahraditelné. Komponenta definuje svoje chování v podobě poskytovaných a vyžadovaných rozhraní. [38]

(subsystémů), jejich vzájemných vztahů, prostředí a principy řídící jeho vývoj [37]. Webová architektura používá abstrakci a modelování pro zjednodušení a komunikaci komplexních struktur. Architektura přináší framework, který popisuje struktury a poskytuje jednodušší pochopitelnost systému. A dále napomáhá vytvářet most mezi analýzou a implementací.

Pro klientské webové aplikace se nejvíce hodí vrstvené architektury, ze kterých lze jmenovat MVC[2] nebo PCMEF[3] [36]. Vrstvená architektura je pravděpodobně nejpoužívanější, protože je obvykle stavěna na spolupráci s databází. Mnoho největších a nejlepších softwarových frameworků[4] bylo stavěno na této architektuře.

Základní vrstvenou architekturu můžeme rozdělit do tří vrstev [23], které na sobě leží. Ve vrstvených architekturách je základní vrstvou perzistentní úložiště, které můžeme chápat jako kolekci souborů nebo data uložená v databázi. Předtím než je možné k datům přistupovat, manipulovat s nimi nebo je zobrazovat, je nutné zjistit, kde se data nacházejí. Na datové vrstvě leží logická (business) vrstva, která definuje možnosti jak je možné přistupovat k datům a manipulovat s nimi. Dále řídí chování systému a pravidla, jak zpracovat uživatelské požadavky a jaké výstupy navrátit zpět. Logická vrstva je tedy ze spodní části obalena datovou vrstvou a z horní části obalena vrstvou prezentační, která definuje jakým způsobem se aplikace prezentuje uživateli. Prezentační vrstva interaguje s uživatelem a přijímá jeho požadavky, které odesílá na zpracování do logické vrstvy. Požadované výstupy logická vrstva odesílá do prezentační vrstvy, která je interpretuje a vizualizuje uživateli.

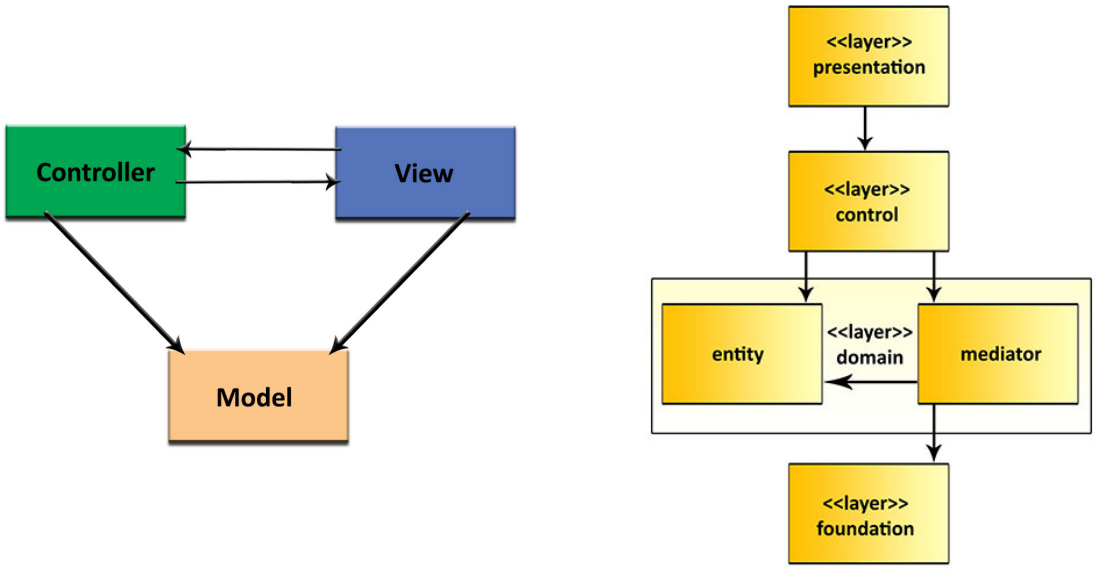
5.3.1 Architektura MVC a PCMEF

MVC

Architektura MVC je s námi již dlouho, avšak nejvíce se uchytil pro webové aplikace. Velké firmy začaly do této architektury investovat a stavět na ní své frameworky[5]. Existují však i skupiny vývojářů a dobrovolníků, kteří vytvořili open-source frameworky[6] postavené na MVC. Tato architektura má čtyři hlavní výhody, kde se mezi první řadí možnost samostatného vývoje uživatelského rozhraní, řídící logiky a datové logiky. Další výhodou je, že změny jedné části téměř neovlivní ostatní. Dále možnost použít Model s několika různými View (několika uživatelskými rozhraními) nebo také úplně bez View (např. pro testovací účely); Poslední výhodou je jednoduchost, přímočarost a škálovatelnost architektury se hodí pro vývoj menších webových aplikací (více než PCMEF). Tato architektura je rozdělena na tři logické části, tak aby bylo možno je upravovat samostatně a aby byl případný dopad na ostatní části co nejmenší. Logika aplikace a zpracování toku událostí má na starosti vrstva Controller. View definuje způsoby prezentace dat uživateli a tedy zajišťuje vykreslování uživatelského rozhraní. Poslední vrstvou je Model, která pokrývá business logiku aplikace a práci s daty. Jednotlivé vrstvy spolu mají konkrétní vazby (viz. obr. 5.2). View vrstva prezentuje uživateli data z Modelu za pomocí uživatelského rozhraní, které zadal vykreslit Controller. Uživatel může interagovat s aplikací kliknutím na nějaký prvek v aplikaci a odesláním požadavku, který zpracuje vrstva Controller. Tato vrstva pak případně upraví nebo dá načíst nová data za asistence Modelu.

[2] Model-View-Controller
[3] Presentation, Control, Mediator, Entity, Foundation
[4] Například - Java EE, Drupal a podobné.
[5] Zend Framework (http://www.zend.com/), ASP.NET (https://www.microsoft.com/cs-cz/)
[6] Laravel (https://laravel.com/), Symfony (https://symfony.com/), CakePHP (https://cakephp.org/), Nette Framework (https://nette.org/cs/)

MVC PCMEF



Obrázek 5.2: Struktura architektury MVC a PCMEF

Architektura PCMEF

Tato softwarová architektura je hojně využívaná v oblasti objektově orientovaného programování. Pro tvorbu větších robustních webových systémů je vhodnější než architektura MVC, protože využívá hierarchii vrstev, které mají volnější vazby. PCMEF je rozdělena dle zodpovědností do celkem pěti vrstev (viz. obr. 5.2). Vrstvy mohou obsahovat další balíčky. Presentation odpovídá za ovládání grafického uživatelského rozhraní a přijímá interakce od uživatele. Aplikační logiku zapouzdřuje Control, který vyhledává informace ve vrstvě Entity. Dále může Control iniciovat interakci mezi vrstvami Mediator a Foundation a zajistit tak nahrání nových dat do vrstvy Entity. Mediator se tedy stará o komunikaci mezi Entity a Foundation. Entity jsou business objekty nahrané v paměti. Poslední vrstva Foundation zajišťuje (low-level) komunikaci se zdrojem dat (např. databází). Samotná architektura PCMEF staví na návrhových vzorech jako jsou fasáda, abstraktní továrna, řetěz odpovědnosti, pozorovatel a další.

5.3.2 Výběr architektury

Vzhledem k funkčním i nefunkčním požadavkům jsem vybral architekturu MVC. Výsledná webová aplikace by měla být minimalistická a jednoduchá pro uživatele. Využití entitních tříd by v tomto případě bylo minimální. Silnější vazby MVC oproti PCMEF nejsou překážkou. Podpora škálovatelnosti je také jedním z velkých důvodů pro přiklonění k této variantě. Pro vývoj aplikace postavené na MVC je možno použít frameworku, který již implicitně řeší běžné problémy nebo například další z nefunkčních požadavků - ošetření proti běžným útokům na webové aplikace. I přesto, že je aplikace na detekci křivky horizontu napsána v Javě a pro PCMEF se dokonale hodí použití Java EE[7] (mohla by se tedy detekce spouštět přímo z kódu webu), je podle mého lepší varianta s PHP a MVC. Java EE vyžaduje po-

[7] Java Platform, Enterprise Edition
(http://www.oracle.com/technetwork/java/javaee/overview/index.html)

měrně velké výpočetní zdroje hlavně z hlediska procesorového času a paměti. MVC aplikace na PHP je méně náročná a spuštění binárního souboru detekce křivky horizontu již není takovým problémem.

5.4 Návrh uživatelského rozhraní webové aplikace

Uživatelské rozhraní je kombinace prostředků či technologií, která umožňuje uživateli komunikovat s programem. Rozhraní komunikace můžeme definovat v grafické formě, pokud se jedná o návrhem uživatelského rozhraní pro webový online systém, kde se jedná především o textové a grafické prvky a jejich rozmístění. Tyto prvky budou sloužit uživateli k pohodlné manipulaci se systémem. Prvky uživatel využije také k zadávání dat do aplikace, a po zpracování v systému prvky zobrazí uživateli výsledky operací. GUI by se tedy v prvé řadě mělo orientovat na uživatele. Navrhování s tímto ohledem je označováno jako UX[8] design [46], což je metodika založena na hlavních principech:

- Design: je vizuální forma, ale i například to, jakým způsobem navrhovaná aplikace funguje. Cílem designu je důraz na účelnost propojení estetiky a funkčnosti navrhovaného systému.
- Použitelnost: jedna z částí, vytvářející uživatelský prožitek při práci s aplikací. Zde je primárním zaměřením odstraňování překážek či nedostatků v ovládání a postupné vylepšování na základě uživatelské zpětné vazby.

Cílem User Experience Design je dosažení co nejlepšího sjednocení několika disciplín, jakými je design, použitelnost, ale také informační architektury a interakce uživatele s aplikací. Pokrývá tedy nejen formu webové aplikace, ale také její obsah a chování.

5.4.1 Návrh kostry GUI

Při navrhování kostry grafického uživatelského rozhraní byl kladen největší důraz na splnění všech funkčních i nefunkčních požadavků definovaných v kapitole 5.1. Z funkčních požadavků jsou to především: Nahrát obrázek, Výběr obrázku, Vizualizace výsledků geolokalizace, Informace o projektu, Informace „Jak to funguje“, Zobrazení nápovědy a Jazyková mutace. Návrh kostry se týká těchto nefunkčních požadavků: Přístupnost, Přívětivost a Podpora platformy.

Vytvořený návrh je k dispozici v příloze B a byl vytvořen za pomoci programu Pencil[9]. Aplikace byla navržena jako jednostránková (viz. obr. B.1), kde je veškerý obsah zobrazen pod sebou na jediné stránce ve vizuálně oddělených horizontálních panelech. V úvodním panelu je uvítání návštěvníka a tlačítka pro rychlou navigaci na vyzkoušení vizuální geolokalizace nebo na informace o projektu. V levém horním rohu je k perzistentně dispozici známá ikona pro menu (viz. obr. B.2), které se vysune z levé části webu a nabídne uživateli nabídku jednotlivých obsahových panelů. Kliknutím na tlačítka v menu nebo v úvodním panelu bude uživatel automaticky animovaně přesunut na požadovaný obsah. V menu bude také možnost zvolení jazykové mutace. Pro návrat na úvodní panel je k dispozici tlačítko v pravém dolním rohu. Obsahové panely jsou v kostře seřazeny: úvodní panel, panel s demonstrací vizuální geo-lokalizace (včetně zobrazení výsledků), panel s informacemi o pro-

[8] User Experience
[9] Pencil Project (https://pencil.evolus.vn/)

jektu a poslední panel s informacemi jakým způsobem probíhá vizuální geo-lokalizace a jak aplikace funguje.

5.5 Výběr technologií

V této kapitole jsou představeny vybrané technologie týkající se výhradně navrhované webové aplikace. Technologie byly vybrány s ohledem na (pokud možno) všechny funkční a nefunkční požadavky podrobně popsané v kapitole 5.1. V první řadě je představen webový framework. Podle zvoleného typu architektury (kap. 5.3.2) - MVC, bylo důležité zvolit vhodnou platformu, která je na této architektuře vystavěna. Framework by se také měl zaměřit v rámci nefunkčních požadavků hlavně na použití moderních přístupů, bezpečnost, škálovatelnost a rychlost. Další podkapitola se týká vhodné selekce knihovny pro robustní a responzivní front-end aplikace. Tato knihovna by měla být zaměřena na bezproblémovou implementaci funkčních požadavků. Z požadavků nefunkčních je důležité, aby knihovna naplňovala požadavky na přístupnost, přívětivost a podporu platformy. Kapitolu zakončují malé pluginy a knihovny jejichž zodpovědnost náleží hlavně na zkvalitnění služeb uživatelům (v rámci uživatelské přívětivosti). Některé z knihoven jsou vybrány za účelem doplnění funkcí, které základní webový framework a front-end nenabízí. Tyto knihovny mají primární zaměření na zbylé funkční a nefunkční požadavky[10].

5.5.1 Nette Framework

Nette [22] je multiplatformní open-source MVC framework pro tvorbu webových aplikací a je tvořen souborem samostatných komponent v PHP 7. Autorem tohoto frameworku, jež je dostupný pod GNU GPL, je David Grudl a další vývojáři z Nette Foundation. Nette klade důraz na zabezpečení, vzhledem k nativní implementaci prevence bezpečnostních děr v podobě prevence proti útokům typu: XSS, CSFR, session hijacking[11], session fixation[12] a další. Framework je postaven na nejmodernějších technologiích, ze kterých lze jmenovat zejména: AJAX / AJAJ, Dependency Injection, SEO, DRY[13], KISS[14], Web 2.0 a *cool* URL. V široké škále nabídky modulů můžeme nalézt ladící nástroje, šablonový systém, unit testing a mnoho dalších pluginů a rozšíření.

5.5.2 Bootstrap

Moderní knihovnou pro podporu vývoje robustního a responzivního grafického uživatelského rozhraní je knihovna Bootstrap [32]. Knihovna je open-source a poskytuje sadu nástrojů pro vývoj HTML, CSS a JavaScript. Od jeho prvního spuštění v srpnu 2011 nabral na velké popularitě a postupně se vyvinul do CSS řízeného projektu zahrnující JavaScriptové pluginy a ikony, které spolupracují s formuláři a tlačítky [41]. Základními stavebními kameny pro vývoj front-endu poskytuje Bootstrap formou kontejnerů. Kontejnery je možné nastavit s pevnou šířkou nebo responzivní a mohou být také kaskádově vnořeny do dalších. Dalším

[10] Například: Nahrát obrázek, Výběr obrázku, Vizualizace výsledků a Jazykové mutace.

[11] Session hijacking - útok, který zneužívá ukradené HTTP cookie uživatele, aby útočník získal neoprávněný přístup k informacím nebo službám webového serveru.

[12] Session fixation - využití bezpečnostní díry v systému umožňující útočníkovi se zafixovat na identifikátor sezení (session ID) nic netušícího uživatele.

[13] Don't Repeat Yourself - princip vývoje softwaru zaměřený na snižování počtu duplicitních informací.

[14] Keep It Short and Simple - princip návrhu, kdy je největší snahou držet třídy, metody a kompletní kód krátký a jednoduchý.

základním prvkem je Grid systém, což je využití řady kontejnerů a sloupců (ukázka kódu 5.1 a jeho realizace 5.3) pro vytvoření schématu obsahu (layout). Grid systém je vytvořen společně s flexbox[15] a je plně responzivní. Boostrap umožňuje velmi rychlý vývoj na stabilní platformě.

```
<div class="container">
    <div class="row">
        <div class="col-sm">
            One of three columns
        </div>
        <div class="col-sm">
            One of three columns
        </div>
        <div class="col-sm">
            One of three columns
        </div>
    </div>
</div>
```

Kód 5.1: Ukázka responzivního kontejneru se třemi sloupci v Boostrap.

Obrázek 5.3: Ukázka responzivního kontejneru vytvořeného kódem 5.1.

5.5.3 Další knihovny a pluginy

Jak už bylo naznačeno v úvodu této kapitoly (viz. 5.5), tento seznam dalších knihoven a pluginů se týká výhradně doplnění některých funkcí nebo vlastností, které základní verze webového frameworku nebo front-end knihovny nenabízí. Tyto knihovny se z větší části zaměřují na zvýšení přívětivosti pro uživatele grafickým různými grafickými a animovanými efekty. Případně doplňují možnosti webové aplikace z backendu formou pluginů webového frameworku.

Kdyby/Translation

Kdyby/Translation[16] je pluginem pro Nette a rozšiřuje tak základní framework o snadnou a pohodlnou podporu lokalizace. Umožňuje tak vývojáři vytvářet vícejazyčné weby za pomoci konfiguračních souborů, do kterých se ukládají jazykově specifické řetězce. V konfiguračním souboru Nette stačí definovat jaké jazyky mají být k dispozici uživateli a plugin se již sám postará o automatické rozpoznání jazyka z URL adresy a nastavení správných ře-

[15] flexbox (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox)

[16] https://github.com/kdyby/Translation

těžců do šablon aplikace. Knihovna mimo jiné umožňuje také podmíněnou selekci řetězců[17] nebo také zvolení správného tvaru řetězce v případě číslovek[18] (pokud je to v rámci daného jazyka zapotřebí).

Animate Scroll

JavaScriptový plugin Animate Scroll[19] poskytuje uživatelsky přívětivou animaci při přesunech na webové stránce s pomocí odkazů na kotvy. Rozložení návrhu webové aplikace na jednostránkovou už od začátku počítá s tím, že se uživatel bude přesunovat po stránce „*scrollováním*“ nebo za pomocí odkazů v menu, které právě budou odkazovat na kotvy jednotlivých obsahových panelů v aplikaci. S tímto pluginem se bude moci uživatel posunovat po stránce atraktivním pohybovým efektem. Mimo jiné je také plugin optimalizován pro použití na tabletech a mobilních zařízeních.

Bootstrap Lightbox

Galerie Lightbox je na světě již velmi dlouho snad ve všech variacích. Modul Bootstrap Lightbox[20] je převzatý původní nápad implementovaný za použití knihovny Bootstrap. Bootstrap mimo jiné podporuje kromě klasických obrázků i například YouTube videa.

Side Menu

Než knihovnou nebo pluginem je Side Menu[21] spíše souborem CSS stylů s několika řádky JavaScriptu. Modul obstarává navrženou funkci „*schovaného*“ bočního menu, které se zobrazí pouze na základě tlačítka v levém horním rohu.

Konva.js

Konva[22] je HTML5 a JavaScriptový framework pracující s grafikou v Canvas prvku a rozšiřuje jeho 2D kontext o interaktivitu s desktopovými a mobilními aplikacemi. Konva je robustní a komplexní framework disponující vlastním API a poskytuje vysoce výkonné animace, přechody, vrstvy, filtry a obsluhami eventů. Použití Konvy v aplikaci bude stejné jako v [34], tedy s její pomocí bude možné uživateli vykreslit detekovanou křivku horizontu uživateli na jejímž základě proběhne vizuální geo-lokalizace.

jQuery Upload File

Utilita jQuery Upload File[23] poskytuje možnost snadno přidat do aplikace uživatelsky přívětivé a pohodlné nahrávání souborů. Plugin disponuje vestavěným automatickým odesláním AJAX HTTP požadavku s nahraným souborem a poskytuje mnoho možností pro splnění konkrétních potřeb. Tento plugin také nativně podporuje drag-n-drop mechaniku, multi-file upload a další. Plugin by měl jednoduše vyřešit otázku nahrání fotografie uživatelem na server.

[17] Na základě splnění nějaké podmínky rozhodne, který z definovaných řetězců má zobrazit.
[18] Podle hodnoty doplní správný tvar slova (např. pro hodnotu „1“ doplní řetězec „kus“; pro hodnotu „4“ doplní řetězec „kusy“; atp.).
[19] https://github.com/mpalpha/animate-scroll
[20] https://github.com/ashleydw/lightbox
[21] https://tympanus.net/codrops/
[22] https://konvajs.github.io/
[23] https://github.com/hayageek/jquery-upload-file

Kapitola 6

Implementace webové aplikace

Výběr architektury v 5.3.2, návrh GUI webové aplikace v 5.4.1 a vybrané technologie v 5.5 se staly základními kameny pro implementaci webové aplikace. Implementovaná aplikace by měla splňovat všechny požadavky definované v 5.1. Implementace probíhala v jazyce PHP 7.2.5 pro serverovou část aplikace. Klientskou část pak definují jazyky HTML 5, CSS 3 a JavaScript (nadstavba v podobě jQuery[1]). Celá webová aplikace byla nadefinována jako jednostránková s veškerým obsahem řazeným pod sebe. Klientská část aplikace by měla se serverovou částí komunikovat pouze za pomocí AJAX požadavků-odpovědí, což jistě zvýší přívětivost pro uživatele. Všechny události v aplikaci v rámci demonstrace vizuální geo-lokalizace, by měly být automaticky zřetězeny a volány postupně. Celý proces je zároveň podrobně popisován uživateli, který za všech stavů zpracování ví co se právě děje. Pro některé hůře chápající stavy nebo pojmy jsou k dispozcii vysvětlivky, které se zobrazí najetím na daný nápis. Jediným nutným důvodem pro znovu-načítání stránky bude změna jazykové mutace aplikace.

Základem webové aplikace bude framework Nette zajišťující základní funkcionalitu. Nette potřebuje ke spuštění konfigurační soubor **config.neon**, ve kterém se může specifikovat prezentér pro chyby, mapování prezentérů, nastavení session, definice modelů a routerů, další parametry a registrace dalších rozšíření. Pro rozšíření o jazykové mutace jsou všechny lokalizované řetězce uloženy ve vlastním souboru dle užitého jazyka. Příklad schématu takového souboru a jeho následné použití v šabloně je znázorněn v ukázce kódu 6.1. Webová aplikace byla lokalizována do angličtiny. S pomocí externích překladatelů[2] ještě navíc do jazyků: slovenština, polština a portugalština.

```
// Definice retezce v lokalizacnim souboru
app:
       header: ''System for Visual Geo-localization''

// Pouziti retezce v sablone
{_app.app.header}
```

Kód 6.1: Ukázka definice řetězce pro lokalizaci a použití v šabloně.

První podkapitola rozebírá implementaci základní webové aplikace bez lokalizace křivky horizontu a vizualizace výsledků. Navazující podkapitola již popisuje integraci programu pro

[1] jQuery v3.2.1 (https://jquery.com/).
[2] Seznam externím korektorů a překladatelů je uveden v poděkování práce.

detekci křivky horizontu do webové aplikace. Poslední podkapitola pojednává o propojení webové aplikace se serverem a zpracováním vizuální geo-lokalizace. A dále formy vizualizace vrácených výsledků.

6.1 Kostra webové aplikace

Back-end webové aplikace se skládá primárně z modelů. V této implementaci byl implementován aplikační model, který bude primárně zastřešovat detekci křivky horizontu, a tím i spuštění programu z kapitoly 3, získání výsledků a jejich zpracování pro možnost vhodné vizualizace výsledků uživateli. Další hlavní činností tohoto modelu bude obsluha komunikace se serverem.

Logika webové aplikace je implementována v prezentérech. Pro aplikaci byl definován základní prezentér jako abstraktní třída, který udržuje injektované kontexty pro lokalizátor a aplikační model. Další funkce základního prezentéru je získání HTTP požadavku. Pokud se jedná o AJAX požadavek, pak metoda vrátí kompletní kontext HTTP požadavku, v opačném případě je odeslána AJAX odpověď, že se nejedná o platný požadavek na aplikaci. Základní prezentér obstarává také zpracování dat, které se mají odeslat do klientské části, a obsluhu samotného odeslání HTTP odpovědi.

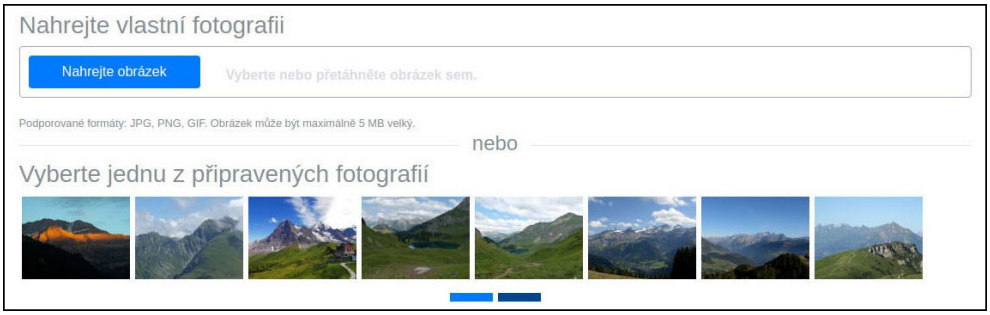
Od základního prezentéru dědí třída aplikačního prezentéru, která řídí celou aplikaci. Aplikační prezentér přijímá požadavky od klientské části, volá příslušné metody v aplikačním modelu a výsledky odesílá jako odpovědi do klientské části. Základní funkcionalitou v kostře webové aplikace je obstarání požadavku na nahrání obrázku uživatelem. Uživatel nahraje obrázek, pro který se vygeneruje náhodné číslo jako identifikátor aktuálního sezení (vyhledávání) uživatele. Z názvu obrázku jsou odstraněny nežádoucí znaky a mezery. Případné speciální znaky abeced jazyků jsou převedeny na základní ASCII písmena. Obrázek je následně přesunut do složky na serveru, kde jsou dočasně ukládány právě vyhledávané fotografie. Aplikační prezentér následně zasílá klientské části odpověď s úspěchem či neúspěchem operace.

Front-end neboli klientská část aplikace je tvořena ze šablon napsaných v Latte[3], což je šablonovací systém dostupný se základní distribucí Nette. Základ aplikace tvoří šablona pro celkový *layout* aplikace. Tato šablona obsahuje HTML hlavičku linkující všechny potřebné CSS soubory se styly. Dále jsou v hlavičce uvedeny základní meta tagy o webové aplikaci. Layout dále obsahuje definici výsuvného menu a jeho obsah. Následuje prázdný blok pro obsah, do kterého se právě vkládá šablona s veškerým obsahem webu. V layoutu jsou pak ještě definovány například patička, kód pro „*Scroll-to-top*“ tlačítko nebo „*Progress bar*“, který plovoucí formou znázorňuje scrollovanou pozici uživatele na stránce. Layout ukončuje linkování všech JavaScriptových kódů a knihoven užitých v aplikaci.

Od začátku byla snaha veškerý vlastní JavaScriptový kód a funkce koncipovat do odděleného souboru. Nicméně části JavaScriptu se neobešly bez umístění přímo v hlavní šabloně pro obsah. Úvod šablony tak obsahuje definici několika JavaScriptových proměnných s lokalizovanými řetězci použitými právě v kódu v externím souboru. Dále je zde kód pro obsluhu nahrání fotky pluginem jQuery Upload File (viz. 5.5.3). Ukázka implementace nahrání a výběru obrázku je znázorněna v 6.1. Samotný kód by mohl být v externím souboru, ale lokalizační řetězce, které plugin Kdyby/Translation (viz. 5.5.3) podle aktuálního jazyka distribuuje do front-endu, jsou k dispozici právě pouze v šablonách Latte. Jejich viditelnost je tedy omezená. Obsahový HTML kód v šabloně je pak tvořen kaskádovými definicemi tvo-

[3] https://latte.nette.org/cs/

řenými právě kontejnery, řádky a sloupci, tak jak definuje Bootstrap (viz. 5.5.2). Postupně je tak definován kód pro úvodní záhlaví s představením aplikace. Dále definice panelu pro demonstrační funkcionalitu vizuální geo-lokalizace. Šablona je zakončena třemi panely pro informace o projektu, popis jak aplikace funguje a panel pro umístění podporovatelů.



Obrázek 6.1: Ukázka implementace nahrání a výběru obrázku.

Jak již bylo zmíněno, funkční vlastní JavaScriptový kód je koncipován do souboru, aby tak byl oddělen od šablony. Pro základní kostru aplikace je zde definována funkcionalita pro animované přesuny po kotvách odkazů (*smooth scroll*). Dále je zde napojena funkce pro scroll-to-top, a také obsluha progress baru.

6.2 Integrace lokalizace křivky horizontu

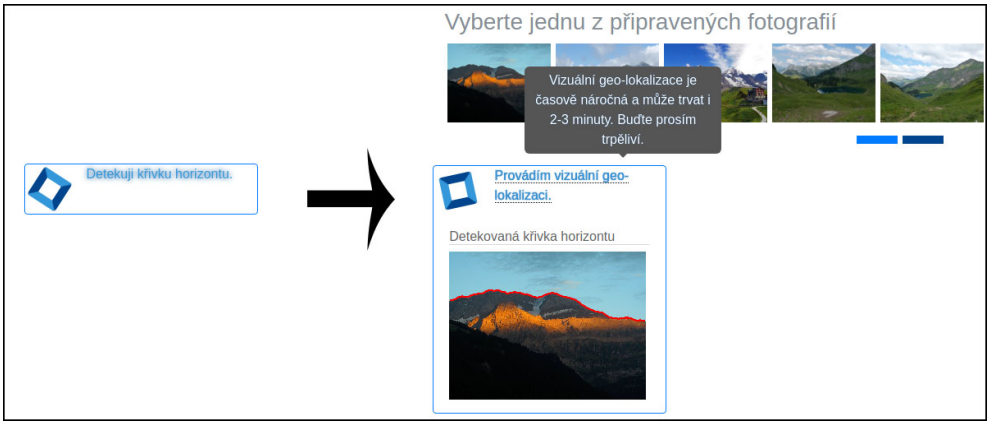
Aplikace pro detekci křivky horizontu je nezbytná pro webovou aplikaci, protože až na základě detekované křivky může proběhnout vizuální geo-lokalizace. Po úspěšném nahrání vlastní fotografie nebo uživatelově výběru předem nachystaných fotografií je z klientské části odeslán požadavek na server s informací o ID obrázku názvu obrázku. Prezentér serverové části požadavek přijme a získá ID a název obrázku. Prezentér zavolá kontrolu v modelu zda-li takový obrázek na serveru existuje a spustí detekci křivky horizontu.

Detekce křivky horizontu je integrovaná do aplikačního modelu, kde se aplikace spouští jako externí program funkcí *shell_exec(run_command)*. Ve spouštěcím příkazu je nutné uvést cestu k *LIBSTDCXX.so.6*, cestu k OpenCV balíkům a samozřejmě nutné parametry spuštění programu. Správně detekovaná křivka horizontu bude produkována aplikací jako řetězec souřadnic. Modelová metoda, která celý proces zastřešuje výsledek aplikace detekce křivky zkontroluje regulárním výrazem a dále je získána hodnota *Field of View* z obrázku. Metoda končí parsováním řetězce s výsledkem do formy, kterou je možné přímo použít pro vizualizaci uživateli. Zpracování se vrátí do prezentéru, který informace o obrázku a detekované křivce uloží do Cookies a odešle je v rámci odpovědi do klientské části.

Klientská část přijme odpověď a ze získaných dat je vytvořen objekt knihovny KonvaJS (viz. 5.5.3), který vykreslí uživateli miniaturu obrázku, a v něm vyznačenou detekovanou křivku horizontu. Ukázka informování uživatele o právě zpracovávané fázi aplikace je znázorněna v obr. 6.2.

6.3 Vizuální geo-lokalizace a vizualizace výsledků

Po vykreslení detekované křivky se rovnou automaticky invokuje požadavek na server pro vizuální geo-lokalizaci. Prezentér opět přebere zodpovědnost a stáhne si z Cookies potřebné



Obrázek 6.2: Ukázka informání uživatele o činnosti aplikace a vykreslení detekované křivky horizontu.

informace z předchozího zpracování lokalizace křivky horizontu. Následně zavolá modelovou metodu, která z parametrů vytvoří pole, které převede na JSON formát a zavolá metodu komunikující se serverem. Komunikace se serverem zůstala stejná jako v případě [34], proto i podoba metody je velmi podobná.

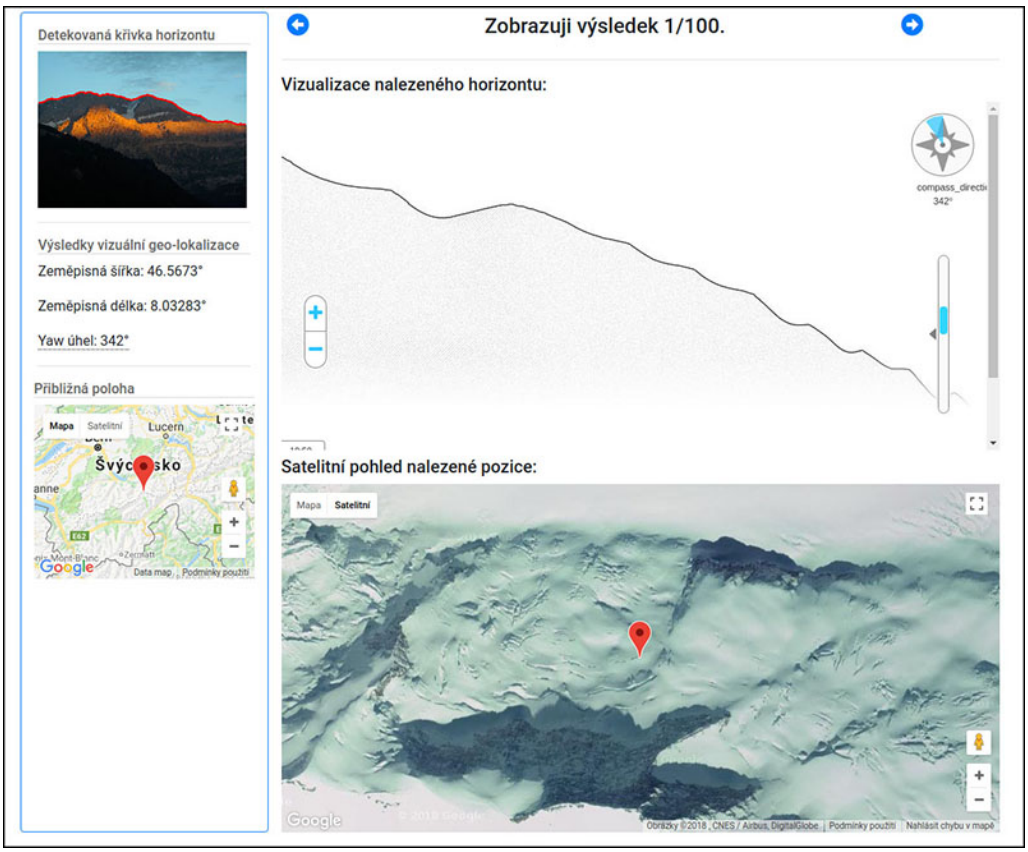
Oproti [34] však nastala jedna zásadní změna. Již se na server neodesílá generovaný obrázek se segmentovaným popředím/pozadím, ale zasílá se pouze pole souřadnic křivky horizontu. Ukázka JSON formátu odesílaného na server je k nahlédnutí zde 6.2. JSON obsahuje položku *coordinates*, kde je zakódována křivka horizontu. Vždy hlavním indexem v tomto poli je X-ová souřadnice. Pod tímto indexem je umístěno pole, ve kterém může být jedna nebo dvě Y-ové souřadnice. Jedna Y-ová souřadnice značí jediný bod. Dvě Y-ové souřadnice značí kolmý spojitý segment křivky horizontu, kde její souřadnice jsou definovány právě intervalem těchto dvou Y-ových hodnot.

Server s implementovaným novým inverzním indexem vyhledá nejlepší výsledky a vrátí je stejným způsobem a ve stejném formátu jako v [34]. Data tedy přijdou ve formátu base64. Po dekódování se musí výsledky rozbalit například pomocí funkce *zlib_decode()*. Z rozbalených dat je již možné dekódovat JSON do asociativního pole, které je vráceno prezentéru.

```
{
        ''coordinates'': {
                ''1'': [ 100, 20 ],
                ''2'': [ 35, 40],
                ''3'': [ 15 ],
                ...
        },
        ''fov'': 68.46,
        ''width'': 240,
        ''height'': 180
}
```

Kód 6.2: Ukázka JSON formátu odesílaného na server.

Prezentér převezme výsledky a odešle HTTP odpověď do klientské části. JavaScriptová metoda obdrží odpověď s výsledky vizuální geo-lokalizace. Uživateli je v danou chvíli zobrazen vždy pouze jeden výsledek. Ukázka s vizualizecemi výsledku vizuální geo-lokalizace je k dispozici v obr. 6.3. V levém panelu se zobrazí zeměpisné souřadnice výsledku a *Yaw* úhel. Pod těmito informacemi se vygeneruje malá Google mapa s klasickou orientační mapou přiblíženou na úroveň viditelnosti států a větších měst. Mapa je vycentrována na nalezené souřadnice výsledku v mapě je také umístěn *marker* zvýrazňující, kam souřadnice ukazují. Do hlavního panelu je pak nejdříve umístěn panel ze služby PeakFinder, který zobrazuje horizont z místa, zaměřeného na souřadnice výsledku a posunutý o Yaw úhel. PeakFinder by tak měl zobrazovat viditelný horizont z místa vráceného výsledku. Pod panelem s panoramatem z PeakFinderu je ještě jedna vizualizace výsledku, a to opět z Google map. Panel zobrazuje satelitní detailní snímek zaměřené pozice opět s markrem. Pokud mají Google mapy pro dané místo k dispozici materiály, je detailní pohled nakloněn o 45 stupňů. Uživatel si tak může na jednom místě ve třech panelech prohlédnout jak vypadá horizont z daného místa, jak detailně a reálně místo vypadá, a kde na mapě se přibližně nachází (v jakém státě, u jakého města, apod.). Pokud by si uživatel chtěl prohlédnout i další výsledky (celkem jich server vrací 100), pak jsou nad hlavními panely zobrazeny šipky, pomocí kterých se může uživatel přesouvat mezi jednotlivými výsledky. Při přesunu na další výsledek jsou automaticky načteny i nové mapy a horizonty, dle aktuálně prohlíženého výsledku. Pokud uživatel zatouží si hned nechat lokalizovat další fotku, pak stačí jen nahrát fotku novou (případně vybrat z připravených) a celý proces může opět bez problémů pokračovat.



Obrázek 6.3: Ukázka vizualizace výsledků vizuální geo-lokalizace.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit online systém pro vizuální geo-lokalizaci v přírodním prostředí. Online systém je tvořen ze serveru, který provádí vizuální geo-lokalizaci a z webové aplikace poskytující uživatelům demonstraci fungování systému. Pro vývoj webové aplikace bylo třeba analyzovat předchozí aplikaci pro vizuální geo-lokalizaci [34] a zjistit největší problémy, se kterými se aplikace potýká. Nová webová aplikace měla být explicitně zaměřena na eliminaci těchto problémů definovaných v 1.1.1. Webová aplikace by rovněž měla splňovat nároky na jednoduchost, snadnou použitelnost, interaktivitu, rychlost a škálovatelnost (viz. 5.1). Kromě webové aplikace bylo třeba se také zaměřit na úpravu serveru, který nemohl být dále škálován kvůli neoptimální strategii udržování celého inverzního indexu v paměti. Bylo třeba serveru implementovat nový inverzní index s cachovací politikou, který umožní další škálování systému. Pro detekci křivky horizontu bylo třeba implementovat algoritmus, který obstojně a automaticky zvládne detekovat křivku horizontu z fotografie.

7.1 Zhodnocení dosažených výsledků

Implementoval jsem velmi jednoduchou a atraktivní jednostránkovou webovou demonstrační aplikaci pro vizuální geo-lokalizaci. Aplikace implicitně podporuje jazykové mutace hned v několika jazycích. V této aplikaci se může uživatel pohybovat jednoduše skrze výsuvné menu nebo pomocí scrollování. Aplikace nabízí vizuální geo-lokalizaci vlastního obrázku nebo může uživatel vybrat z již připravených fotografií. Po výběru obrázku proběhne plně automatický proces skládající se z detekce křivky horizontu a vizuální geo-lokalizace, přičemž je uživatel průběžně informován o činnosti aplikace. Výsledky geo-lokalizace jsou uživateli prezentovány formou několika pohledů na lokalizované místo.

Dle mého názoru implementovaná webová aplikace odstranila všechny problémy specifikované v 1.1.1. V [34] je uvedeno, že rychlost zpracování jen pouhé vizuální geo-lokalizace je 150–300 s. V této implementaci se rychlost zpracování pohybuje v intervalu 20–180[1] s. Poloautomatická detekce křivky horizontu z [34] byla nahrazena automatickou implementací s velmi dobrou přesností. Nevhodná vizualizace výsledků z [34] byla odstraněna a výsledky se nyní vizualizují formou dvou interaktivních map a panelu s vykresleným horizontem. Velký počet obrazovek z [34] byl eliminován na pohou jednu stránku v současné implementaci. Nová webová aplikace také disponuje moderním a atraktivním designem než [34].

Dále jsem implementoval program pro detekci křivky horizontu podle [24]. V implementaci jsem natrénoval SVM, jehož model jsem podrobil evaluaci a testování na datové sadě

[1] Čas může být ovlivněn například tím, že pro hledaný horizont není v paměti nahrána ani jedna featura.

GeoPose3K [12]. Celkem jsem pro tento algoritmus a natrénovaný model provedl několik experimentů pro zjištění případných možností vylepšení.

Poslední implementovanou částí byl nový inverzní index pro server provádějící vizuální geo-lokalizaci. Inverzní index si nyní ukládá své featury a ID dokumentů do hierarchicky umístěných souborů na pevném disku. Pro běh indexu jsem implementoval tři různé cahovací politiky, které se starají o řízení a správu omezeného prostoru inverzního indexu v operační paměti. Pro cachovací politiky jsem provedl sérii experimentů pro vyhodnocení, která politika bude nejoptimálnější.

7.2 Další vývoj projektu

Webová aplikace může být dále vyvíjena z pohledu možných dalších možností vizualizace výsledků. S výsledky jsou vráceny i souřadnice nalezených horizontů a souřadnice hledané křivky horizontu. Pokud by se povedlo například do panoramatu PeakFinderu zakomponovat i vrácené souřadnice výsledků, mohl by se horizont přímo vyznačit v panelu PeakFinderu. Webová aplikace by také mohla být řádně otestována pro mobilní telefony. Boostrap nativně řeší téměř všechny problém responzivity, avšak pro velmi malá rozlišení by mohly být některé prvky a panely nevhodně uspořádané. Aplikace by tak mohla být otestována a případně stylově upravena pro použití na všech typech a velikostech obrazovky mobilních telefonů.

Jedním ze směrů dalšího vývoje projektu je pro inverzní index s featurami a ID dokumentů implementovat optimalizace popsané v [16]. Další možností může být implementace další cachovací politiky a otestování oproti aktuální FIFO politice. Bylo by tak teoreticky možné nalézt ještě rychlejší a optimálnější politiky.

Literatura

[1] ANSI/IEEE 1471-2000 - IEEE Recommended Practice for Architectural Description for Software-Intensive Systems [online]. 2001 [cit. 2018-01-20].
URL https://standards.ieee.org/findstds/standard/1471-2000.html

[2] OpenCV [online]. 2018 [cit. 2018-01-18].
URL https://opencv.org/

[3] Google Maps Platform [online]. https://cloud.google.com/maps-platform/, 2018 [cit. 2018-05-21].

[4] Ahmad, T.; Bebis, G.; Nicolescu, M.; aj.: An Edge-Less Approach to Horizon Line Detection. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, Prosinec 2015, s. 1095–1102, doi:10.1109/ICMLA.2015.67.

[5] Ahmad, T.; Bebis, G.; Regentova, E. E.; aj.: *A Machine Learning Approach to Horizon Line Detection Using Local Features*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ISBN 978-3-642-41914-0, s. 181–193, doi:10.1007/978-3-642-41914-0_19.
URL http://dx.doi.org/10.1007/978-3-642-41914-0_19

[6] Ahmad, T.; Campr, P.; Čadík, M.; aj.: Comparison of Semantic Segmentation Approaches for Horizon/Sky Line Detection. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Institute of Electrical and Electronics Engineers, Květen 2017, ISBN 978-1-4799-1961-1, s. 1–8.
URL http://www.fit.vutbr.cz/research/view_pub.php.en.iso-8859-2?id=11352

[7] Avidan, S.; Shamir, A.: Seam Carving for Content-aware Image Resizing. *ACM Trans. Graph.*, ročník 26, č. 3, Červenec 2007, ISSN 0730-0301, doi:10.1145/1276377.1276390.
URL http://doi.acm.org/10.1145/1276377.1276390

[8] Baatz, G.; Saurer, O.; Köser, K.; aj.: *Large Scale Visual Geo-Localization of Images in Mountainous Terrain*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-33709-3, s. 517–530, doi:10.1007/978-3-642-33709-3_37.
URL https://doi.org/10.1007/978-3-642-33709-3_37

[9] Baatz, G.; Saurer, O.; Köser, K.; aj.: Dataset CH1 [online]. [cit. 2018-01-18].
URL https://cvg.ethz.ch/research/mountain-localization/CH1.tgz

[10] Baboud, L.; Čadík, M.; Eisemann, E.; aj.: Automatic photo-to-terrain alignment for the annotation of mountain pictures. In *CVPR 2011*, June 2011, ISSN 1063-6919, s. 41–48, doi:10.1109/CVPR.2011.5995727.

[11] Badrinarayanan, V.; Kendall, A.; Cipolla, R.: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *CoRR*, ročník abs/1511.00561, 2015, 1511.00561.
URL http://arxiv.org/abs/1511.00561

[12] Brejcha, J.; Čadík, M.: GeoPose3K: Mountain Landscape Dataset for Camera Pose Estimation in Outdoor Environments [online]. 2017 [cit. 2018-01-18].
URL http://merlin.fit.vutbr.cz/elevation/geoPose3K_final_publish.tar.gz

[13] Čadík, M.: LOCATE - vizuální lokalizace v přírodě [online]. http://cadik.posvete.cz/locate/locate_cz.pdf, 2011 [cit. 2018-03-03].

[14] Canny, J.: A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník PAMI-8, č. 6, Listopad 1986: s. 679–698, ISSN 0162-8828, doi:10.1109/TPAMI.1986.4767851.

[15] Chang, C.-C.; Lin, C.-J.: LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, ročník 2, 2011: s. 27:1–27:27, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[16] Chen, D. M.; Tsai, S. S.; Chandrasekhar, V.; aj.: Inverted Index Compression for Scalable Image Matching. In *2010 Data Compression Conference*, Březen 2010, ISSN 1068-0314, s. 525–525, doi:10.1109/DCC.2010.53.

[17] Dachary, L.; The Ht://Dig group: Mifluz - C++ library to use and manage inverted indexes [online]. https://www.gsp.com/cgi-bin/man.cgi?section=3&topic=mifluz, 2016 [cit. 2018-04-09].

[18] Djordjevic, B.; Timcenko, V.: Ext4 File System Performance Analysis in Linux Environment. AIASABEBI'11, World Scientific and Engineering Academy and Society (WSEAS), 2011, ISBN 978-1-61804-028-2, s. 288–293.
URL http://dl.acm.org/citation.cfm?id=2042791.2042846

[19] Eeles, P.; Cripps, P.: *Architektura softwaru*. Computer Press, 2011, ISBN 978-80-251-3036-0.

[20] Fan, R.-E.; Chang, K.-W.; Hsieh, C.-J.; aj.: LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, ročník 9, 2008: s. 1871–1874.

[21] Gonzalez, R.; Woods, R.: *Digital Image Processing*. Pearson/Prentice Hall, 2008, ISBN 9780131687288, 738–762 s.
URL https://books.google.cz/books?id=8uGOnjRGEzoC

[22] Grudl, D.; Nette Foundation: Nette [online]. https://nette.org/cs/, 2016 [cit. 2018-05-09].

[23] Henderson, C.: *Building Scalable Web Sites: Building, Scaling, and Optimizing the Next Generation of Web Applications*. O'Reilly Media, 2006, ISBN 9780596555245.
URL https://books.google.cz/books?id=wIWU94zKEtYC

[24] Hung, Y.-L.; Su, C.-W.; Chang, Y.-H.; aj.: Skyline localization for mountain images. In *2013 IEEE International Conference on Multimedia and Expo (ICME)*, Červenec 2013, ISSN 1945-7871, s. 1–6, doi:10.1109/ICME.2013.6607424.

[25] Hunt, J.; McManus, A.: *Key Java: Advanced Tips and Techniques*. Practitioner Series, Springer London, 2013, ISBN 9781447106074.
URL https://books.google.cz/books?id=4eblBwAAQBAJ

[26] Ladický, L.; Torr, P. H.: The Automatic Labelling Environment [online]. 2018 [cit. 2018-01-17].
URL http://www.robots.ox.ac.uk/~phst/ale.htm

[27] Li, Y.; Snavely, N.; Huttenlocher, D. P.; aj.: *Worldwide Pose Estimation Using 3D Point Clouds*. Cham: Springer International Publishing, 2016, ISBN 978-3-319-25781-5, s. 147–163, doi:10.1007/978-3-319-25781-5_8.
URL https://doi.org/10.1007/978-3-319-25781-5_8

[28] Long, J.; Shelhamer, E.; Darrell, T.: Fully Convolutional Networks for Semantic Segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Červen 2015.

[29] Lowe, D. G.: Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, ročník 60, č. 2, Listopad 2004: s. 91–110, ISSN 1573-1405, doi:10.1023/B:VISI.0000029664.99615.94.
URL https://doi.org/10.1023/B:VISI.0000029664.99615.94

[30] Malan, R.; Bredemeyer, D.: Defining Non-Functional Requirements. Prosinec 2001.

[31] Massung, S.; Geigle, C.; Zhai, C.: MeTA: A Unified Toolkit for Text Retrieval and Analysis. In *Proceedings of ACL-2016 System Demonstrations*, Berlin, Germany: Association for Computational Linguistics, Srpen 2016, s. 91–96.
URL http://anthology.aclweb.org/P16-4016

[32] Otto, M.; Thornton, J.; Bootstrap contributors: Bootstrap [online].
https://getbootstrap.com/, 2018 [cit. 2018-05-09].

[33] Pelikán, J.: *Sémantická segmentace v horském prostředí*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=20179

[34] Pospíšil, M.: *Webová aplikace pro vyhledávání panoramat*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2016.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=18809

[35] Puntambekar, A.: *Analysis of Algorithm and Design*. Technical Publications, 2009, ISBN 9788184316223.
URL https://books.google.cz/books?id=xWaa1YURXtgC

[36] Richards, M.: *Software Architecture Patterns*. O'Reilly Media, Inc., 2015, ISBN 9781491925409.

[37] Rossi, G.; Pastor, O.; Schwabe, D.; aj.: *Web Engineering: Modelling and Implementing Web Applications*. Human–Computer Interaction Series, Springer London, 2007, ISBN 9781846289231.
URL https://books.google.cz/books?id=-l1jW9O2bCEC

[38] Rumbaugh, J.; Booch, G.; Jacobson, I.: *The Unified Modeling Language Reference Manual*. Addison-Wesley object technology series, Addison-Wesley, 2010, ISBN 9780321718952.
URL https://books.google.cz/books?id=T7c3RwAACAAJ

[39] Saurer, O.; Baatz, G.; Köser, K.; aj.: Image Based Geo-localization in the Alps. *Int. J. Comput. Vision*, ročník 116, č. 3, Únor 2016: s. 213–225, ISSN 0920-5691, doi:10.1007/s11263-015-0830-0.
URL http://dx.doi.org/10.1007/s11263-015-0830-0

[40] Soldati, F.: PeakFinder [online]. https://www.peakfinder.org/, 2010 [cit. 2018-05-21].

[41] Spurlock, J.: *Bootstrap: Responsive Web Development*. O'Reilly Media, 2013, ISBN 9781449344603.
URL https://books.google.cz/books?id=LZm7Cxgi3aQC

[42] Steinwart, I.; Christmann, A.: *Support Vector Machines*. Information Science and Statistics, Springer New York, 2008, ISBN 9780387772424.
URL https://books.google.cz/books?id=HUnqnrpYt4IC

[43] The Cesium Consortium: CesiumJS [online]. https://cesiumjs.org/, 2018 [cit. 2018-05-21].

[44] The HDF Group: Hierarchical Data Format, version 5. 1997-NNNN, http://www.hdfgroup.org/HDF5/.

[45] UCSC Genome Informatics Group, University of California: UCSC Genome Browser Wiki: File system performance [online].
http://genomewiki.ucsc.edu/index.php/File_system_performance, 2017 [cit. 2018-05-01].

[46] Unger, R.; Chandler, C.: *A Project Guide to UX Design: For user experience designers in the field or in the making*. Voices That Matter, Pearson Education, 2012, ISBN 9780132931724.
URL https://books.google.cz/books?id=dF7li-90OYQC

[47] Zamir, A. R.; Ardeshir, S.; Shah, M.: GPS-Tag Refinement Using Random Walks with an Adaptive Damping Factor. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Červen 2014, ISSN 1063-6919, s. 4280–4287, doi:10.1109/CVPR.2014.545.

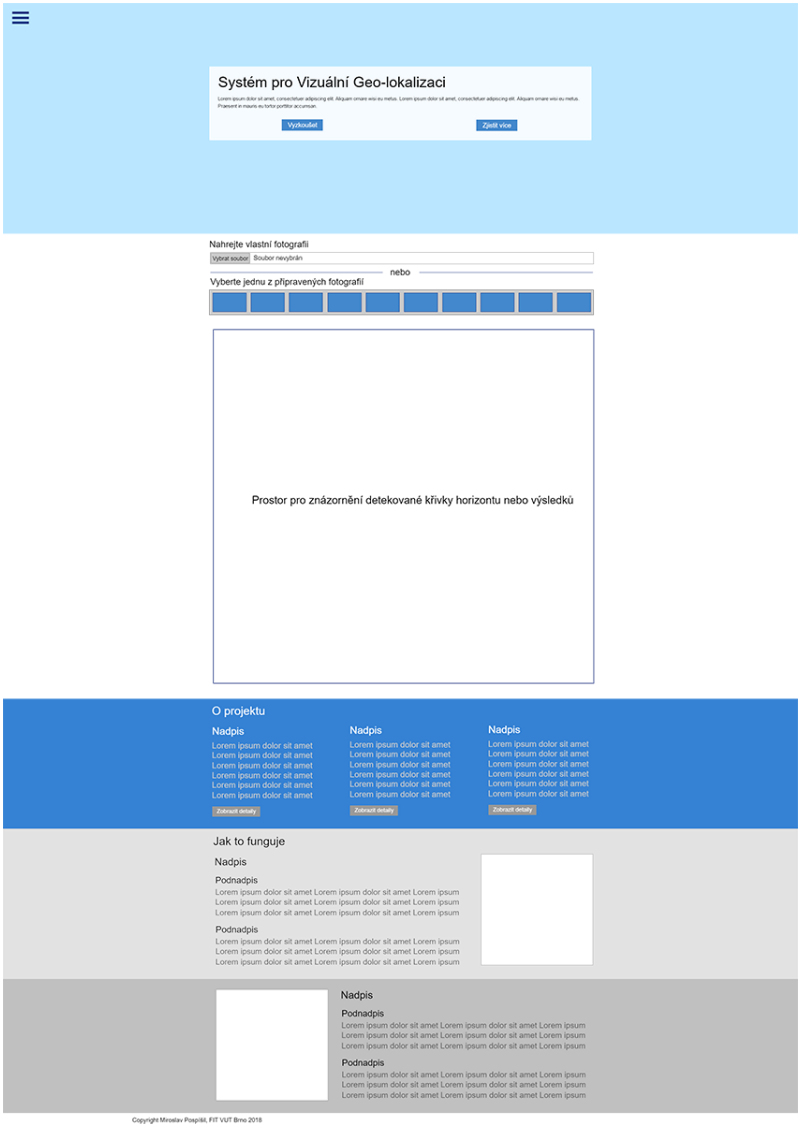
Příloha A

Obsah DVD

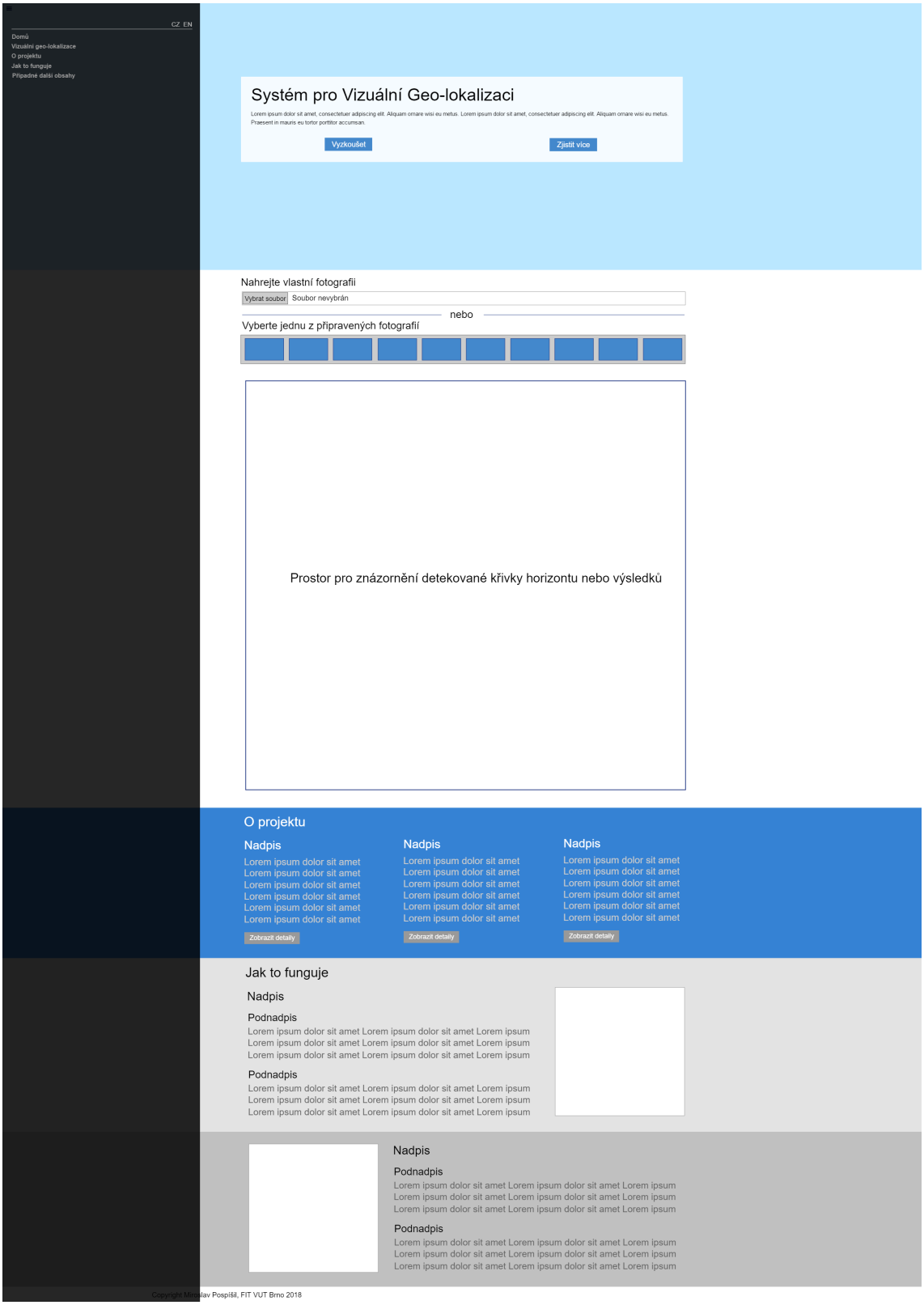
- Diagramy
- Dokumentace.pdf
- Mockup webové aplikace
- Návod k instalaci.txt
- Obrazové materiály
- Plakát.png
- Profilace - měření (detekce křivky horizontu)
- Ukázky webové aplikace
- Zdrojové kódy - Detekce křivky horizontu
- Zdrojové kódy - Dokumentace
- Zdrojové kódy - Inverzní index
- Zdrojové kódy - Webová aplikace

Příloha B

Návrh kostry aplikace



Obrázek B.1: Návrh kostry aplikace



Obrázek B.2: Návrh kostry aplikace - vysunuté menu

Příloha C

Výsledky experimentů

Algoritmus	Střední přesnost	Průměrná absolutní vzdálenost pixelů
Originální impl.	95,0563 %	14,6260 px
Impl. s ignorováním (hranice 10)	95,0472 %	13,9748 px
Impl. s ignorováním (hranice 20)	94,8409 %	13,5312 px
Impl. s ignorováním (hranice 30)	94,2755 %	13,7678 px
Impl. s ignorováním (hranice 40)	84,8694 %	26,6737 px
Impl. s ignorováním (hranice 50)	85,6568 %	37,1541 px
Impl. bez SVM	84,8693 %	26,6737 px
Impl. s LIBLINEAR	85,6568 %	37,1541 px

Tabulka C.1: Výsledky měření všech experimentů

Algoritmus	Střední přesnost	Průměrná absolutní vzdálenost pixelů
Implementovaný algoritmus [24]	96,37 %	9,973 px
FCN8s-Pascal [28]	95,51 %	32,161 px
FCN16s-Pascal [28]	95,39 %	32,888 px
FCN32s-Pascal [28]	95,20 %	33,534 px
FCN8s-SiftFlow-g [28]	94,91 %	34,975 px
FCN8s-SiftFlow-s [28]	95,63 %	31,399 px
Hozion-ALE-CH1 [26]	94,11 %	43,959 px
Horizon-DCSI-CH1 [4]	87,37 %	99,742 px
SegNet-CH1 [11]	114,893 %	90,385 px
FCN8s-SiftFlow-s-CH1 [28]	94,86 %	37,947 px
FCN8s-Pascal-CH1 [28]	94,32 %	41,596 px

Tabulka C.2: Srovnání implementovaného algoritmu s algoritmy z [6]. První řádek: naměřené hodnoty implementovaného algoritmu. Zbytek tabulky: Data z tabulky 3 z [6].

Příloha D

Profilace - výsledky měření

Všechny hodnoty jsou uvedeny v milisekundách.

Metoda	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
Skyline_localization.main	5969	6145	5884	4452	4316	4535	4943	4916	5091	5918	5144,9	5017
Skyline_localization.runLocalization	5818	5950	5739	4326	4157	4403	4843	4788	4956	5085	5006,5	4899,5
Localization.localizate	5800	5950	5738	4323	4155	4395	4838	4784	4956	5085	5002,4	4897
libsvm.svm.svm_predict	3372	3538	3669	2540	2547	2514	3126	3034	2787	3383	3051	3080
FeatureVector.extractSkyline	1204	834	804	509	675	757	697	789	910	527	770,6	773
libsvm.svm.svm_load_model	798	1084	730	546	472	722	628	523	787	679	696,9	700,5
Localization.linkSkylineEdgeSegments	397	431	496	699	422	348	349	378	415	439	437,4	418,5
svm.svm_scale.run	11,2	22,7	7,39	4,54	13,8	6,49	14,8	9,59	10,6	25,9	12,701	10,9

Metoda	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
Skyline_localization.main	2725	2688	2146	2039	2665	2194	2112	2413	2855	2361	2419,8	2387
Skyline_localization.runLocalization	2606	2564	2027	1899	2564	2048	2022	2312	2743	2242	2302,7	2277
Localization.localizate	2589	2562	2014	1898	2550	2040	2014	2306	2725	2242	2294	2274
Train.exportFeatureVector	307	381	349	290	242	196	241	408	283	253	295	286,5
FeatureVector.extractSkyline	875	1096	634	597	846	850	843	731	877	1159	850,8	848
Runtime.exec	16,4	4,33	3,84	2,29	10,3	16,6	1,83	0,429	1,53	1,98	5,9529	3,065
UNIXProcess.waitFor	1353	1064	1012	993	1433	972	906	1147	1533	820	1123,3	1038
Localization.linkSkylineEdgeSegments												

Příloha E

Výsledky měření času (generovaná data)

Procentuální hodnoty vyjadřují procentuální velikost inverzního indexu v paměti. Všechny hodnoty jsou uvedeny v sekundách.

Původní systém	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
100 %	0,03059	0,030339	0,028246	0,029531	0,034183	0,031285	0,034244	0,028134	0,028718	0,02685	0,030212	0,029935

LRU	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	1,83314	1,59509	1,50036	1,36361	1,20412	1,14749	1,08209	1,08777	1,06214	1,20467	1,308048	1,204395
2 %	1,26529	1,36312	1,0572	1,50917	1,1888	1,71862	1,53423	1,02477	1,09488	1,6734	1,342948	1,314205
5 %	1,94651	1,04605	1,11763	1,15065	1,65122	1,17328	1,26411	1,58594	1,77692	1,31463	1,402694	1,28937
10 %	1,54168	1,42137	1,34328	1,37614	1,36345	1,29435	1,95492	1,48289	1,8534	1,13552	1,4767	1,398755
15 %	2,18464	1,70335	1,2561	1,29206	1,13273	1,76875	1,40498	1,78698	1,58051	1,72577	1,583587	1,64193
20 %	1,38161	1,9499	1,48007	1,43546	1,78513	2,38401	1,4695	1,38477	2,24479	2,03892	1,755416	1,6326

RANDOM	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	1,44658	1,26548	1,17622	1,06471	0,99398	0,92048	0,92105	0,87874	0,88095	1,19712	1,074531	1,029345
2 %	2,1854	2,1455	1,64121	1,3489	1,91103	2,351	1,35748	1,9801	0,98401	1,3434	1,724804	1,77612
5 %	0,8996	1,1128	1,3506	1,2048	1,47219	1,89934	1,08404	1,84041	1,354	1,69778	1,391556	1,3523
10 %	1,40096	1,3132	1,18381	1,11302	1,11063	1,18569	1,11315	1,10631	1,09434	1,40052	1,202163	1,14848
15 %	1,89746	1,72382	1,00425	0,9758	1,97101	2,31613	1,9463	1,6463	1,30045	1,348	1,612952	1,68506
20 %	1,73735	1,48073	1,3581	1,2589	1,15584	1,3489	1,84888	1,4201	0,9189	1,0034	1,35311	1,3535

FIFO	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	1,85777	1,50588	1,36897	1,23446	1,7433	2,3414	1,65716	1,90711	1,64694	1,89739	1,716038	1,70023
2 %	1,899	2,10104	1,86413	1,01712	1,31717	2,0007	2,3984	1,1848	1,5417	1,34187	1,666593	1,702915
5 %	2,3546	1,68456	1,03415	1,84304	1,6999	1,3489	1,34401	2,1035	1,45194	0,98187	1,584647	1,56825
10 %	1,62364	1,15908	1,15624	1,56913	1,61401	1,52252	1,47573	1,57405	1,43194	1,337	1,446334	1,499125
15 %	1,4807	1,73431	1,2004	1,3749	1,26043	1,06869	1,30189	1,48498	1,03466	1,34165	1,328261	1,32177
20 %	1,47848	1,38807	1,01406	1,10793	0,87789	1,61994	0,87436	1,26275	0,984	1,46997	1,207745	1,18534

Příloha F

Výsledky měření počtu přepisů (generovaná data)

Procentuální hodnoty vyjadřují procentuální velikost inverzního indexu v paměti.
Hodnoty udávají počet přepisů v paměti z 10.000 pokusů.

LRU	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	9896	9888	9895	9906	9902	9911	9912	9931	9909	9907	9905,7	9906,5
2 %	9846	9945	9852	9999	9815	9856	9981	9986	9926	9848	9905,4	9891
5 %	9743	9682	9720	9645	9768	9800	9778	9744	9798	9729	9740,7	9743,5
10 %	9611	9596	9579	9635	9550	9648	9636	9613	9614	9534	9601,6	9612
15 %	9591	9560	9524	9573	9496	9495	9561	9523	9538	9517	9537,8	9531
20 %	9491	9472	9417	9455	9499	9414	9430	9471	9485	9473	9460,7	9471,5

RANDOM	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	9919	9904	9912	9905	9920	9905	9893	9922	9917	9896	9909,3	9908,5
2 %	9542	9821	9505	9908	9916	9423	9790	9572	9822	9557	9685,6	9681
5 %	9754	9411	9566	9776	9562	9993	9625	9412	9912	9962	9697,3	9689,5
10 %	9523	9537	9528	9496	9549	9507	9518	9545	9521	9492	9521,6	9522
15 %	9910	9599	9643	9722	9442	9481	9498	9700	9930	9539	9646,4	9621
20 %	9901	9918	9919	9906	9892	9821	9725	9475	9787	9702	9804,6	9856,5

FIFO	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	9898	9900	9920	9911	9913	9890	9899	9920	9908	9947	9910,6	9909,5
2 %	9850	9939	9961	9918	9903	9893	9894	9891	9882	9892	9902,3	9893,5
5 %	9842	9816	9778	9746	9846	9746	9779	9776	9761	9799	9788,9	9778,5
10 %	9745	9639	9730	9533	9535	9723	9541	9502	9535	9775	9625,8	9590
15 %	9550	9500	9418	9505	9530	9502	9531	9558	9497	9528	9511,9	9516,5
20 %	9465	9471	9466	9488	9430	9501	9491	9505	9470	9476	9476,3	9473,5

Příloha G

Výsledky měření času (reálná data)

Procentuální hodnoty vyjadřují procentuální velikost inverzního indexu v paměti. Všechny hodnoty jsou uvedeny v sekundách.

Původní systém	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
100 %	0,0156663	0,015905	0,016289	0,017563	0,018039	0,018189	0,015504	0,018672	0,016353	0,01571	0,0167928	0,016321

LRU	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	36,4044	38,6864	38,6313	38,3064	38,4531	38,2954	38,2827	40,5265	36,578	36,8653	38,10295	38,3009
2 %	99,7219	104,51	107,981	110,335	108,492	105,085	103,695	104,267	103,961	105,086	105,31339	104,7675
5 %	253,589	282,312	278,433	273,012	279,989	277,431	283,426	274,341	269,319	272,97	274,4822	275,886
10 %	509,627	536,58	556,69	551,619	513,811	540,041	546,522	571,178	552,067	562,573	544,0708	549,0705
15 %	752,775	840,808	790,566	853,408	767,213	820,586	915,013	925,048	802,44	758,06	822,5917	811,513
20 %	983,335	995,156	1016,99	1097,68	1172,12	1125,25	1146,68	1157,97	1180,5	1140,02	1101,5701	1132,635

RANDOM	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	1,89755	1,70857	1,85908	1,78845	1,71888	1,29388	3,1668	1,6245	1,40391	1,6822	1,814382	1,71373
2 %	1,67138	2,20862	1,20078	1,89891	1,81646	2,49742	1,48724	2,04423	2,68822	4,78977	2,230303	1,97157
5 %	1,43294	2,296	1,16788	1,36482	1,38995	1,2608	1,28294	1,38983	1,33668	1,16913	1,409097	1,35075
10 %	3,05908	2,09577	1,61435	1,30798	1,39913	1,62545	1,93877	1,93975	1,25511	1,86478	1,810017	1,74512
15 %	1,94289	2,59606	1,72566	1,65066	1,83951	1,78542	1,48008	1,48721	1,39929	1,72727	1,763405	1,72647
20 %	1,75345	1,71991	2,59	1,83583	1,84927	1,38962	1,81169	1,75132	2,12363	2,31093	1,913565	1,82376

FIFO	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	2,09558	2,73288	2,42236	2,32571	1,45445	1,52394	1,65114	3,15949	1,59092	1,46958	2,042605	1,87336
2 %	2,03717	1,44777	2,38424	1,6105	1,52999	1,79662	1,86851	2,06306	2,52958	2,09759	1,936503	1,95284
5 %	1,45508	2,21458	2,61929	1,60617	2,33061	2,03508	1,38879	1,43021	1,96649	1,67118	1,871748	1,81884
10 %	1,38136	1,39035	1,48999	2,00572	1,5579	1,41429	2,03206	2,41774	1,87925	1,52782	1,709648	1,54286
15 %	1,38395	2,2958	1,33563	2,16784	1,71208	1,51484	1,24284	1,52886	1,45967	1,92055	1,656206	1,52185
20 %	1,50022	1,59669	1,29451	2,31756	1,31185	1,56752	1,5489	1,92955	1,24811	1,36699	1,56819	1,52456

Příloha H

Výsledky měření počtu přepisů (reálná data)

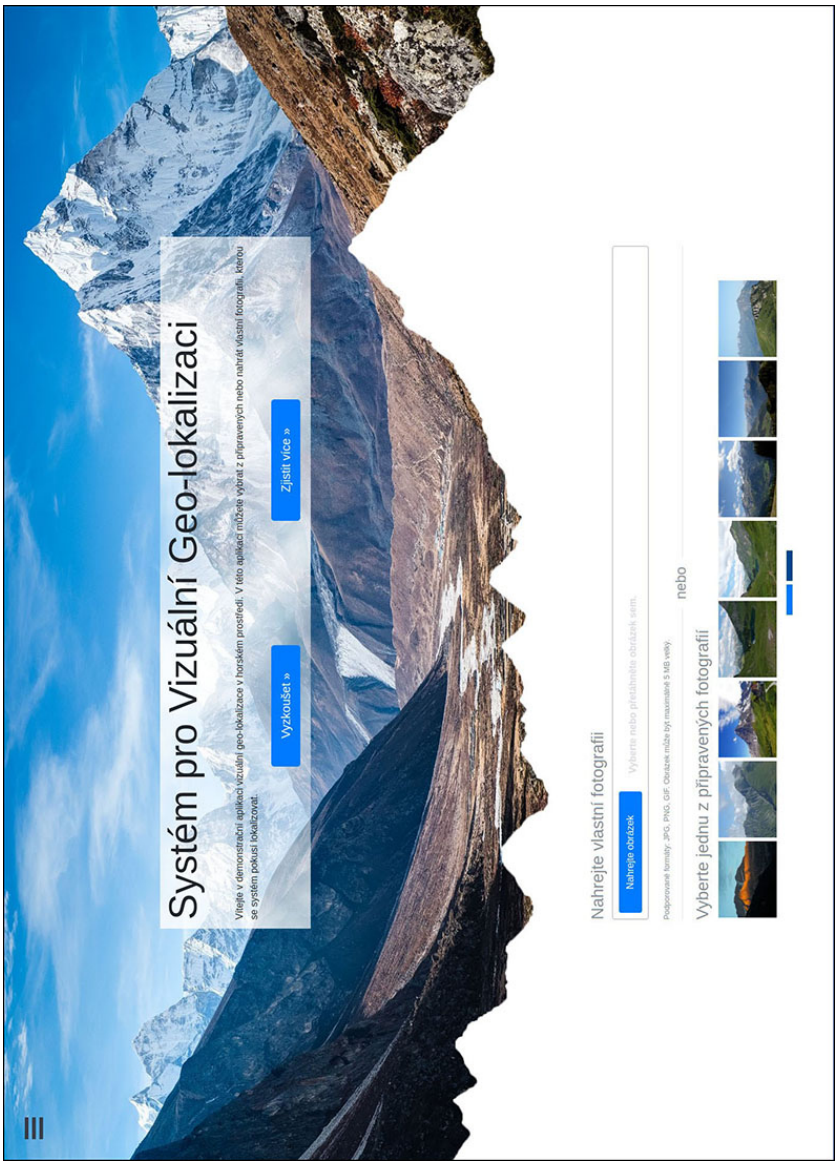
Procentuální hodnoty vyjadřují procentuální velikost inverzního indexu v paměti.
Hodnoty udávají počet přepisů v paměti ze 100.000 pokusů.
Tabulka s měřením LRU je vynechána, neboť všechna měření vracela stejnou hodnotu - 100.000.

RANDOM	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	99760	99759	99781	99773	99769	99761	99742	99741	99746	99748	99758	99759,5
2 %	99725	99749	99733	99726	99720	99742	99719	99713	99736	99730	99729,3	99728
5 %	99745	99709	99708	99702	99696	99721	99701	99715	99706	99738	99714,1	99708,5
10 %	99718	99702	99710	99719	99712	99713	99703	99720	99711	99721	99721,9	99712,5
15 %	99739	99732	99723	99719	99699	99695	99658	99738	99692	99707	99710,2	99713
20 %	99702	99713	99699	99712	99717	99749	99694	99713	99658	99695	99705,2	99707

FIFO	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr	Medián
1 %	99733	99716	99734	99725	99707	99708	99699	99751	99716	99723	99721,2	99719,5
2 %	99711	99728	99725	99709	99700	99742	99714	99739	99705	99718	99719,1	99716
5 %	99704	99744	99728	99703	99703	99689	99702	99740	99706	99710	99712,9	99705
10 %	99731	99684	99707	99677	99688	99722	99721	99706	99697	99714	99704,7	99706,5
15 %	99713	99698	99697	99698	99714	99688	99701	99706	99699	99711	99702,5	99700
20 %	99695	99699	99721	99683	99695	99710	99677	99693	99721	99695	99698,9	99695

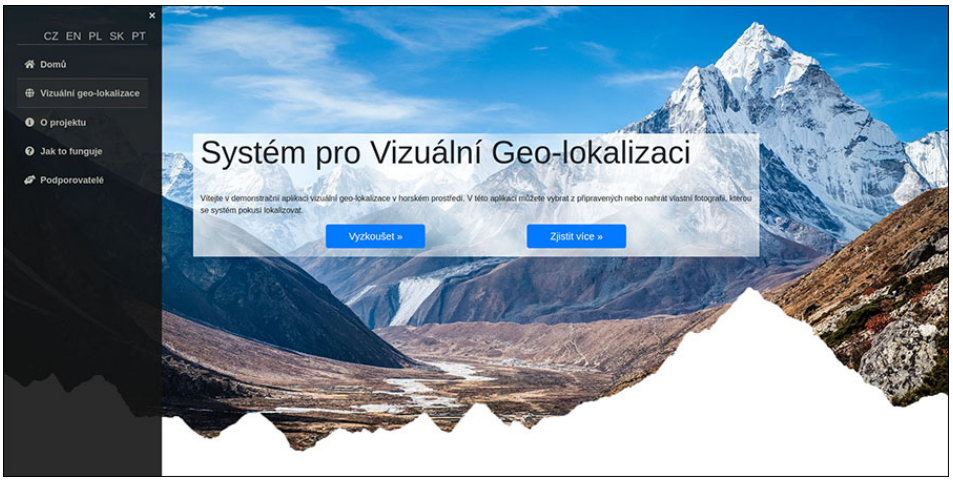
Příloha I

Ukázky aplikace

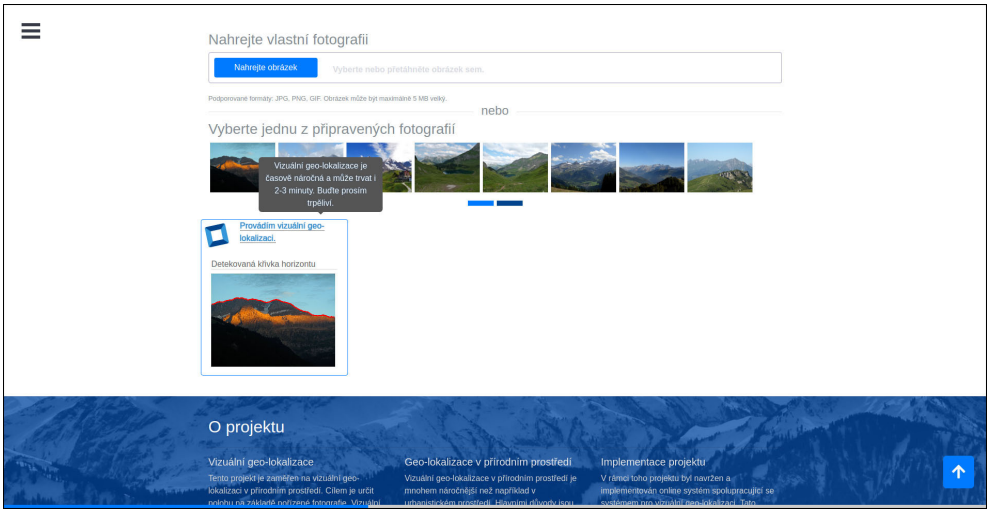


Obrázek I.1: Ukázka vrchní části aplikace.

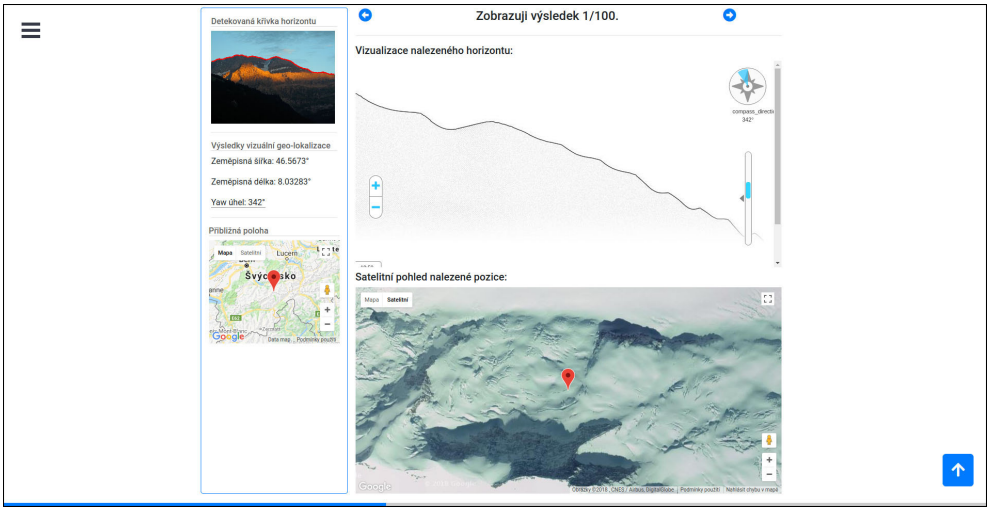
Obrázek I.2: Ukázka spodní části aplikace.



Obrázek I.3: Ukázka vysunutého menu.



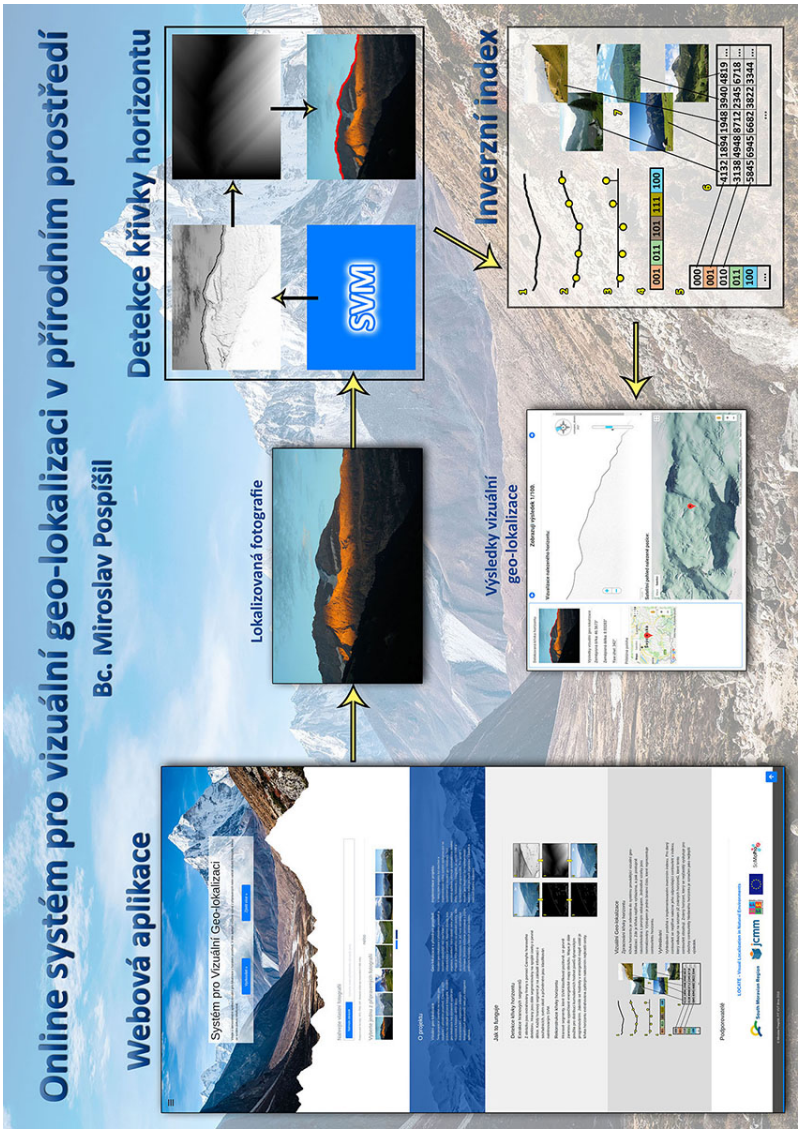
Obrázek I.4: Ukázka práce vizuální geo-lokalizace.



Obrázek I.5: Ukázka výsledků vizuální geolokalizace.

Příloha J

Plakát



Obrázek J.1: Ukázka plakátu.