VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

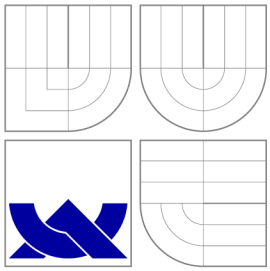
PANORAMA Z PTAČÍ PERSPEKTIVY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

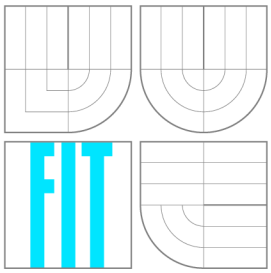
AUTOR PRÁCE
AUTHOR

LUKÁŠ SOBOTKA

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PANORAMA Z PTAČÍ PERSPEKTIVY

BIRDS EYE PANORAMA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ SOBOTKA
AUTHOR

VEDOUCÍ PRÁCE — MARTIN KOLÁŘ, M.Sc.
SUPERVISOR

BRNO 2015

Abstrakt

Cílem této práce je vytvoření panoramatu z fotografií pořízených z náhodně pohybujícího se zařízení s operačním systémem Android. Toto zařízení je umístěné na létajícím aparátu (drak, heliový balón, dron, ...). Vzniklé fotografie snímají zemi z výšky, z ptačí perspektivy. Byla tedy vytvořená a otestovaná aplikace pro OS Android, která vytváří kolekci snímků a následně provádí algoritmus pro vytvoření panoramatu. Tento algoritmus určuje pomocí metody SURF deskriptory, které jsou filtrovány algoritmem RANSAC pro nalezení optimální homografie. Provádí se iterativně nad celou kolekcí pořízených fotografií. Obrazová data jsou zpracována pomocí knihovny OpenCV.

Abstract

The goal of this paper is to create panorama from the collection of photographies, which are taken from randomly moving device with OS Android. The device is placed on flying vehicle (a kite, a helium balloon, a drone etc.). The photographies are taken from birds eye view. The Android application was created and tested. This application creates a collection of photographies and then runs an algorithm to create panorama. This algorithm specifies descriptors using SURF, which are filtered by RANSAC algorithm for finding the optimal homography.This algorithm is perfomed iteratively over the collection. The image data is processed using the OpenCV library.

Klíčová slova

Panorama, spojování obrazů, klíčové body, SURF, RANSAC, homografie, ortorektifikace, Android, OpenCV

Keywords

Panorama, image stitching, keypoints, SURF, RANSAC, homography, orthorectification, Android, OpenCV

Citace

Lukáš Sobotka: Panorama z Ptačí Perspektivy, bakalářská práce, Brno, FIT VUT v Brně, 2015

Panorama z Ptačí Perspektivy

Prohlášení

Prohlašuji, že jsem svoji bakalářskou práci na téma Panorama z Ptačí Perspektivy vypracoval samostatně pod vedením pana Martina Koláře, M.Sc. a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce.

.....
Lukáš Sobotka
18. května 2015

Poděkování

Rád bych poděkoval svému vedoucímu bakalářské práce Martinu Kolářovi, M.Sc. za vedení, cenné rady a odbornou pomoc, kterou mi poskytl v průběhu tvorby této práce.

© Lukáš Sobotka, 2015.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

Panoramatické fotografie jsou známé již od první poloviny 19. století jako fotografie s neobvykle velkým úhlem záběru. Tato metoda je používána, pokud se nevejde fotografovaný objekt na jeden snímek z důvodu úzkého zorného pole. Jako další důvod k vytváření panoramatických fotografií může sloužit vnímání lidského oka. Oko má zorné pole v horizontálním směru asi 140°, ale fotoaparáty s klasickými objektivy o ohniskové vzdálenosti 28mm zachycují snímky v zorném úhlu 74°. Spojování více fotografií do jedné tedy zvětšuje na výsledném snímku zorné pole v horizontálním směru a tím se přibližuje vlastnostem lidského oka.

Tato práce se u panoramatické fotografie zabývá rozšířením zorného pole i vertikálně, dochází tedy ke spojování fotografií nejen v horizontálním, ale i vertikálním směru. Jak už název práce, Panorama z ptačí perspektivy, napovídá, nejedná se o klasické panorama krajiny, ale předpokládá se pořizování fotografií z výšky. To je možné uskutečnit vypuštěním héliového balónu, draka, či jiného aparátu, který je schopen vynést do dostatečné výšky zařízení s operačním systémem Android.

Výstupním programem této bakalářské práce je aplikace pro operační systém Android. Chování této aplikace lze rozdělit na dvě fáze. Za prvé je nutné pořídit kolekci fotografií, s kterou se následně bude pracovat. Za druhé je spuštěn algoritmus pro spojování pořízených fotografií. Výsledkem aplikace by měl být jeden obraz zobrazující mapu okolí, která bude podobná satelitním snímkům například ze serveru `maps.google.com` či `mapy.cz`.

První kapitola této práce se zaobírá potřebnými informacemi k pochopení problému spojování fotografií – detekování klíčových bodů, určování deskriptorů, vyhledávání spolu souvisejících deskriptorů a vytvoření matice homografie.

Další kapitola seznamuje s vytvářením aplikace pro operační systém Android. Je zde popsán samotný operační systém, jeho architektura, jeho filozofie a nutné znalosti pro vývoj aplikace.

Třetí kapitola se věnuje implementaci aplikace. Seznamuje s problémy vzniklými při vývoji aplikace a způsobem jejich řešení.

Následující kapitola vyhodnocuje vzniklé snímky vytvořené aplikací zařízením s operačním systémem Android.

Poslední, závěrečná kapitola shrnuje činnost prováděnou v rámci této práce a nabízí další možnosti pro řešení aplikace.

Kapitola 2

Spojování fotografií

Tato kapitola obsahuje potřebné informace k pochopení problematiky spojování dvou fotografií do jednoho výsledného obrazu. Důležité je najít ve dvou fotografiích překrývající se části, aby bylo posléze možné sloučit v panoramatickou fotografii.

Problém spojování je řešený metodou založené na příznacích – ta extrahuje z obrazu kolekci příznaků, které následně porovnává s kolekcí druhého obrazu a hledá podobné dvojice. Tato dvojice se považuje za shodný bod obou obrazů. Na základě nalezených dvojic se vypočítá matice homografie, s kterou už můžeme deformovat obraz a spojit do výsledné panoramatické fotografie.

V této kapitole se nachází základní poznatky detekování obrazu, hledání množiny příznaků, vyhledávání spolu souvisejících dvojic deskriptorů, základy deformace obrazu a popis homografie.

2.1 Hrany

Lidské vnímání je založeno na rozpoznávání hran (edge), nejen proto je hrana důležitým prvkem používaným v oblasti počítačové analýzy obrazu. Jedná se o lokální vlastnost bodu a jeho bezprostředního okolí – hranu v diskrétním obraze vnímáme tam, kde dochází k výrazné změně sousedních pixelů. Hrana je určena gradientem, tj. velikostí a směrem. Směr lze popsat vektorovým operátorem nabla ∇ [34]:

$$\nabla f(x,y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y} \right)$$

a velikost gradientu je určena jako délka vektoru:

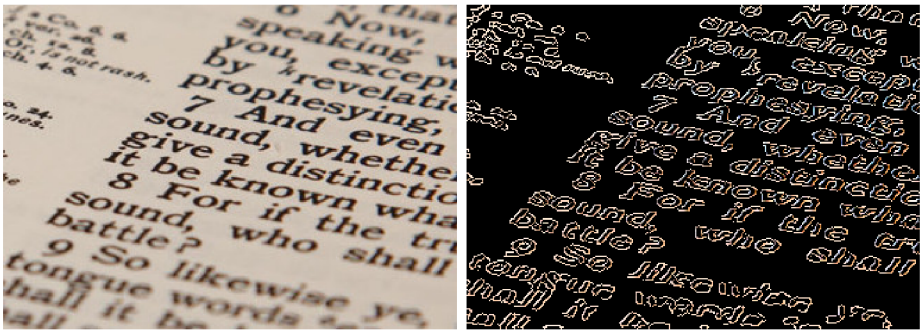
$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f(x,y)}{\partial x} \right)^2 + \left(\frac{\partial f(x,y)}{\partial y} \right)^2}$$

Hrana tedy udává, jak rychle se mění intenzita v malém okolí pixelu. [30].

2.2 Detekce hran

Detektor hran je kolekce důležitých metod zpracování obrazu, které identifikují body na základě diskontinuity v obraze – ostře měnícím se jasu. Obrazová data v reálném světě jsou inherentně diskrétní, neexistuje funkce vyjádřená pro diskrétní definiční obor, proto

není žádný inherentní způsob určení hran v daném diskrétním obraze. V diskrétním obraze gradient pouze odhadujeme.[27]



Obrázek 2.1: Ukázka hranového detektoru Canny, výstup na pravé straně. Vlevo je originální obraz. (vlastní práce)

Pro určení hrany, gradientu pixelu, se používají postupy založené na analýze okolí pixelu s použitím konvolučních, nebo jiných operátorů. Konkrétně se jedná o konvoluci obrazu I s Gaussovým operátorem G. Každému místu, kde se vyskytuje změna intenzity, odpovídá vrchol první derivace a současně také nulový bod druhé derivace. Úloha detekce těchto změn je řešena nalezením průsečíků s osou u druhé derivace ∇^2 intenzity. To znamená, že se snažíme najít nulové body pro

$$f(x, y) = \nabla^2 \left[G(x, y) * I(x, y) \right],$$

kde $I(x, y)$ je obraz, $*$ je operace konvoluce a ∇^2 je Laplaceův operátor. [28]

2.3 Detektor klíčových bodů

Klíčové body (keypointy) označují specifické body v obrazu, kterým jsou něčím speciální – hrany, rohy, osamocené body apod. Tyto body následně slouží pro získání deskriptorů. V dnešní době již existuje více detektorů, mezi které patří ORB, BRISK, FREAK, FAST, SIFT, SURF a další [4][5]. Mezi nejvýkonnější a nejrychlejší detektory se řadí SURF detektor.

Detektor SURF

Zdroje [32] a [14] podrobně popisují detektor SURF a algoritmus, kterým je prováděn. Jsou zde také definovány Hessovy matice a integrální obraz, který jsou v algoritmu využívány. V této části jsou použity citace z těchto zdrojů.

Algoritmus SURF (Speeded Up Robust Features) navazující na předchozí algoritmus SIFT je vytvořený roku 2006 Herbertem Bayem. Jak název algoritmu napovídá, hlavní jeho výhodou je rychlost. V porovnáním s algoritmem SIFT je mnohem rychlejší i přes zachování jeho vlastností (invariantnost měřítka, osvětlení a šumu).

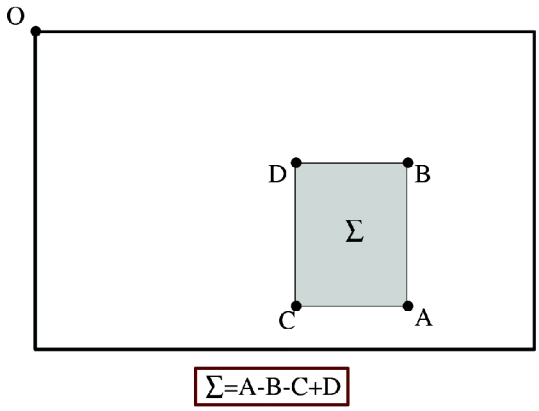
Algoritmus SURF ještě před samotnou detekcí vypočítá pro zpracovávaný obraz jeho integrální obraz, následně může proběhnout detekce klíčových bodů, kterou algoritmus provádí pomocí rychlých a přesných Hessových matic.

Integrální obraz

Bod integrálního obrazu $I_{\sum}(\mathbf{x})$ na souřadnicích $\mathbf{x} = (x, y)$ je definován jako suma všech pixelů vstupního obrazu I tvořící obdélník daný počátkem souřadnic a bodem $\mathbf{x}$. Platí zde následující rovnice:

$$I_{\sum}(\mathbf{x}) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j)$$

Jakmile máme vypočítán integrální obraz $I_{\sum}$, stačí nám čtyři operace k výpočtu součtu intenzit přes jakýkoliv obdélník, jehož strany jsou souběžné s osami. Doba výpočtu je nezávislá na velikosti obdélníkové části. Výpočet součtu intenzity obdélníku je znázorněn na obrázku 2.2.



Obrázek 2.2: Použitím integrálního obrazu lze vypočítat součtem čtyř hodnot plochu obdélníkové oblasti libovolné velikosti. Převzato z [32].

Hessovy matice

Pro vyhledávání významných bodů byla vybrána Hessova matice pro její dobrou přesnost i čas. Hessova matice $\mathcal{H}(x, \sigma)$ v bodě $\mathbf{x}$ a měřítkem σ je definována:

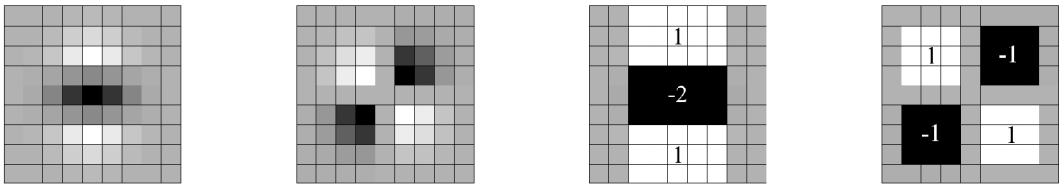
$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{pmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{pmatrix},$$

kde $L_{xx}(x, \sigma)$ je konvoluce Gaussovy druhé derivace $\frac{\partial^2}{\partial x^2} g(\sigma)$ s obrazem I v bodě $\mathbf{x}$ a podobně je to pro $L_{xy}(x, \sigma)$ a $L_{yy}(x, \sigma)$.

Detekce keypointů

Gaussova funkce je optimální pro scale-space analýzu. Pro dostatečně kvalitní výsledek však není nutné použít zcela přesných výpočtů, proto autoři nepoužili složitější Gaussovu funkci, ale nahradili ji tzv. obdélníkovými filtry, kterými aproximují její druhé derivace. Tímto krokem se celý výpočet výrazně zrychlil.

Na obrázku 2.3 je znázorněný obdélníkový filtr o velikosti 9×9, který aproximuje Gaussovu funkci s nejlepším měřítkem $\sigma = 1.2$. Aproximace si označíme D_{xx}, D_{yy} a D_{xy}. Váhy



Obrázek 2.3: Vlevo je zobrazena druhá derivace Gaussovy funkce ve směru y, vedle ní je druhá derivace Gaussovy funkce ve směru xy. Dále jsou zobrazeny aproximace obdélníkovým filtrem pro druhou derivaci Gaussovy funkce ve směru y a druhou derivaci Gaussovy funkce ve směru xy. Šedá oblast je rovna nule. Převzato z [32].

aplikované na obdélníkové oblasti se uchovávají pro větší efektivitu výpočtu, ale je nutné vyvážit relativní váhu ve výrazu pro Hessův determinant konstantou $\omega = \frac{|L_{xy}(1.2)|_F |D_{xx/yy}(9)|_F}{|L_{xx/yy}(1.2)|_F |D_{xy}(9)|_F}$, kde $|x|$ je Frobeniova norma.

Výsledkem nám je rovnice

$$\det \mathcal{H}(\mathsf{approx}) = D_{xx}D_{yy} - (\omega D_{xy})^2.$$

Aproximací determinantu odpovídá oblast na pozici $\mathbf{x}$. Výstup je uložen jako mapa oblastí v různých měřítkách. Následně jsou vyhledávána lokální maxima pomocí kvadratické interpolace.

2.4 Vyhledávání deskriptorů

Stejně jako u detektorů existuje celá řada deskriptorů, mezi než patří ORB, FREAK, SIFT, HOG a další. Také autoři SURF detektoru vytvořili vlastní deskriptor.

SURF deskriptor

Pro výpočet deskriptoru se používá čtvercové okolí s délkou strany 20σ se středem v detekovaném bodě, kde je čtverec natočen podle orientace významného bodu. Tento čtverec je rozdělen na 4×4 podoblastí o velikosti $5\sigma \times 5\sigma$. V každé z nich se vybere 5 pravidelně rozmístěných bodů a pro ně jsou vypočteny hodnoty (d_x, d_y). Pro každou z podoblastí je určen subdeskriptor daný vektorem $\mathbf{v} = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$. Posledním krokem výpočtu deskriptoru je spojení subdeskriptorů všech podoblastí do výsledného 64-dimenzionálního deskriptoru. [14]

2.5 Vyhledávání podobných deskriptorů

Pro porovnání podobnosti dvou fotografií je nutná znalost deskriptorů. Podobnost se vyhodnocuje na základě vzdálenosti mezi dvěma deskriptory různých fotografií. Pro porovnávání vzdáleností je možné použít jeden z algoritmů Brute Force, či FLANN. [29]

2.5.1 Brute Force

Metoda Brute Force porovnává každý deskriptor jednoho obrazu s každým deskriptorem druhého obrazu. Tím je zaručené nalezení dvojic s nejlepší shodou. Tato metoda je ovšem příliš časově náročná, proto se nehodí na rozsáhlé databáze obrázků. [24]

2.5.2 FLANN

FLANN (Fast Library for Approximate Nearest Neighbors)[1] je stromově založený algoritmus, který automaticky vybírá nejlepší metodu na základě datového souboru.[18] Vybírá ze tří metod: náhodný k-dimenzionální strom, hierarchický k-means a Brute Force vyhledávání. [24]

2.6 RANSAC

RANSAC (Random sample consensus) je algoritmus vyvinutý pro počítačové vidění v roce 1981 M. Fischlerem a R. Bollesem [19]. Jedná se o iterativní metodu pro odhad matematického modelu z množiny datových bodů. Algoritmus předpokládá, že ne všechny vstupní body nenáleží hledanému modelu.

Vstupní datové body algoritmus 1 rozděluje do dvou skupin tzv. outliers a inliers. Outliers jsou chybové body, tedy nekorespondující body, které jsou odlehlé od ideálního řešení. Inliers jsou korespondující body nacházející se v blízkém okolí ideálního řešení. Nalezení těchto inliers bodů je výsledkem algoritmu, jehož přesnost ovlivňuje poměr inliers a outliers bodů. Určení těchto bodů je znázorněné na algoritmu [16]

Algoritmus 1: RANSAC[16]

1: Náhodně se vybere minimální počet bodů, které určí parametry modelu.
2: Vyřeší se model pro vybrané parametry.
3: Určí se, kolik bodů z původní množiny všech bodů splňují kritérium předem definované tolerance ϵ.
4: Pokud je podíl inliers bodů a počtu množiny všech bodů překročí předem definovaný práh τ, opětovně se odhadnou parametry modelu s využitím inliers bodů a ukončí se algoritmus.
5: V opačném případě se opakují kroky 1 až 4 (maximálně N–krát).

Dále je tento algoritmus robustní, jelikož je poměrně dobře odolný k velkému množství chyb (outliers) ve vstupních datech, a nedeterministický, protože vytváří výsledek jen s určitou pravděpodobností, která s přibývajícími iteracemi stoupá.[23][19]

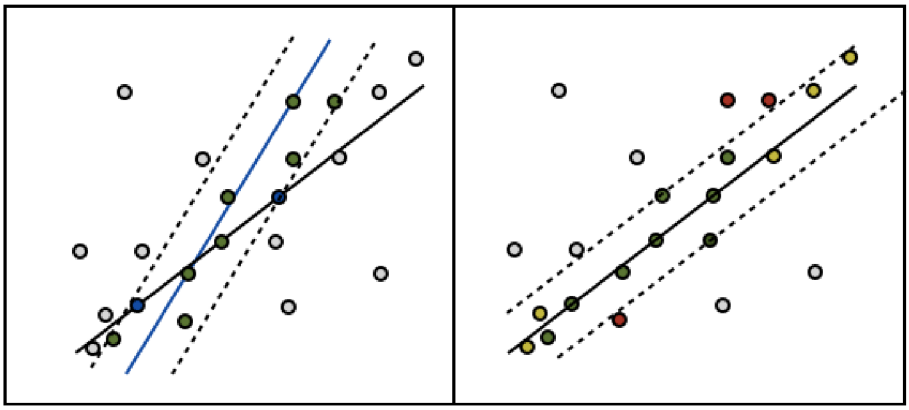
2.7 Homografie

Homografie je geometrická transformace popisující transformaci souřadnice bodu $a = [x, y, 1]^T$ roviny π na odpovídající souřadnice $a' = [x', y', 1]^T$ v rovině π'. Je definována jako perspektivní transformace mezi dvěma projektivními rovinami π a π!.

Homografie je transformační matice o 3 řádcích a 3 sloupcích, která převádí pixel jednoho snímku $a = (x, y, 1)$ na bod v jiném obraze $a' = (x', y', 1)$. Tento převod se provádí na základě vztahu $wa' = H \cdot a$, kde H je matice homografie a w je homogenní souřadnice měřítka.

$$a' = H \cdot a = \begin{bmatrix} wa'_x \\ wa'_y \\ w \end{bmatrix} = \begin{bmatrix} h_00 & h_01 & h_02 \\ h_10 & h_11 & h_12 \\ h_20 & h_21 & h_22 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ 1 \end{bmatrix}$$

[1] v překladu rychlá knihovna pro nalezení přibližných nejbližších sousedů



Obrázek 2.4: Na levé straně vybrány dva body (modré) pro utvoření nové přímky (černá). Předběžné inliers body (na pravé straně) lépe pasují k nové přímce, obsahují i některé nové body (žlutá) a některé další jsou vyloučeny (červená). Převzato z [23].

Při aplikaci tohoto převodu na všechny pixely obrazu vzniká nový obraz, což je zdeformovaná verze původního obrazu. V oblasti počítačového vidění slouží homografie ke spojení v jeden celek dvou obrazů stejné rovinné plochy v prostoru. Pro správný výpočet homografie z dvou různých obrazů jsou potřeba alespoň 4 korespondující body. [22][15]

Algoritmus 2: Homografie [22]

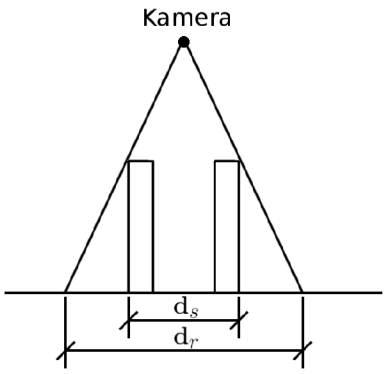
1. **Zájmové body:** určí se zájmové body v každém snímku.
2. **Předpokládané korespondence:** na základě blízkosti a podobnosti v intenzitě sousedních bodů se vyberou dvojice zájmových bodů.
3. **Ransac:** pomocí RANSAC algoritmu (viz algoritmus 1) se vybere homografie H s nejvyšším počtem inliers bodů. V případě shody se vybere řešení, které má nejnižší standardní odchylku inliers.
4. **Optimální odhad:** opakovaně se provede odhad homografie H ze celé množiny korespondujících bodů určených jako inliers body.
5. **Výběr:** další korespondence zájmových bodů jsou teď určeny pomocí odhadované homografie H.

Poslední dva kroky mohou být opakovány do té doby, kdy bude počet korespondujících inliers bodů stabilní.

2.8 Ortorektifikace

Ortorektifikace je proces odstraňování efektu perspektivy a reliéfu krajiny za účelem vytvoření planimetricky správného obrazu (viz obrázek 2.6).[17] Tedy ortofotosnímek, ortorektifikací modifikovaná fotografie, je upravený snímek z centrální do ortogonální projekce. Na ortofotosnímku je eliminován vliv náklonu osy kamery od svislého směru a vliv výškových rozdílů mezi body v předmětovém prostoru.[21] Výsledný ortofotosnímek má konstantní měřítko, kde jsou jednotlivé prvky reprezentovány ve správných pozicích. Důsledkem takového vykreslení je možnost korektního měření úhlů, vzdáleností a ploch [17]. Ukázka

nepřesnosti měření u perspektivního zobrazení je znázorněna na obrázku 2.5.

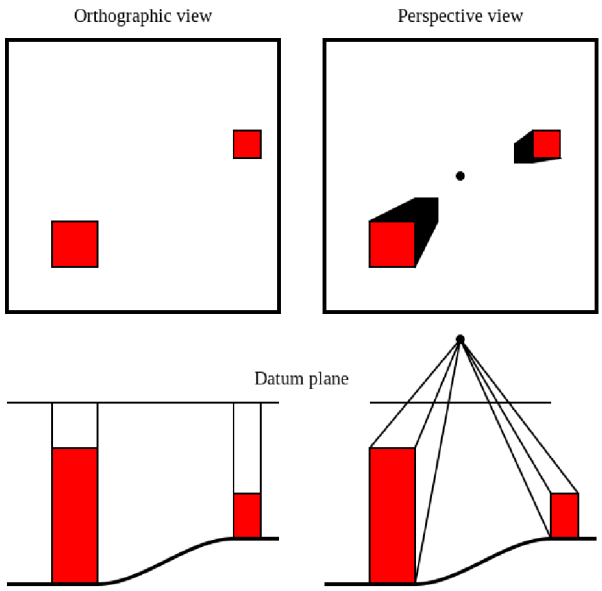


Obrázek 2.5: Rozdíl vzdálenosti budov mezi měřením z perspektivního snímku – d_s a reálnou vzdáleností – d_r. (vlastní práce)

Pro výpočet ortofotosnímku potřebujeme původní digitální snímek včetně znalosti prvků vnitřní a vnější orientace. Dále potřebujeme digitální model terénu (DMT) nebo povrchu (reliéf včetně budov, porostů apod.). Pro pozemní prvek, jehož poloha je dána souřadnicemi X,Y, je nutné určit nejprve výšku Z digitálního modelu terénu. V případě, že je DMT definován jako mříž (čtvercový rastr), je souřadnice Z určena interpolací z nejbližších čtyř bodů DMT. [21]

Ortorektifikované obrazy jsou používané v různých aplikacích jako jsou Google Earth, ArcMap, atd. [17] Dále se také ortofotografie běžně používají při vytváření geografických informačních systémů (GIS).

Kvalita výsledného ortorektifikovaného snímku je závislá zejména na kvalitě vlícovacích bodů a přesnosti digitálního modelu terénu. Mezi metody pro získání dat digitálního modelu terénu patří fotogrammetrické metody, digitalizace vrstevnic existujících map, laserový skener, polární metody, radarová interferometrie a metody používající GPS.



Obrázek 2.6: Rozdíl ortorektifikovaného a perspektivního obrazu. Převzato z [3].

Kapitola 3

Vývoj aplikací na platformě Android

Tato kapitola se zaobírá popisem operačního systému Android. Obsahuje základní informace týkající se Androidu, popisuje jeho filozofii a rozebírá architekturu Androidu, tedy jeho vnitřní strukturu. Další část kapitoly se už věnuje samotnému vývoji aplikace – popisuje základní prvky aplikace a potřebné kroky k jejímu vytvoření.

V následující kapitole je čerpáno ze zdrojů [20] a [33], které pojednávají o stejném tématu, programováním pro OS Android.

3.1 Android

OS Android je rozsáhlý operační systém vytvořený činností společnosti Google. Jedná se o software pod licencí open-source, tedy s otevřeným zdrojovým kódem, který vydává Open Handset Alliance, nezisková skupina společností zabývajících se výrobou, provozem a poskytováním služeb pro telekomunikační zařízení . Cílem této skupiny je vytvořit kompletní, otevřenou a bezplatnou mobilní platformu.

Operační systém je založen na Linuxovém jádře, které zajišťuje zabezpečení systému jako celku, správu paměti, správu procesů, přístup k síti a ovladačům všech vnitřních senzorů a komponent. Jednotlivé aplikace k funkcím jádra nepřistupují přímo, ale prostřednictvím Android API. Mezi nejčastější zařízení používající OS Android patří mobilní telefony a tablety.

Android je neustále vyvíjený systém. Aktualizace systému ale nemusí nutně proběhnout na všech zařízeních, tudíž mezi uživateli figuruje v několika verzích. Rozšíření jednotlivých verzí systému Android je zobrazeno v tabulce 3.1. Podle této tabulky v současné době používá systém Android verze 2.2 0,4% všech uživatelů zařízení s Android systémem. Stojí tedy za uvážení, zda-li má ještě smysl vytvářet aplikaci pro tuto verzi.

S novými verzemi Androidu přichází opravy zjištěných chyb, přidává se nová funkčnost do systému a vznikají nové funkce, které jsou kompatibilní s vyššími verzemi systému.

3.2 Architektura

Architektura operačního systému Android se skládá z pěti vrstev, kde každá z těchto vrstev provádí různé operace a vystupuje víceméně samostatně. V praxi však dochází ke spolupráci

Verze	Kódové označení	API	Podíl
2.2	Froyo	8	0,4%
2.3.3 – 2.3.7	Gingerbread	10	6,4%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	5,7%
4.1.x	Jelly Bean	16	16,5%
4.2.x		17	18,6%
4.3		18	5,6%
4.4	KitKat	19	41,4%
5.0	Lollipop	21	5,0%
5.1		22	0,4%

Údaje shromážděné během 7 dnů, s ukončením 6. dubna 2015. Verze s podílem menším 0,1% nejsou zaznamenané.

Tabulka 3.1: Rozšíření verzí systému Android. Převzato z [11].

jednotlivých částí a vrstvy tímto nejsou mezi sebou striktně odděleny. Schéma architektury operačního systému Android je vyobrazeno na obrázku 3.1.

3.2.1 Linux Kernel

Jak již bylo výše (viz 3.1) zmíněno, Android je postaven na Linuxovém jádře. Linux je nejrozšířenější operační systém a stejně jako Android se vydává pod licencí open-source. Mezi hlavní výhody výběru Linuxového jádra náleží přenositelnost a bezpečnost.

Přenositelnost

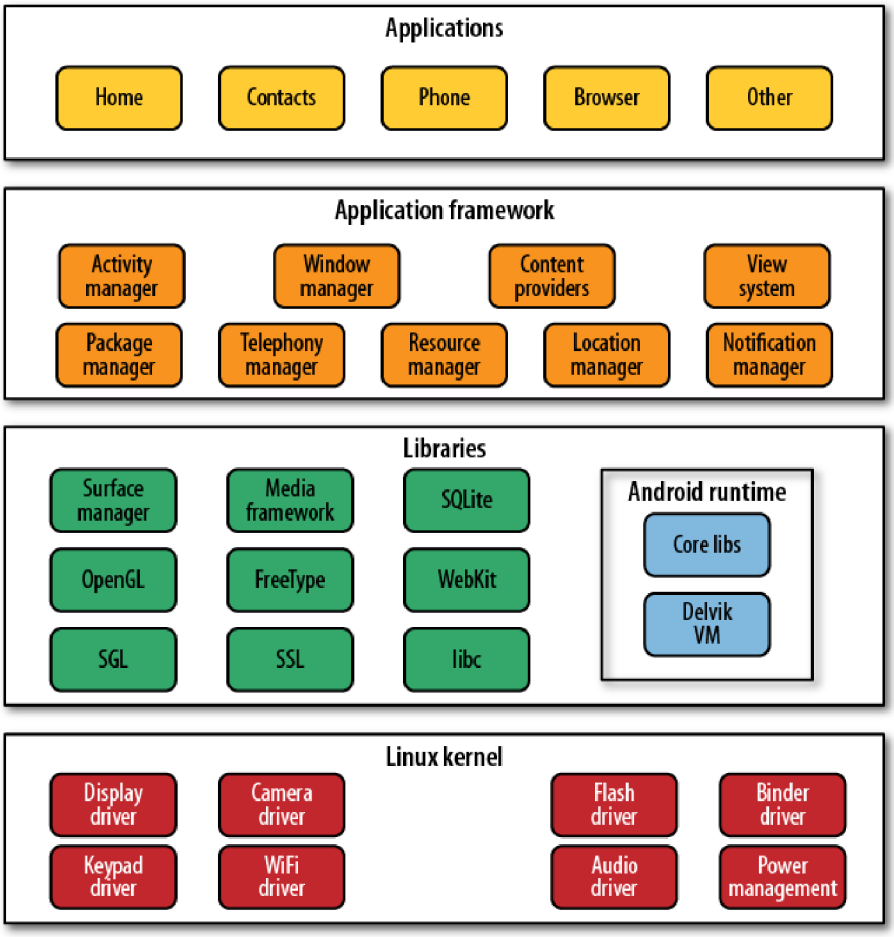
Linux je přenosná platforma, která se dá poměrně snadno přeložit na různých hardwarových architekturách. Android využívá úroveň hardwarových abstrakcí, která je definovaná na Linuxu. Na Androidu díky tomu není potřeba se starat o základní hardwarové funkce. Většina nízkoúrovňových funkcí byla implementovaná v přenositelném jazyku C, který umožňuje portovat Android na různá zařízení.

Bezpečnost

Linux je vysoce bezpečný systém vyzkoušený a ověřený v průběhu desetiletí. Android tedy, co se týče bezpečnosti, spoléhá na Linux. Všechny aplikace spuštěné na Androidu běží jako samostatné procesy na Linuxu s oprávněními, které stanovuje systém Linux. Android hned při startu předává řízení zavedením jádra Linuxu do operační paměti, čímž Linux přebírá kontrolu nad koordinací běžících procesů, správou paměti, správou sítí, atd.

3.2.2 Libraries

Další vrstvou po Linuxovém jádru je vrstva Libraries. Nativní knihovny jsou implementované v jazyce C/C++ a často se přejímají z open-source komunity za účelem poskytovat potřebné služby pro aplikační vrstvy . Android dále nabízí knihovny s celou řadou rozhraní API pro vývoj aplikací . Některé příklady knihoven jsou uvedeny na obrázku 3.1.



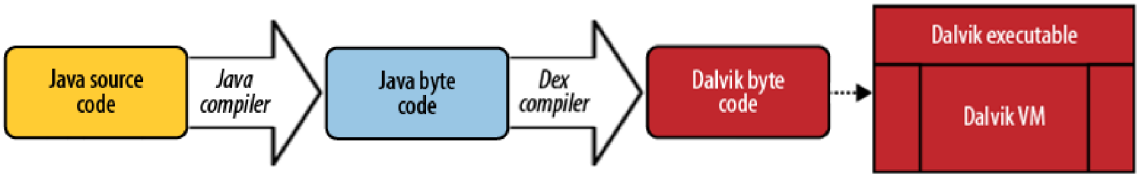
Obrázek 3.1: Architektura operačního systému Android. Převzato z [20].

3.2.3 Android Runtime

Tato vrstva obsahuje základní Java knihovny a Dalvik Virtual Machine (DVM), virtuální stroj speciálně vytvořený společností Google pro Android. Programátoři vyvíjí Android aplikace prostřednictvím jazyka Java, jehož knihovny jsou licencovány jako open-source, avšak virtuální stroj (JVM) pro překlad programu do spustitelné podoby již volně šiřitelný není. To je jeden z důvodů vzniku DVM. Jako další důvod figuruje optimalizace virtuálního stroje pro potřeby mobilních zařízení, kde jsou důležité vlastnosti výkon a úspora energie.

Jak již bylo zmíněno na začátku odstavce, obsahem této vrstvy jsou také knihovny programovacího jazyka Java, jejíž obsah lze téměř srovnat s platformou Java Standard Edition (SE).

Jak lze vidět na obrázku 3.2, Android aplikace implementované v jazyce Java jsou překládány do Java byte kódu, a nakonec do mezikódu pomocí Dalvik kompilátoru. Výsledný byte kód je spuštěn na DVM. Každá aplikace je samostatný proces s vlastní instancí DVM. Jedná se o jazyk interpretovaný.



Obrázek 3.2: Překlad z jazyku Java do spustitelného souboru na systému Android. Převzato z [20] a upraveno.

3.2.4 Application framework

Jedná se o nejdůležitější vrstvu pro vývojáře. Otevřená vývojová platforma operačního systému nabízí prostředí pro tvorbu různorodých aplikací. Aplikační rámec umožňuje přistoupit ke službám, které mohou vývojáři využívat přímo ve svých aplikacích. Mezi nabízenými službami jsou například Notification Manager, Activity Manager, Package Manager a jiné.

3.2.5 Application

V poslední, nejvyšší vrstvě jsou již vývojáři vytvořené Android aplikace. Některé z nich jsou již předinstalované, ostatní si může každý uživatel stáhnout a nainstalovat ručně či z online katalogu.

3.3 Vývoj aplikace

Při vývoji aplikací na operační systém Android se používá převážně třech programovacích jazyků – Java, XML a SQL. První z nich, jazyk Java, jehož překlad je již zmíněn v sekci 3.2.3, se používá k definování chování samotné aplikace. Značkovacím jazykem XML je implementováno vizuální rozložení jednotlivých oken. Poslední z uvedených, jazyk SQL, se využívá pro dlouhodobější uchování dat ve vnitřním databázovém systému SQLite.

Při založení nového projektu v jednom z vývojových prostředí se automaticky vytvoří adresářová struktura, kde jsou uloženy potřebné soubory.

Složka „src“ obsahuje veškeré zdrojové soubory v jazyce Java v daném jmenném prostoru.

Ve složce „res“ se uchovávají takzvané resources – statická data použitá při chodu aplikace. Mohou zde být uložené například různé layouty pro lišící se velikosti displeje různých zařízení, různé obrázky, zvukové stopy či texty pro lokalizace do různých jazyků.

Do složky „bin“ se ukládají veškeré soubory vytvořené při překladu – zkompilované třídy jazyka Java, knihovny potřebné pro správný chod aplikace, již zmíněné resources a výstupní soubor překladu, instalační soubor pro Android aplikace, který je označený koncovkou apk.

Důležitou součástí každého projektu je AndroidManifest.xml, který se nachází v kořenové struktuře. Tento soubor je základní kámen celé aplikace – popisuje danou aplikaci a všechny její použité komponenty, specifikuje verze, pro které je aplikace určená, určuje práva, která aplikace bude mít.

Nainstalovat právě přeloženou aplikaci je možné několika způsoby. Jedním z nich je nahrát instalační soubor na zařízení a ručně ji nainstalovat. Další způsob nabízí běžně používaná vývojová prostředí, u kterých bývá možnost provést instalaci aplikace na zařízení připojené pomocí USB kabelu přímo z vývojového prostředí. Pokud vývojář nevlastní zařízení s operačním systémem Android, lze využít varianty instalace na virtuální zařízení.

Díky němu je možné vyzkoušet aplikace na různá rozlišení displeje.

U některých vývojových prostředí také nemusí být nutná znalost jazyka XML, jelikož jejich GUI nabízí možnost vytvářet XML kód například pomocí uživatelsky přívětivé metody Drag and drop – přesouváním jednotlivých elementárních prvků do výsledné podoby rozložení displeje. Zároveň s přidáváním elementů se i automaticky generuje patřičný XML soubor.

3.4 Komponenty aplikace

Každá aplikace se skládá z volně vázaných komponent. Mezi nejdůležitější patří aktivity (Activity), služby (Services) a záměry (Intents).

Activity (Aktivity)

Aktivity v aplikaci reprezentují prezentační vrstvu. Jedná se o základní vizuální komponentu, která reprezentuje jednu obrazovku aplikace. Životní cyklus aktivity je podrobněji popsán v sekci 3.5.

Services (Služby)

Služby slouží k provádění dlouhotrvajících operací na pozadí. Uživateli neposkytují žádné graficko-uživatelské rozhraní. Službu může spustit komponenta aplikace a následně bude stále aktivně běžet na pozadí, i když uživatel přejde do jiné aplikace.

Intents (Záměry)

Záměr se obecně definuje jako abstrakce operace, kterou je potřeba vykonat. Celý aplikační prostor se v podstatě skládá z komponent (aktivity) a ze zpráv mezi komponentami (záměry). Záměr se obecně skládá z činnosti, která se má vykonat, parametru, nad kterým má být tato činnost vykonána, a aplikace, která má tuto akci provést.

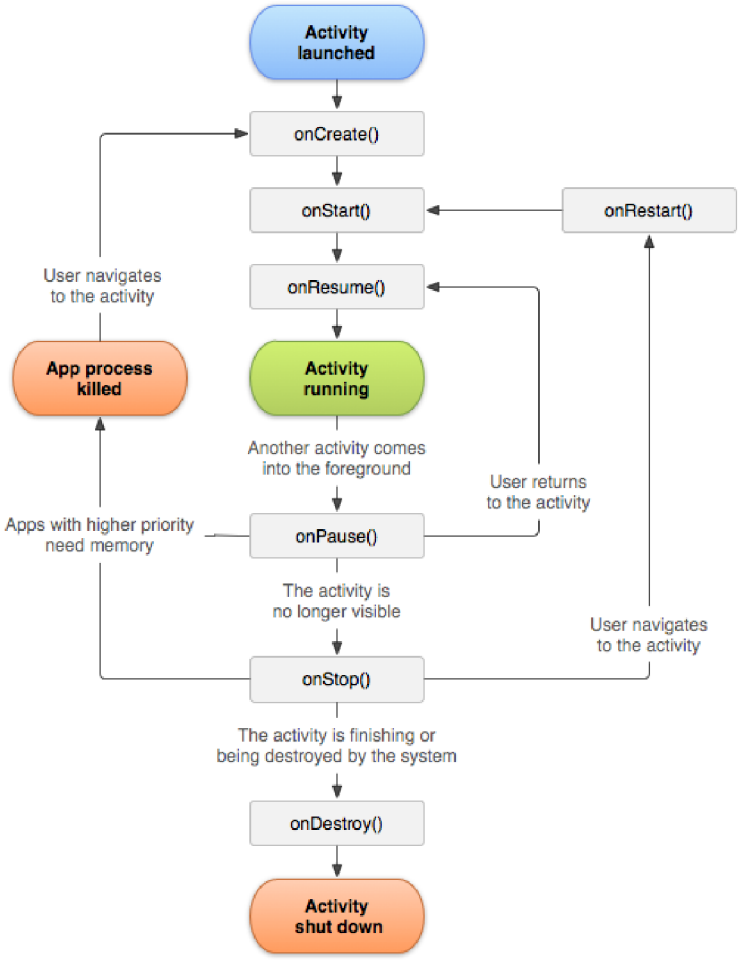
3.5 Životní cyklus aktivity

Aplikace se typicky skládá z více aktivit, kde právě jedna z nich je určená jako hlavní – zobrazí se po spuštění aplikace jako první. Na ni mohou následně navazovat další, které předchozí aktivitu pozastaví a uloží ji na zásobník LIFO (last in, first out). Podobný proces může následovat s dalšími aktivitami. Při ukončení aktivity se obnoví poslední uložená aktivita ze zásobníku.

Operační systém Android počítá s omezenými zdroji operační paměti, proto také vytvořila mechanismy pro zachování volného místa. Tento způsob je zobrazen na obrázku 3.3, z něhož je patrné, že v případě nedostatku paměti při spuštění nové aktivity se násilně ukončí aktivita ze zásobníku a tím vznikne volná paměť.

Každá aktivita obsahuje metody důležité pro životní cyklus aktivity. Mezi metody, které se provádí na začátku aplikace, patří `onCreate()`, která je volána hned při vytvoření aktivity, metoda `onStart()`, která se provádí, pokud se uživatel rozhodne vrátit do aktivity. Dále metoda `onPause()` se provádí, pokud dojde uživatelem k přesunu do jiné aktivity. Mezi ukončovací metody patří `onDestroy()`, jež proběhne při ukončení aktivity v rámci

aplikace, a `onStop()` volaná v době, kdy aktivita již není viditelná pro uživatele z důvodu pokračování komunikace jiné aktivity, jež se kryje s původní.



Obrázek 3.3: Schéma životního cyklu aktivity. Převzato z [10].

3.6 Uživatelské rozhraní

V aplikaci reprezentuje prezenční vrstvu vizuální komponenta zvaná aktivita, kde každá představuje jednu obrazovku aplikace. Aktivita se skládá ze základních komponentů třídy `View` – tlačítka, textová pole, výběrové seznamy atd., či komponentů třídy `ViewGroup`, která reprezentuje samotné rozvržení [33].

Dobré uživatelské rozhraní umožňuje uživateli pracovat s aplikací účelně a pohodlně. Správný návrh uživatelského rozhraní by měl splňovat následující body [1]:

- konzistence – podobné operace by se měly provádět stejným způsobem
- zpětná vazba – uživatel by měl být informován o prováděných operacích
- předcházení chyb – nedovolit uživateli zadat chybné údaje
- tolerance chyb – uživatel může zadat chybné údaje bez vážnějších následků

3.7 Android NDK

Zdroje [31], [26] a [2] pojednávají zavedení knihovny Android NDK a dále se zabývají potřebnými kroky pro vytvoření rozhraní mezi jazyky C++ a Java.

Android NDK (Native Development Kit) je sada nástrojů, které vývojáři povolují implementovat v aplikaci určené pro operační systém Android nejen v jazyce Java, ale i v C/C++. Je doporučeno použít NDK knihovny pouze ve výjimečných případech, mezi něž udává nutnost zvýšení výkonu (např. třídění velkých objemů dat), použití knihoven třetích stran (OpenCV, Ffmpeg, ...) či programování nízkoúrovňových aplikací. Použití nativního kódu totiž nemusí nutně znamenat zvýšení výkonu, ale vždy zvyšuje složitost.

Java Native Interface

Pro spuštění funkce napsané v jazyce C/C++ na Java Virtual Machine (JVM) je nutné rozhraní, skrz které se bude daná funkce volat. K tomuto účelu slouží Java Native Interface (JNI) – C/C++ kód je sestavený do dynamické knihovny, která poskytuje příležitost JNI k volání funkcí z programu napsaném v Javě.

JNI byl původně vyvíjený pro zajištění binární kompatibility na Oracle JVM, který je kompatibilní se všemi JVM. Hlavní výhoda JNI je schopnost spustit zkompilovaný C/C++ kód bez ohledu na platformu.

Implementace v Java kódu

Na straně Android aplikace se zpřístupnění nativních funkcí dělí na dvě fáze. Nejprve je nutné určit a načíst knihovnu, ve které se požadované funkce nachází. K tomu slouží funkce `System.loadLibrary("nazev knihovny")`, která požaduje řetězec jako parametr. Tento parametr udává název nativní knihovny, který je specifikován v souboru Android.mk.

Druhá fáze je určení konkrétní funkce, která tvoří rozhraní pro volání funkce z nativní knihovny. Deklarace funkce se provádí stejným způsobem jako u běžné funkce, jen je nutné přidat klíčové slovo `native`. Jako příklad je uvedena funkce s jedním parametrem:

```
private native void Stitching(String path);
```

Implementace v nativním kódu

Nativní funkce je nutné propojit s konkrétním rozhraním deklarovaným v Java kódu, což se vytváří následujícím způsobem:

```
jdouble Java_package_class_method (JNIEnv *ENV, jobject OBJ, jstring STR)
{
    const char *str = (*ENV)->GetStringUTFChars(ENV, STR, 0);
    (*ENV)->ReleaseStringUTFChars(ENV, STR, str);
    return 10;
}
```

V definici funkce se objevují tři parametry. První z nich `*ENV` je ukazatel na rozhraní, druhý parametr `OBJ` ukazuje na objekt, uvnitř kterého je nativní metoda deklarována, a poslední z nich `STR` reprezentuje předávané parametry. U předávaných parametrů musí dojít ke konverzi mezi datovým typem jazyku Java a jazyku C/C++. Konverze datového typu `String` je v popisovaném kódu naznačena prvním příkazem funkce.

Android.mk

Soubor Android.mk je makefile pro překlad nativní části projektu. Android.mk musí začínat definicí proměnné `LOCAL_PATH`, která určuje cestu ke zdrojovým souborům. Funkce `include $(CLEAR_VARS)` maže všechny proměnné s výjimkou `LOCAL_PATH`. Volání této funkce je důležité, protože překlad probíhá v jednom kontextu, kde jsou všechny proměnné globální. `LOCAL_MODULE` specifikuje název výstupního modulu, která bude vygenerována. Tento název musí být bez mezer a unikátní. Proměnná `LOCAL_SRC_FILES` obsahuje seznam zdrojových souborů jazyku C/C++. Posledním příkazem souboru Android.mk je funkce `include $(BUILD_SHARED_LIBRARY)`, která spouští překlad pro vytvoření dynamické knihovny.[13]

Application.mk

Soubor Application.mk definuje několik proměnných, kde se mezi nejdůležitější řadí proměnná `APP_ABI`, která určuje cílovou architekturu procesoru. Dále je možné využít proměnnou `APP_PLATFORM`, která definuje název cílové platformy, `APP_MODULES`, jež definuje seznam použitý nativních modulů, a mnoha dalších, které společně dělají překlad více flexibilní.

3.8 OpenCV

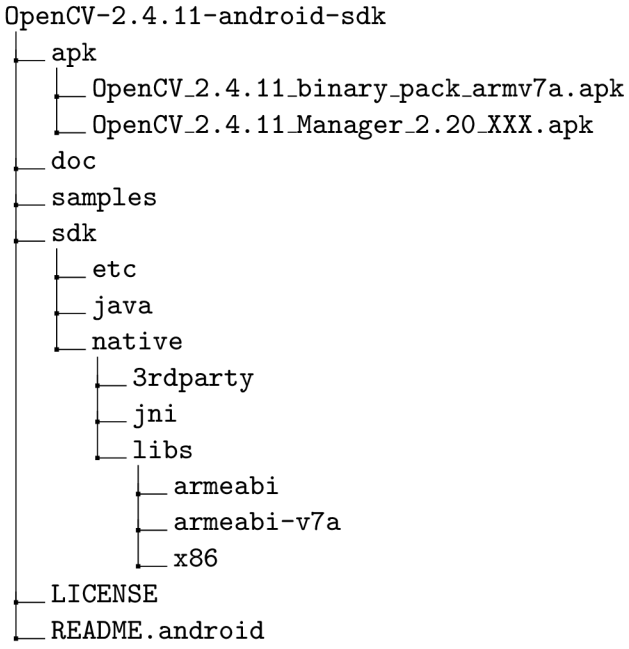
Knihovna OpenCV

OpenCV (Open Source Computer Vision Library) je open-source knihovna, která byla navržená tak, aby poskytovala společnou infrastrukturu pro aplikace pracující s počítačovým viděním. Je šířena pod licencí BSD, a proto je její použití zdarma pro akademické a částečně i pro komerční využití. Knihovna OpenCV nabízí rozhraní pro jazyky C/C++, Python, Java a řadí se mezi multiplatformní – podporuje Windows, Linux, Mac OS, iOS a Android. OpenCV byla navržena pro výpočetní ekfektivitu, se silným důrazem na aplikace v reálném čase. Je napsaná v optimalizovaném C/C++, může využít zpracování multi-core. [12]

OpenCV pro Android

Pro použití OpenCV na OS Android slouží balíček OpenCV4Android SDK. Struktura tohoto balíčku je zobrazena na obrázku 3.4.

Nejdůležitějším adresářem je `sdk`, který obsahuje OpenCV API a další potřebné knihovny pro Android. V dalším adresáři `doc` je dokumentace a v adresáři `samples` ukázkové zdrojové kódy pro vývojové prostředí Eclipse. Adresář `apk` obsahuje aplikaci OpenCV Manager API, která zpřístupňuje na zařízení s OS Android knihovnu OpenCV. [13] [6]



Obrázek 3.4: Struktura OpenCV4Android SDK. Převzato z [12].

Kapitola 4

Implementace

Tato kapitola se věnuje implementaci aplikace – popisuje konkrétní nástroje použité při vývoji, návrh aplikace a jednotlivé fáze, které provázely vývoj aplikace. Jsou zde popsány a řešeny problémy, které vznikly v průběhu vývoje.

4.1 Vývojové nástroje a prostředí

Pro vytvoření projektu bylo třeba znalosti dvou jazyků. Aplikace je napsána v jazyce Java, který je typický pro implementaci aplikací pro operační systém Android. Dále byl použit jazyk C++, kterým byla implementovaná část spojování fotografií.

Java Development Kit (JDK)

JDK je soubor základních nástrojů a knihoven pro vývoj aplikací pro platformu Java. Základní součástí tohoto balíčku jsou Java Runtime Enviroment, který slouží ke spuštění aplikací i vývojových nástrojů, překladač, debuger a další nástroje. Pro implementaci byl použit balíček OpenJDK verze 7.

Android Software Development Kit (SDK)

SDK je balíček vývojových nástrojů, jež umožňuje vytvářet aplikace pro různé operační systémy. Důležitou součástí SDK jsou knihovny API, dále pak dokumentace a různé ukázkové kódy. Android SDK, určený pro vývoj Java aplikací na platformu Android, je totožný balíček s SDK, jen je navíc rozšířen o emulátor pro simulaci zařízení s OS Android na počítači a dalšími pomocnými nástroji pro vývoj a ladění aplikací. Při vyvíjení aplikace byla použita verze 21.1.2.

Vývojové prostředí Eclipse

Eclipse je vedle Android Studia[1] nejpoužívanější nástroj pro vývoj Android aplikací. Původně je Eclipse nástrojem pro vývoj v jazyce Java, mezi jeho výhody však patří snadná rozšiřitelnost, kterou nabízí celá řada pluginů. Díky tomu poskytuje podporu dalších programovacích jazyků a nástrojů. Tato aplikace byla vytvářena pomocí vývojového prostředí Eclipse verze 3.8.1.

[1] http://developer.android.com/sdk/index.html

Android Development Tool (ADT)

ADT propojuje Eclipse s balíčkem Android SDK. Tím nám vzniká vývojové prostředí s editorem vizuálních aplikací, ladícími panely, tvorbou APK instalátorů, emulátorem OS Android a možností nahrávat aplikaci přímo na testované zařízení připojené pomocí USB kabelu. Android Development Tool byl použit ve verzi 23.0.4.

Knihovna AndroidNDK a OpenCV

Tyto knihovny byly již v této práci popsány – knihovna AndroidNDK se nachází v sekci 3.7, knihovna OpenCV v sekci 3.8. Při vývoji bylo použita knihovna OpenCV verze 2.4.10 a knihovna AndroidNDK ve verzi Revision 10d.

4.2 Návrh aplikace

Vývoj probíhal ve třech fázích. V první fázi byl vytvořen a testován algoritmus spojování fotografií. Implementace této části je v sekci 4.3. Ve druhé fázi, kterou se zabývá sekce 4.4, jsem vytvářel aplikaci pro OS Android, která pořídí sérii fotografií. Zde vznikal i návrh jednoduchého grafického rozhraní pro snazší ovládání pro uživatele. Třetí fáze, popsaná v sekci 4.5, propojovala první dvě – algoritmus spojování fotografií byl implementován do zdrojového kódu aplikace a bylo vytvořeno rozhraní pro jeho volání. Aplikace byla dále dolaďována a upravována do finální verze.

4.3 První fáze – spojování

Cílem první fáze bylo vytvoření algoritmu pro spojování fotografií. Tento algoritmus byl vytvářen na OS Ubuntu 14.04, na němž byl kompilován pomocí překladače GCC verze 4.8.2. Zdrojový kód algoritmu i jeho přeložená verze jsou uloženy na přiloženém CD (viz příloha A). Zjednodušené schéma chování je zobrazeno v algoritmu 3.

Algoritmus 3: Spojování fotografií

```
1: načítám kolekci
2: vyhledávám deskriptory
3: while existuje nespojená dvojice do
4:     vyhledávám mezisnímkové korespondence
5:     if snímky spolu sousedí then
6:         spojuji snímky
7:         ukládám výsledný snímek
8:         vyhledávám deskriptory
9: for všechny nalezené korespondence do
10:    spojuji i s prolínáním
11: ukládám výsledné panorama
```

4.3.1 Načítání kolekcí

Prvním krokem algoritmu je vytvoření dočasné složky `./tmp`, do které se zkopíruje celá kolekce fotografií. K tomuto kroku jsem přistoupil z důvodu oddělení originálních fotografií

od těch, které se zpracovávají. Navíc při kopírování dochází k přejmenování na co nejkratší název – dvouciferné číslo a přípona souboru. Důvod přejmenování je uveden v části 4.3.8.

4.3.2 Vyhledávání deskriptorů

Pro každý zkopírovaný soubor se následně určují klíčové body pomocí SURF detektoru (viz část 2.4), z kterých SURF deskriptor vyhledává deskriptory. Vypočtené deskriptory jsou uloženy do souboru se stejným názvem jako zpracovávaná fotografie s příponou `xml`. Uložení do souboru předcházím vysokým paměťovým nárokům, které by nastaly při zachování deskriptorů v proměnných programu. V algoritmu jsou vždy v paměti uloženy deskriptory maximálně ze dvou fotografií. Jeden soubor s deskriptory může mít velikost i kolem 50MB.

4.3.3 Mezisnímková korespondence

Po nalezení deskriptorů algoritmus vyhledávání určí dvojici fotografií (výběr popsán v části 4.3.10), která se bude zpracovávat. Pomocí FLANN algoritmu dojde k vyhledání podobných deskriptorů. Výsledné dvojice deskriptorů jsou ještě filtrovány na základě vzdálenosti mezi vyhledanou dvojicí. Ty dvojice, které prošly filtrem, už považuji za odpovídající. Pokud počet odpovídajících dvojic je dostatečně velký, pokračuji ve spojování, v druhém případě pokračuji v algoritmu hledáním korespondencí mezi další dvojicí fotografií.

4.3.4 Spojení snímků

Vybrané odpovídající dvojice deskriptorů slouží k výpočtu matice homografie. V algoritmu se vypočítávají dvě homografie pro obě možné varianty – homografie pro spojení prvního a druhého snímku a homografie pro opačné pořadí snímků, tedy druhého a prvního. Z těchto dvou homografií se vybere ta lepší, která se následné aplikuje.

4.3.5 Překrývání snímků

Pro spojení snímků se standardně využívá funkce `warpPerspective()` knihovny OpenCV. Ta funguje dobře, pokud spojujeme dva snímky, které jsou zaplněné v celé své části viditelnými body. Výsledné panorama má být ale tvořeno vícero snímky za sebou, čímž na průběžném snímku dvou a více fotografií vznikají transparentní místa – bod, jehož aplha kanál je roven nule. Bohužel zmíněná funkce v případě přidání snímku neignoruje transparentní body přidávaného snímku a přepíše jimi viditelné body podkladového snímku, tudíž pro mou potřebu je tato funkce nepoužitelná. Tato situace je zobrazena na obrázku 4.1.

V knihovně OpenCV jsem nenašel funkci podobné `warpPerspective()`, která by vykonávala stejný úkon a zároveň by ignorovala transparentní body. Z tohoto důvodu jsem napsal vlastní funkci pro přidávání snímku, která zároveň, pokud je to požadováno, provádí vyhlazování přechodu mezi oběma snímky. Vyhlazováním se podrobněji zabývá část 4.3.6. Funkce přidávání obrazu je popsána v algoritmu 4. Bohužel tato funkce prochází každý pixel obrazu, což není moc efektivní způsob.

4.3.6 Vyhlazování přechodů – blender

Blender je algoritmus pro prolnutí dvou a více obrazů. Cílem blenderu je vytvořit výsledný obrázek, v němž nebudou vidět žádné ostré přechody mezi původními snímky. Mezi často

Obrázek 4.1: Vlevo je podkladový obraz, který je spojován s prostředním obrazem. Výsledek funkce `warpPerspective()` je znázorněn na pravém obrazu. Obraz je ohraničen černým rámem, bílá barva je ve skutečnosti průhledná. Průhledný pravý dolní roh prostředního obrazu překreslí podkladový obraz. Převzato z [8] a upraveno programem.

Algoritmus 4: Překreslování snímků

```
for každý řádek do
  for každý sloupec do
    if přidávaný pixel na daném řádku a sloupci je viditelný then
      if nastavené vyhlazování then
        určím průhlednost přidávaného bodu
        určím hodnotu pixelu na základě průhlednosti a bodů z obou obrazů
        uložím hodnotu pixelu na dané souřadnice podkladového obrazu
      else
        uložím přidávaný pixel na dané souřadnice podkladového obrazu
```

používané algoritmy blenderu[2] patří lineární a vícepásmový.[25]

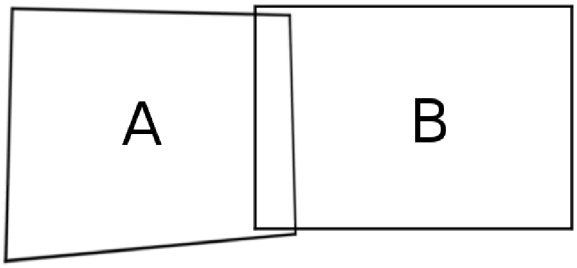
Lineární algoritmus je základní a velmi rychlý algoritmus. Nevýhodou této metody jsou viditelné švy, rozmazání a viditelní duchové (rychle pohybující se objekty zachycené na různých místech na obou snímcích). Pokud nejsou kladeny požadavky na přílišnou kvalitu je tento algoritmus dostačující.[25]

Vícepásmový (multiple-band) algoritmus je už pokročilejší metoda dosahující lepších výsledků než lineární, ale je také pomalejší. Tento algoritmus zaručuje plynulé přechody mezi snímky navzdory rozdílu osvětlení. Přechodové oblasti budou stěží viditelné.[25]

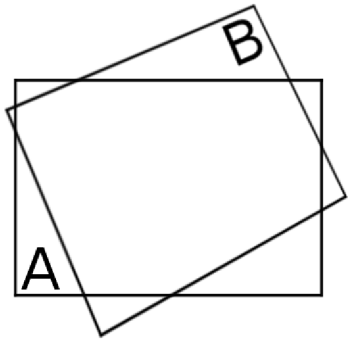
V panoramatu jsou snímky přidávány vedle sebe, dochází tedy k překrytí pouze jedné hrany, jak zobrazuje obrázek 4.2. U pořízených fotografií z výšky dochází k překrytí, které je znázorněné na obrázku 4.3. Může dojít k překrytí až na všech čtyř hranách. V tomto případě jsem nebyl schopen použít klasických metod blenderu, proto jsem vytvořil vlastní metodu.

Vyhlazení přechodů probíhá společně s překrýváním obrazů, kde pro každý bod přidávaného snímku je určena jeho váha viditelnosti (opacity). Ta reprezentuje procento, jak moc má být viditelný bod přidávaného obrazu. Doplňkem 100% k získanému procentu získáme váhu viditelnosti podkladového snímku. Pro představu je přidávaný snímek po aplikaci váhy

[2] Porovnání metody je na těchto stránkách http://www.kolor.com/wiki-en/action/view/Interpolation_and_blenders



Obrázek 4.2: Překrytí snímků u klasického panoramatu. (vlastní práce)



Obrázek 4.3: Překryv testovacích snímků z výšky. (vlastní práce)

viditelnosti zobrazen na obrázku 4.4 vlevo dole. V této metodě vyhlazování se nevytváří takový snímek, ale rovnou se určuje výsledný snímek. Kanály bodů výsledného snímku jsou určené následujícím způsobem:

```
// 1. kanál, složka B, modrá
vystupBod[0] = pridavanyBod[0] * opacity + podkladovyBod[0] * (1 - opacity);
// 2. kanál, složka G, zelená
vystupBod[1] = pridavanyBod[1] * opacity + podkladovyBod[1] * (1 - opacity);
// 3. kanál, složka R, červená
vystupBod[2] = pridavanyBod[2] * opacity + podkladovyBod[2] * (1 - opacity);
// 4. kanál, složka aplha
vystupBod[3] = 255;

vystupniObraz[x,y] = vystupBod;
```

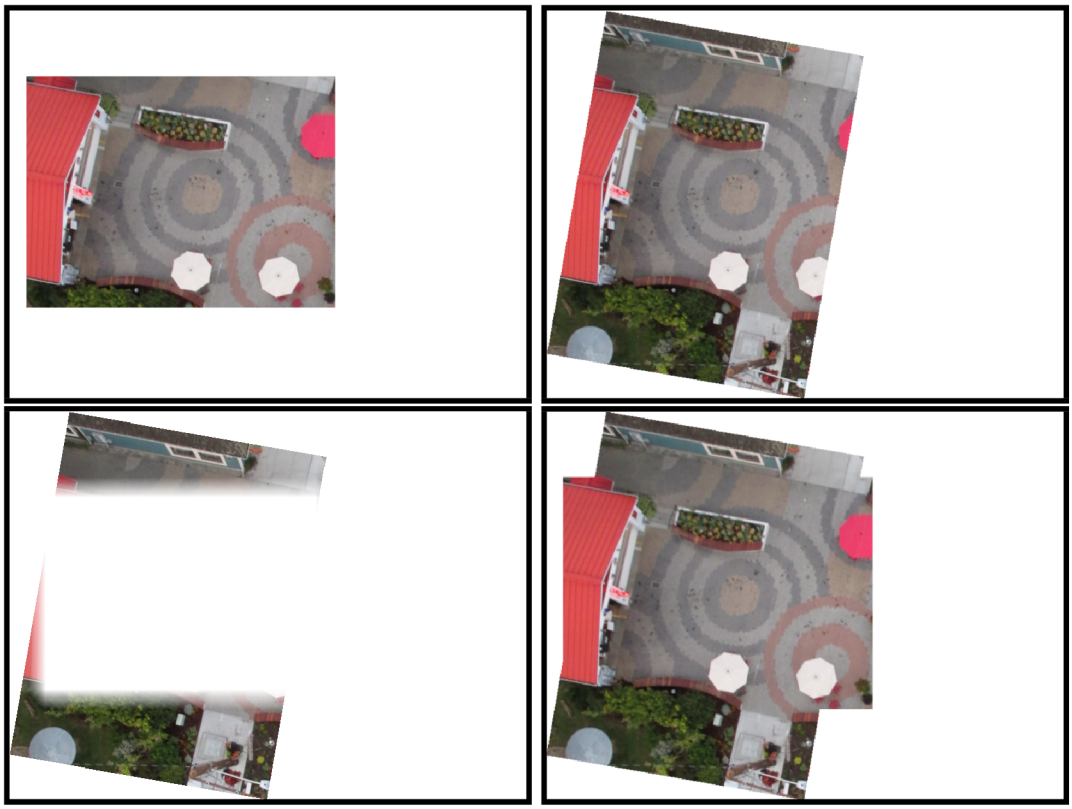
4.3.7 Určení váhy viditelnosti

Váha viditelnosti (opacity) se určuje pro každý bod přidávaného obrazu. Může nabývat hodnot od nuly do jedné, kde nula reprezentuje transparentní bod a jednotka viditelný.

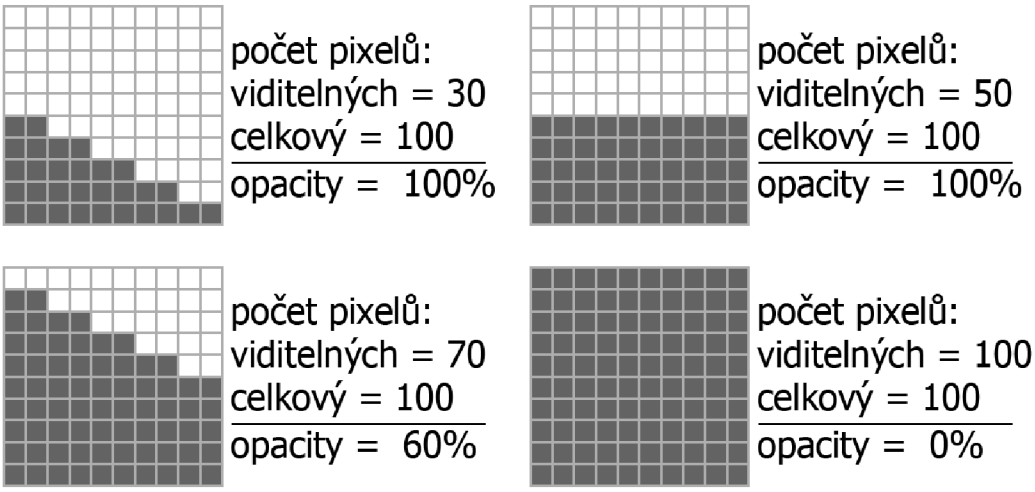
Máme bod na přidávaném obraze o daných souřadnicích, který je viditelný. Určím čtvercové okolí bodu v podkladovém obraze o hraně délky $2d$, kde d je hloubka prolínání. Hodnota váhy viditelnosti závisí na procentu viditelných pixelů v okolí. Tato závislost je popsána v tabulce 4.1. Z tabulky je patrné, že lineární závislost je pouze v případě rozsahu od 50% do 100% viditelných pixelů, v ostatních případech se jedná o konstantní závislost. Převedení procent viditelných pixelů na váhu viditelnosti je zobrazené v algoritmu 5.

Procent viditelných pixelů	0%	0% → 50%	50% → 100%	100%
Váha viditelnosti	0%	0%	0% → 100%	100%

Tabulka 4.1: Převod mezi procentem viditelných pixelů v okolí podkladového obrazu a váhou viditelnosti přidávaného bodu.



Obrázek 4.4: Nahoře jsou zobrazeny dva spojované obrazy – vlevo podklad, vpravo přidávaný. Levý obraz dole je výstup vyhlazení přechodu s hloubkou prolínání 30 pixelů. Vpravo dole je už výsledný obraz – spojený podklad a obraz s vyhlazeným přechodem. Obraz je ohraničen černým rámem, bílá barva je ve skutečnosti průhledná. Převzato z [8] a upraveno programem.



Obrázek 4.5: Ukázky okolí podkladového snímku a k němu odpovídající procento opacity. (vlastní práce)

Algoritmus 5: Určení váhy viditelnosti (opacity)

```
for každý řádek okolí do
  for každý sloupec okolí do
    if pixel podkladového obrazu na daném řádku a sloupci je viditelný then
      inkrementuji počet viditelných pixelů
určím procentuální podíl viditelných pixelů, výsledek v rozmezí 0% − 100%
od tohoto podílu odečtu hodnotu 0.5, výsledek v rozmezí −50% − 50%
oříznu záporné výsledky, výsledek v rozmezí 0% − 50%
vynásobím hodnotou 2, získám v opacity pro bod podkladového snímku
určím doplněk – opacity pro bod přidávaného snímku
```

4.3.8 Názvy pro ukládání

Originální snímky pro spojování jsou kopírovány a ukládány pod jiným názvem. Tento název je složen z dvouciferného čísla a přípony originálního souboru.

Název souboru s uloženými deskriptory je vždy shodný s názvem snímku, ke kterému deskriptory patří. Knihovna OpenCV nabízí k uložení deskriptorů dvě přípony – `xml` a `yaml`. K uložení jsem zvolil mně známější jazyk `xml`.

Průběžné mezivýsledky se ukládají pod názvem, jež je složen z přípony a kombinace separátorů a názvů snímků obsažených v daném mezivýsledku. Názvy jednotlivých snímků jsou pro zpětné parsování a také pro přehlednost odděleny separátorem. V algoritmu je jako separátor použit znak `'@'`.

Posledním souborem, který algoritmus ukládá, je finální snímek. Tento souboru jako jediný z vytvořených není dočasný, přetrvává i přes skončení celého algoritmu. Z tohoto důvodu byl pro tento snímek použit relativně unikátní název generovaný na základě aktuálního data a času. Tím by se mělo zamezit nechtěnému přepisování stejně pojmenovaných souborů.

Při ukládání souboru jsem narazil na omezení délky názvu souboru na operačním systému Android. Při pokusu vytvoření a uložení souboru, jež přesáhl maximální velikost názvu, došlo k pádu aplikace. Bohužel jsem nikde v oficiálních zdrojích tuto informaci nenašel, proto následující závěry jsou pouze úvahou. Na linuxových operačních systémech je omezení délky názvu souboru na 255 znaků. OS Android je postaven na linuxovém jádře, tudíž by mohl mít stejné omezení. Experimentálně vyzkoušeno na Androidu API 15, kde se tato úvaha zdála pravdivá.

4.3.9 Omezení a filtrování špatných mezivýsledků

Již od první verze se aplikace potýká se špatně spojenými fotografiemi. Tedy její důležitou částí je vytvoření a aplikování pravidel omezení, které vedou k lepší výsledkům.

Vyhledávání deskriptorů

První filtrace probíhá při vyhledávání podobných deskriptorů (viz část 4.3.2). V této části algoritmus FLANN (viz část 2.5.2) vybere seznam dvojic deskriptorů. Tuto dvojici algoritmus následně seřadí podle vzdálenosti mezi oběma deskriptory a najde minimální vzdálenost. Na základě této vzdálenosti je filtrován seznam dvojic deskriptorů – vytváří se nový seznam, do

kterého se ukládají pouze vyfiltrované výsledky. Tento filtr je přebrán z oficiálních stránek knihovny OpenCV [7]. Filtr je dán následující podmínkou:

```
match.distance <= max(2*min_dist, 0.02)
```

Ve spojování zpracovávaných snímků se pokračuje se seznamem dvojic deskriptorů, pokud počet dvojic vybraných filtrem je větší než 10. Tato hranice je mnou definovaná, byla určená na základě experimentování.

Matice homografie

Další omezující pravidlo je použito u matice homografie (popsána v sekci 2.7), která se skládá ze tří sloupců a tří řádků. Jako validní matici považuji tu, která ani v jednom prvku nepřekračuje mnou definované meze. U matic s prvky přesahující tyto meze se dá předpokládat, že snímek s aplikovanou homografií bude příliš zdeformovaný. Proto při překročení meze dochází k přerušení spojování zpracovávané dvojice snímků. Horní mez je určená hodnotou 3000, dolní mez hodnotou -3000. Meze byly určené na základě testování.

Procento zakrytí mezivýsledku

Další selekce nevalidních výsledků je založena na procentu viditelných bodů spojeného snímku. I přes určení mezí prvků matice homografie se ve výsledcích objevovaly snímky, které byly špatně spojené. Homografie přidávaný snímek zdeformovala tak, že překryl celou plochu výsledného snímku. Z těchto zkušeností bylo zavedeno omezení založené na základě procentuálního zastoupení viditelných bodů. Pokud je toto procento v rozmezí $10\% - 90\%$ je spojení dvou snímků považované za validní. Procentuální rozmezí bylo určené experimentováním s problémovými snímky.

Omezení počtu fotografií

V sekci 4.3.8 je popsán problém s délkou názvu snímku. Z těchto důvodů byl omezen počet zpracovávaných snímků na 80. Tato hodnota byla určena následujícím výpočtem.

Předpokládá se maximální délka názvu souboru 255 znaků, z čehož jsou 4 znaky použity pro příponu souboru (`.jpg`, `.png`, či `.xml`), zbývajících 251 znaků je vyhrazeno pro název souboru. Ten se skládá z názvů jednotlivých spojených snímků oddělených separátorem (viz 4.3.8) – jedná se tedy o tři písmena (např. `01@`). Výsledný nejvyšší počet snímků pro spojování je vypočtený následujícím podílem:

$$\frac{\text{počet znaků pro název}}{\text{počet znaků pro jeden snímek}} = \frac{251}{3} = 83.\overline{6} \doteq 80$$

4.3.10 Vyhledávání dvojic

Na správném výběru dvojic fotografií závisí úspěch výsledného panoramatu. První aplikovaným řešením byla metoda, jež vyhledává mezisnímkové korespondence každé fotografie s každou v rámci jedné kolekce a nově vzniklých mezivýsledků. Tato metoda posloužila k získání prvních výsledků aplikace, které vznikly s testovací kolekcí čtyř fotografií. Výsledky byly výborné, jelikož se vždy aplikovala nejlepší homografie. Při testování s větší kolekcí fotografií se ale prokázala značná časová i paměťová náročnost této metody. Pokusy

o její optimalizaci byly úspěšné, avšak doba potřebná k získání výsledného panoramatu byla příliš vysoká.

Další metoda použitá pro řešení tohoto problému je daleko efektivnější, ale projeví se na výsledné kvalitě. Je založena na myšlence, že fotografie jsou pořizovány v relativně rychlém časovém sledu za sebou. Díky tomu lze předpokládat, že následující snímek bude mít překryv s právě vyfoceným. Algoritmus uloží první snímek jako mezivýsledek a poté v cyklu prochází jednotlivé fotky kolekce. Pokud daná fotografie nebyla ještě přidána, snaží se ji spojit s mezivýsledkem (viz algoritmus 6). Tato metoda je efektivní, jelikož nevytváří žádné spojení, které by nebylo použité ve finálním obraze.

Algoritmus 6: Vyhledávání dvojic

```
1: první snímek určím jako mezivýsledek
2: while je uložený snímek snímek v minulém cyklu do
3:   for každý snímek z kolekce do
4:     if mezivýsledný snímek a snímek z kolekce nebyl zpracován then
5:       zpracovávám mezivýsledný snímek a snímek z kolekce
6:       pokračuji ve spojování snímků, viz algoritmus 3
```

4.4 Druhá fáze – tvorba Android aplikace

Cílem této fáze je vytvořit a otestovat aplikaci pro operační systém Android. Aplikace má zatím za úkol provést nafocení série snímků. V této části také vznikal návrh jednoduchého grafického rozhraní, které slouží pro snadnou komunikaci mezi aplikací a uživatelem.

4.4.1 Chování aplikace

Při spuštění aplikace se namapují prvky uživatelského rozhraní a nastaví se v nich defaultní hodnoty. Dále je třeba provést inicializaci kamery, která je použitá pro zachycení snímků. Pokud má zařízení více kamer, je vybrána ta hlavní a následně je otevřena funkcí `Camera.open()`, jež vrací instanci kamery. V tuto chvíli je možné nastavit parametry kamery – nastavuje se povolení automatického ostření, vypnutí blesku a režim „Sport“, pokud jej kamera nabízí. Posledním krokem inicializace kamery je nastavení náhledu, jež provádění funkce `camera.setPreviewTexture()`. Jelikož není náhled potřeba, ale jeho přiřazení kameře je nutné pro zachycení snímku, předává se této funkci instance třídy `SurfaceTexture`.

Aplikace má dvě základní funkce chování. První z nich vytváří sérii fotografií, druhá spojuje fotografie v určené složce. Obě dvě na svém začátku zakazují možnost změny u všech prvků uživatelského rozhraní a získají uživatelem zadané parametry.

Při přerušení i zrušení aplikace se metodou `camera.release()` uvolňuje instance kamery. Tento krok je důležitý, protože na systému Android může být vytvořena pouze jedna instance kamery. Proto v případě neuvolnění by nešly spustit další aplikace využívající kameru. Chování při přerušení aplikace závisí také na stavu aplikace. Pokud aplikace snímá kolekci fotografií, je tato činnost ukončena, z důvodu uvolnění kamery. Pokud aplikace spojuje fotografie, algoritmus spojení není přerušen a probíhá dál na pozadí.

Všechny výstupní soubory jsou uloženy ve složce „BirdsPanorama“, která je vytvořená při spuštění aplikace. Tato složka je umístěná v adresáři určené systémem pro ukládání

obrázků. Jsou zde ukládána výsledná panoramata a složky s jednotlivými kolekcemi pořízených fotografiích. Dále se zde může nacházet složka „img“ s fotografiemi určenými pro spojení.

Funkce focení a spojování

Při zvolení této funkce se provedou přípravy pro zachycení snímků. V rámci příprav se vytvoří cílová složky pro uložení fotografií. Tato složka je vytvořena pod názvem složeným z aktuálního data a času.

Následně se nastavují uživatelem zadané parametry – v kameře je nastaveno rozlišení snímaných fotografií. Poté je využit další parametr – zpoždění před začátkem focení. Toto zpoždění je implementováno pomocí třídy `Handler` a její metodou `postdelay(r, time)`, kde parametr `time` je čas v milisekundách určující zpoždění pro spuštění vlákna `r`.

Po uplynutí času zpoždění je spuštěn časovač třídy `CountDownTimer(suma, interval)`, který po dobu určenou parametrem `suma` každých `interval` milisekund zavolá svou metodu `public void onTick()`. Oba parametry jsou vypočtené z uživatelem zvoleného nastavení počtu fotografií a dobou mezi zachycením dvou fotografií. V metodě `onTick()` se již volá metoda pro zachycení fotografie `takePicture(null, null, mPicture)`, která je volána nad instancí kamery. Předávaný parametr `mPicture` je instance třídy `PictureCallBack`, ve které se provádí zpracování pořízeného snímku – je zde vytvořen nový soubor, do kterého je následně uložena zachycená fotografie. Nezbytným krokem pro zachycení další fotografie je znovuobnovení náhledu, které provedeme dvěma příkazy `camera.stopPreview();` `camera.startPreview();`. Po vypršení doby časovače se automaticky volá jeho metoda `onFinish()`, v které vzniká nové vlákno pro spojování fotografií.

Funkce spojování

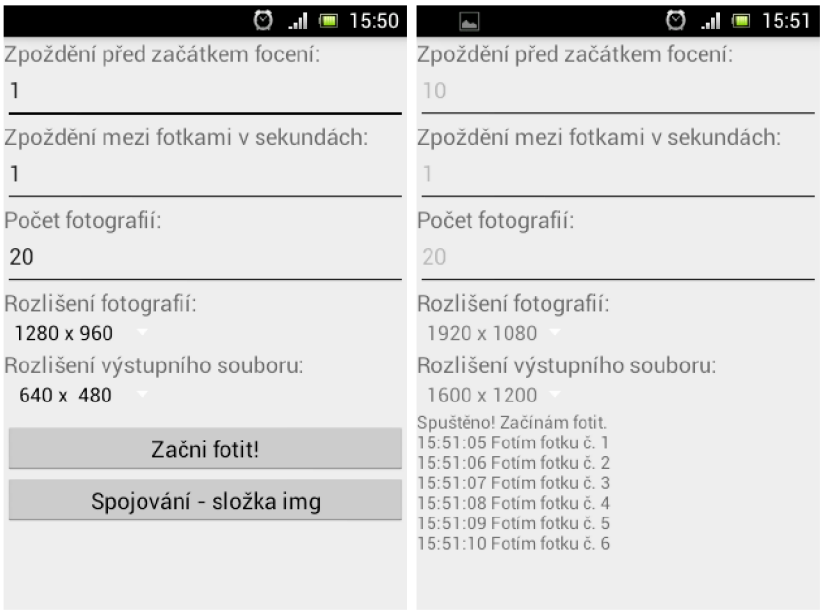
Tato funkce vynechává činnost zachytávání fotografií a spouští v novém vlákně algoritmus pro spojování. Předpokládá se, že složka „img“ obsahuje fotografie určené ke spojování. Pokud tato složka neexistuje, aplikace oznámí tuto skutečnost uživateli, jinak se spouští algoritmus spojování fotografií.

4.4.2 Uživatelské rozhraní

Uživatelské rozhraní nabízí uživateli nastavení potřebných parametrů a informuje uživatele o stavu, v kterém se aplikace nachází. Aplikace nepatří mezi ty, které by uživatel používal po delší dobu – uživatel nastaví parametry, spustí ji, připne na létající aparátu a nechá běžet. Z tohoto důvodu bylo navrženo jednoduché rozhraní3.6, u kterého lze vše potřebné nastavit z hlavního menu. Screenshot vzhledu hlavního menu je na obrázku 4.6.

Editovatelné položky

Hlavní menu aplikace nabízí tři editovatelné položky, ke kterým se vážou i popisky objasňující jejich význam. První položkou je zpoždění před začátkem focení, které udává dobu potřebnou k připnutí zařízení se spuštěnou aplikací k létajícímu aparátu a vypuštění do dostatečné výšky. Další položkou je zpoždění mezi fotkami a poslední položkou je počet fotografií. Všechny položky reprezentují číselný údaj, proto vkládaný znak je omezen pouze na číslice.



Obrázek 4.6: Ukázky uživatelského rozhraní aplikace – vlevo vzhled hned při startu, vpravo vzhled při pořizovaní fotografií. (vlastní práce)

Výběrové pole

Pod editovatelnými položkami jsou dvě výběrová pole, která slouží k získání rozlišení. První z nich udává rozlišení pořizovaných fotografií. Toto pole je inicializováno při začátku operace a jeho obsah záleží na parametrech kamery, z které jsou získávána podporovaná rozlišení. Defaultně je nastavené nejvyšší podporované rozlišení, jehož šířka je maximálně 1400 pixelů.

Druhým výběrovým polem je rozlišení výstupního obrazu. Obsah tohoto pole je pevně dán. Uživatel má na výběr z pěti rozlišení: 640×480, 1200×900, 1600×1200, 2800×2100 a 4000×3000.

Tlačítka

O spuštění činnosti aplikace se starají dvě tlačítka, kde každé z nich je navázané na příslušnou metodu v Java kódu. První z nich má název „Začni fotit!“ a spouští se jím zachytávání snímků. Druhým tlačítkem „Spojování – složka img“ se spouští operace spojování snímků. Obě dvě činnosti prováděná při stisku jsou popsaná v části 4.4.1.

4.4.3 Zachycení fotografie

Největší problém při vytváření Android aplikace vznikl při implementaci zachycení snímku z kamery. Prvotní jednoduchá aplikace pro zachycení jednoho snímku byla naimplementovaná pomocí oficiálního návodu [9], který využívá třídu `android.hardware.Camera`, která je označována za zastaralou pro API 21. Aplikace nebyla funkční i přes různé úpravy kódu a hledáním rad na oficiálních i neoficiálních fórech.

Druhá varianta implementace využívala třídy `android.hardware.camera2`. Jako testovací zařízení byl použit LG Nexus 5 s API 21, takže použitou třídu zařízení již podporuje. Při spuštění této aplikace byl výsledek stejný jako při první variantě.

Dalším krokem byla konzultace s vývojářem aplikací pro Android, který mi pomohl odladit první variantu, tedy oficiálně doporučený způsob. Bohužel i s jeho zkušenostmi se nepodařilo aplikaci uvést do funkčního stavu. Proto navrhoval použít knihovnu využívající novou třídu `android.hardware.camera2` Android Camera2Basic[3], která poskytuje rozhraní pro připojení a ovládání kamery. Ovšem opět neúspěšně. Na jeho radu jsem ještě vyzkoušel knihovnu CWAC-Camera[4], která vytváří zjednodušené rozhraní pro práci s kamerou. Dvouhodinová konzultace mi pomohla ujasnit si některé vývojářské praktiky, ovšem aplikaci se nepodařilo zprovoznit. Všechny varianty popsané v tomto odstavci byly testovány na dvou různých zařízeních LG Nexus 5 s API 21 a třetím zařízení Sony Ericsson Xperia mini s API 14.

Po týdenní odmlce jsem se opět vrátil k aplikaci za účelem přesného definování problému pro jeho zveřejnění na oficiálním fóru Android vývojářů. Při návratu zdrojového kódu k první variantě jsem očekával chybové hlášení. Pokus o spuštění aplikace byl ale úspěšný a zachycený snímek byl uložen do souboru. Nedokáži vysvětlit, kde nastala chyba – předchozí chybová hlášení nejsou zachovaná. Od této doby už podobné problémy se zachycením snímku neobjevují.

4.5 Třetí fáze finální aplikace

Cílem třetí fáze je propojit grafické rozhraní aplikace, vytvořené v druhé fázi (sekce 4.4), s algoritmem pro spojování fotografií, vytvořeným v první fázi (sekce 4.3). Jak bylo popsáno v sekci 3.7, pro zprovoznění C/C++ funkce v Android aplikaci jsou nejdůležitější tyto tři kroky: implementovat její rozhraní pro volání, definovat soubory `Android.mk` a `Application.mk`. Všechny soubory spojené se zprovozněním C/C++ zdrojových kódu jsou uloženy ve složce „jni“, která se nachází v kořenovém adresáři projektu.

4.5.1 Android.mk

Tento soubor je makefile zdrojového kódu C/C++. Jsou zde definované potřebné knihovny, mezi které patří knihovna `libopencv_java.so` a `libnonfree.so`. V neposlední řadě je tu definován překlad souboru `PhotoStitching.cpp`, který obsahuje zdrojový kód spojování snímků.

4.5.2 Application.mk

Mezi důležité definice tohoto souboru patří definování cílové architektury procesoru. Aplikace byla vyvíjená pro zařízení s ARM čipy, proto jako cílová architektura byla zvolena `armeabi-v7a`. Dále je tu definovaný název cílové architektury `android-8` a moduly použité pro překlad.

4.5.3 Rozhraní Java a C/C++

Algoritmus spojování snímků potřebuje ke spuštění tří parametrů – cesta ke složce „BirdsPanorama“, název složky s fotografiemi a rozlišení výsledného snímku. V zdrojovém kódu Java je rozhraní implementováno následující definicí:

```
private native void Stitching(String path, String folder, String size);
```

[3] Bližší informace na adrese: https://github.com/googlesamples/android-Camera2Basic
[4] Více na adrese: https://github.com/commonsguy/cwac-camera

Této definici metody v jazyce Java odpovídá definice v jazyku C/C++, která tím dokončuje vytvoření rozhraní mezi oběma funkcemi. Název této metody je určen cestou jednotlivými balíčky až k definované Java metodě `Stitching`. V C/C++ je uvedena tato definice:

```
void Java_com_photo_Main_Stitching
(JNIEnv* env, jobject t, jstring jpath, jstring jfolder, jstring jsize);
```

Po vytvoření rozhraní je nutné převést Java String na řetězec jazyk C/C++. Tato konverze byla prováděna následující funkcí:

```
const char *cString = env->GetStringUTFChars(javaString, 0);
```

Po převodu potřebných parametrů se už může přistoupit k zavolání hlavní funkce algoritmu spojování z první fáze 4.3.

4.5.4 Problém předávaní parametrů

Při spuštění výše vzniklého rozhraní, v kterém dochází k předávání parametru z Java kódu, probíhalo spojování v pořádku. Pro lepší informování uživatele o právě zpracovávaných fotografiích jsem vytvořil nové rozhraní pro přidání textu na displej. Zde nastal problém s knihovnou OpenCV, u které přestaly fungovat některé funkce. Stejný problém nastal také při předávání návratové hodnoty výsledku spojovacího algoritmu. Z těchto důvodů jsem ustoupil od předávání informací ze strany zdrojového kódu C/C++. Kvůli tomu nemůže být Java aplikace informovaná o úspěšnosti ukončení spojovacího algoritmu a uživatel je informován pouze o začátku a konci spojování.

Kapitola 5

Výsledky

Cílem této kapitoly je otestování vzniklé aplikace. Jsou zde blíže upřesněny konkrétní zařízení, na kterých byla aplikace testovaná. Dále je zde popsán výběr testovaných snímků a vyhodnocení algoritmu spojování.

5.1 Testovací zařízení

Ze začátku vývoje probíhalo testování aplikace na virtuálním zařízení z balíčku ADT. Bylo ale nutné přistoupit k testování na fyzických zařízeních. K testům byly použity mobilní telefony LG Nexus 5 a Sony Ericsson Xperia mini. Specifikace těchto mobilních telefonů je uvedena v tabulce 5.1.

zařízení	frekvence procesoru	paměť RAM	API	rozlišení fotoaparátu
SE Xperia mini	1 GHz	512 MB	14	5 MP
LG Nexus 5	2,3 GHz	2 GB	21	8MP

Tabulka 5.1: Specifikace použitých testovacích zařízení.

5.2 Doba výpočtu

Časově nejnáročnější operace celé aplikace je vyhledávání deskriptorů v algoritmu spojování. Tato doba zásadně ovlivňuje dobu výpočtu výsledného panoramatu. Závislost mezi rozlišením snímku, který je zobrazen na obrázku 5.1, a časem stráveným k získání deskriptorů je uvedena v tabulce 5.2. Testování proběhlo na zařízeních, které jsou blíže specifikované v kapitole 5.

Na zařízení SE Xperia mini při vyhledávání deskriptorů pro snímky v rozlišení vyšším než 2800×2100 byla aplikace náhle ukončena bez žádných chybových hlášení. Odhaduji, že toto ukončení bylo provedeno operačním systémem, z důvodu příliš velkých hardwarových požadavků.

Dobu výpočtu výsledného panoramatu lze přibližně určit ze zmíněných průměrných hodnot tabulky 5.2. Vypočtená doba je maximální možná, za kterou se spojí všechny snímky kolekce. Platí následující vzorec:

$$t = n * t_{foto} + (n - 1) * t_{result},$$

	640 × 480	1200 × 900	1600 × 1200	2800 × 2100	4000 × 3000
SE Xperia mini	0:41	2:38	5:05	–	–
LG Nexus 5	0:22	1:10	2:04	6:56	15:23

Tabulka 5.2: Doba v minutách strávená výpočtem deskriptorů jednoho snímku. Experimentálně určené z průměru deseti snímků.

kde t určuje celkovou dobu výpočtu výsledného panoramatu, n udává počet fotografií v kolekci, t_{foto} určuje čas výpočtu deskriptorů pro rozlišení fotografií a t_{result} určuje čas výpočtu deskriptorů pro rozlišení výsledného obrazu.

Obrázek 5.1: Snímek použitý k testování doby získání deskriptorů. Převzato z [8].

5.3 Testovací snímky

Bohužel se nepodařilo sehnat žádný létající aparát, který by vynesl mobilní telefon do dostatečné výšky, aby pořídil kolekci fotografií. Proto byly, se souhlasem vedoucího, jako testovací snímky použité fotografie vytvořené jako ukázková data pro OpenDroneMap[1]. Tyto fotografie jsou pořízené na různých místech, z různých zdrojů (dron, balón, ...). Jsou volně ke stažení ze stránek projektu na serveru GitHub [8].

Ne všechny stažené snímky jsou vhodné pro účely testování, protože se zde nachází i kolekce fotografií, které nebyly pořízené z létajícího aparátu. Proto z fotografií byly vybrány skupiny snímků, které byly pořízené z výšky, v rámci jedné lokality a v přibližně stejné vzdálenosti od země. Vybraným fotografiím bylo o polovinu sníženo rozlišení (přibližně na 2000 × 1500) pro rychlejší zpracování SURF deskriptorem. Vybrané testovací kolekce fotografií byly uloženy do mobilních telefonů a byl spuštěn algoritmus spojování.

5.4 Testovací kolekce

Pro testování spojování byly použity kolekce fotografií Langley, Boruszyn a Caliterra. Z každé kolekce kolekce bylo vybráno prvních 30 snímků, kromě kolekce Boruszyn, u které

[1]OpenDroneMap je open-source nástroj pro zpracování fotografií pořízených pomocí dronu

jsem vybral 20 snímků. Testování proběhlo na obou zařízeních. Výsledky jsou zobrazeny na následujících obrázcích:

Obrázek 5.2: Výsledek spojování kolekce Langley na zařízení Nexus 5 v rozlišení 4000×3000. Doba spojování: 5h 11min. Převzato z [8] a upraveno programem.

Obrázek 5.3: Výsledek spojování kolekce Langley na zařízení SE Xperia mini v rozlišení 1600×1200. Doba spojování: 4h 02min. Převzato z [8] a upraveno programem.

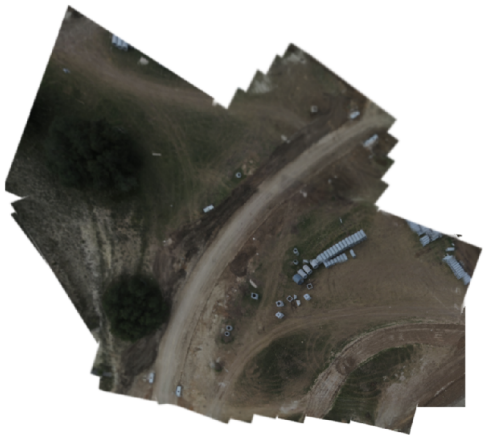
Jak bylo popsáno v části 4.3.10, úspěšné spojení panoramatu závisí na kvalitě prvního snímku. Proto byly testovány i různé varianty pro první snímek. Více vytvořených panoramat i různých zdrojových kolekcí je uloženo na přiloženém CD (viz příloha A).

5.5 Nepodařená spojení

I přes eliminaci špatných spojení dvou fotografií popsané v části 4.3.9 se stále vyskytují problémové snímky. Tato chybná spojení jsou zapříčiněná špatným určením matice homografie či chybným vyhledáním podobných deskriptorů. Ukázka chybných panoramat je na obrázku 5.8.

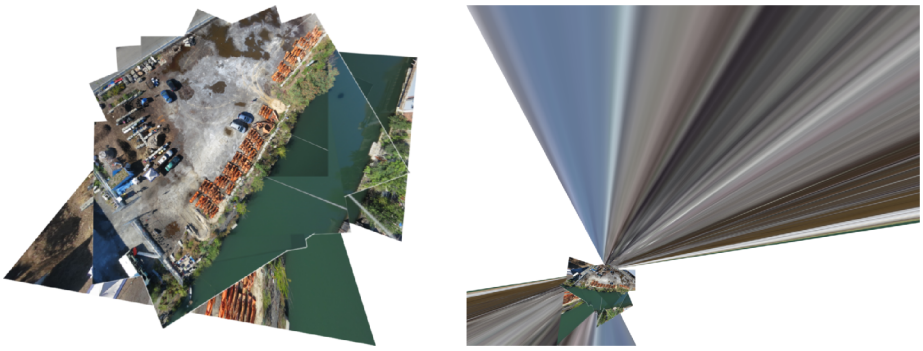
Obrázek 5.4: Výsledek spojování kolekce Boruszyn na zařízení Nexus 5 v rozlišení 4000 × 3000. Doba spojování: 2h 32min. Převzato z [8] a upraveno programem.

Obrázek 5.5: Výsledek spojování kolekce Boruszyn na zařízení SE Xperia mini v rozlišení 1200 × 900. Převzato z [8] a upraveno programem.



Obrázek 5.6: Výsledek spojování kolekce Caliterra na zařízení Nexus 5 v rozlišení 1600 × 1200. Doba spojování: 1h 20min. Převzato z [8] a upraveno programem.

Obrázek 5.7: Výsledek spojování kolekce Caliterra na zařízení SE Xperia mini v rozlišení 1600 × 1200. Doba spojování: 2h 48min. Převzato z [8] a upraveno programem.



Obrázek 5.8: Ukázka nevalidního spojení snímků. Vlevo je chyba zapříčiněná špatným vyhledáním podobných deskriptorů, vpravo chybně určenou maticí homografie.

Kapitola 6

Závěr

V této práci jsou podrobně popsané algoritmy používané pro spojování fotografií do panoramatu. Důležitým prvkem spojení dvou fotografií je dobré určení deskriptorů každého snímku, což je prováděné metodou SURF. Ze získaných deskriptorů se efektivní metodou FLANN vyhledávají mezisnímkové korespondence, z kterých jsou pomocí RANSAC algoritmu eliminovány špatné výsledky pro optimální určení matice homografie. Aplikováním této matice získáme složený snímek dvou na sebe navazujících fotografií. Dále je tu popsán problém ortorektifikace, procesu pro získání planimetricky správného obrazu.

Dalším bodem práce je navržení a implementování aplikace pro operační systém Android. Tato aplikace, i přes počáteční neúspěchy, nakonec zdárně provádí zachycování snímků z kamery. Tyto snímky následně aplikace zpracovává navrženým algoritmem spojování. Aplikace je časově náročná, na čemž se nejvíce podílí metoda SURF pro získání deskriptorů.

Součástí práce je také otestování aplikace na mobilním zařízení. To prokázalo občasné chyby, které byly zapříčiněné nesprávně určenou maticí homografie u zpracovávaných dvojicích fotografií. Tím vznikl obraz sloučený ze dvou fotografií, které na sebe nenavazovaly. Tím výsledné panorama ztrácelo na kvalitě.

Výsledkem této bakalářské práce je stabilní aplikace, která je schopná, v rámci hardwarových možností zařízení, spojit kolekci fotografií v panorama.

6.1 Rozšíření aplikace

Tato závěrečná sekce uvádí další směry, kterými může vést další vývoj a vylepšování aplikace.

Vyhledávání deskriptorů

Časově nejnáročnější část této aplikace je vyhledávání deskriptorů pomocí SURF deskriptoru, který má na druhou stranu velmi kvalitní výsledky. Snížení časové náročnosti by šlo dosáhnout implementováním jiného deskriptoru. Byl by potřeba najít deskriptor, který bude rychlejší, ale jeho výsledky budou také dostatečně kvalitní.

Úprava parametrů

V některých kolekcích fotografií spojovací algoritmus spojí dva snímky homografií, která není vhodná pro spojení daných snímků. Toto špatné spojení lze eliminovat správným

nastavením omezujících parametrů, jež jsou popsány v sekci 4.3.9. Tyto parametry byly určené na základě experimentování s dvěma kolekcemi fotografií. Je možné, že v případě dalšího, důkladnějšího experimentování s vícero kolekcemi by bylo možné nalézt vhodnější parametry.

Důležitost první fotografie

V sekci 4.3.10 zabývající se výběrem dvojic snímků je popsána důležitost prvního snímku kolekce, od které se odvíjí úspěšnost výsledného panoramatu. Tím vzniká možné riziko v případě nekvalitního prvního snímku. Tento problém by šel řešit dvěma způsoby. Základní snímek, od kterého by se odvíjelo spojování, by vybral uživatel. Tímto krokem ale nebude vznik panoramatu zcela automatický. Druhý způsob řešení je zvolit druhou fotografii jako základní snímek v případě neúspěchu spojování prvního snímku. Pokud by ale nastala situace, že by nešel spojit žádný snímek z kolekce, byl by tento algoritmus určování dvojic snímků velmi neefektivní.

Váha viditelnosti

Určení váhy viditelnosti, jež je vysvětlené v sekci 4.3.7, není vyřešeno efektivně. Pro každý bod přidávaného snímku se prochází celé jeho čtvercového okolí pro získání počtu viditelných pixelů. Z důvodu úspory doby potřebné pro vytvoření panoramatu se prolínání nepoužívá při vytváření mezivýsledků. Pro zlepšení efektivity by se dala použít modifikace integrálního obrazu (viz 2.3), který provádí součet hodnot pixelů. V tomto případě by se nesčítaly hodnoty pixelů, ale počet viditelných pixelů. Součet viditelných pixelů okolí bodu, pro než se určuje váha viditelnosti, by byl následně určen pouhým součtem čtyř hodnot.

Literatura

[1] ZÁKLADY TVORBY UŽIVATELSKÉHO ROZHRANÍ. [online], 2007 [cit. 2015-05-18].
URL http://phoenix.inf.upol.cz/esf/ucebni/gui-dostal.pdf

[2] Java Native Interface Specification. [online], 2012-11-12 [cit. 2015-05-07].
URL http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html

[3] Orthophoto. [online], 2014-12-13 [cit. 2015-05-03].
URL http://en.wikipedia.org/wiki/Orthophoto

[4] Feature Detection and Description — OpenCV 2.4.11.0 documentation. [online], 2015-02-25 [cit. 2015-02-14].
URL http://docs.opencv.org/modules/features2d/doc/feature_detection_and_description.html

[5] Feature Detection and Description — OpenCV 2.4.11.0 documentation. [online], 2015-02-25 [cit. 2015-02-14].
URL http://docs.opencv.org/modules/nonfree/doc/feature_detection.html

[6] OpenCV4Android SDK – OpenCV 2.4.11.0 documentation. [online], 2015-02-25 [cit. 2015-05-07].
URL http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html

[7] Feature Matching with FLANN — OpenCV 2.4.11.0 documentation. [online], 2015-02-25 [cit. 2015-05-14].
URL http://docs.opencv.org/doc/tutorials/features2d/feature_flann_matcher/feature_flann_matcher.html

[8] OpenDroneMap. [online], 2015-2-21 [cit. 2015-05-16].
URL https://github.com/OpenDroneMap/odm_data

[9] Camera — Android Developers. [online], 2015-5-13 [cit. 2015-05-15].
URL http://developer.android.com/reference/android/hardware/Camera.html

[10] Activity — Android Developers. [online], 2015 [cit. 2015-04-28].
URL http://developer.android.com/reference/android/app/Activity.html

[11] Dashboards — Android Developers. [online], 2015 [cit. 2015-04-28].
URL http://developer.android.com/about/dashboards/index.html

[12] OpenCV — OpenCV. [online], 2015 [cit. 2015-05-07].
URL http://opencv.org/

[13] The Developer's Guide — Android Developers. [online], [cit. 2015-05-07].
URL http://www.kandroid.org/ndk/docs/ANDROID-MK.html

[14] Bay, H.; Tuytelaars, T.; Van Gool, L.: Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, Springer, 2006, s. 404–417.

[15] Benda, M.: Homografie a epipolární geometrie. [online], 2010-10-31 [cit. 2015-04-05].
URL http://trilobit.fai.utb.cz/homografie-a-epipolarni-geometrie

[16] Derpanis, K. G.: Overview of the RANSAC Algorithm. *York University*, ročník 68, 2010.

[17] Domtheos: Orthorectification – OSSIM. [online], 2014-07-02 [cit. 2015-03-30].
URL http://www.encyclopediaofmath.org/index.php?title=Edge_detection

[18] Domtheos: FLANN - Fast Library for Approximate Nearest Neighbors. [online], 2014-12-14 [cit. 2015-05-05].
URL http://www.cs.ubc.ca/research/flann/

[19] Fischler, M. A.; Bolles, R. C.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*, ročník 24, č. 6, 1981: s. 381–395, ISSN 0001-0782.
URL http://doi.acm.org/10.1145/358669.358692

[20] Gargenta, M.: *Learning Android*. O'Reilly Media, Inc., 2011, ISBN 978-1-449-39050-1.

[21] Hanzl, V.: Fotogrammetrie – Teoretické základy fotogrammetrie. [online], 2006 [cit. 2015-03-30].
URL http://fast.darmy.net/opory%20-%20III%20Bc/GE15-Fotogrammetrie_I--M01-Teoreticke_zaklady_fotogrammetrie.pdf

[22] Hartley, R.; Zisserman, A.: *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[23] Hast, A.; Nysjö, J.; Marchetti, A.: Optimal RANSAC – towards a repeatable algorithm for finding the optimal set. 2013.

[24] Iwamura, M.; Sato, T.; Kise, K.: What is the Most EfficientWay to Select Nearest Neighbor Candidates for Fast Approximate Nearest Neighbor Search? In *Computer Vision (ICCV), 2013 IEEE International Conference on*, IEEE, 2013, s. 3535–3542.

[25] Juan, L.; Gwun, O.: SURF applied in panorama image stitching. In *Image Processing Theory Tools and Applications (IPTA), 2010 2nd International Conference on*, IEEE, 2010, s. 495–499.

[26] Lee, S.; Jeon, J. W.: Evaluating performance of Android platform using native C for embedded systems. In *Control Automation and Systems (ICCAS), 2010 International Conference on*, IEEE, 2010, s. 1160–1163.

[27] Lindeberg, T.: Edge detection - Encyclopedia of Mathematics. [online], 2011-02-07 [cit. 2015-03-15].
URL http://trac.osgeo.org/ossim/wiki/orthorectification

[28] Marr, D.; Hildreth, E.: Theory of Edge Detection. *Proceedings of the Royal Society of London B: Biological Sciences*, ročník 207, č. 1167, 1980: s. 187–217, ISSN 0080-4649, doi:10.1098/rspb.1980.0020.

[29] Schaeffer, C.: A Comparison of Keypoint Descriptors in the Context of Pedestrian Detection: FREAK vs. SURF vs. BRISK. 2013.

[30] Sonka, M.; Hlavac, V.; Boyle, R.: *Image Processiong, Analysis, and Machine Vision*. Thomson, 2008, ISBN 0-495-08252-X.

[31] Titov, V.: The Developer's Guide — Android Developers. [online], 2013-12-16 [cit. 2015-05-07].
URL http://elekslabs.com/2013/12/introduction-into-android-ndk.html

[32] Tuytelaars, T.; Mikolajczyk, K.: Local invariant feature detectors: a survey. *Foundations and Trends® in Computer Graphics and Vision*, ročník 3, č. 3, 2008: s. 177–280.

[33] Ujbnyai, M.: *Programujeme pro Android*. Grada Publishing, a.s., 2012, ISBN 978-80-247-3995-3.

[34] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-454-0X.

Příloha A

Obsah CD

- `CD-ROM`
  - `Bakalářská práce` ..... zdrojový soubor LaTeX, vygenerované PDF
  - `Kolekce snímků` ..... kolekce snímků použité pro testování
  - `Výstup aplikace` ..... ukázky vzniklých panoramat
  - `Zdrojové kódy` ..... vytvořené zdrojové kódy
    - `Aplikace` ..... zdrojový kód Android aplikace
    - `C++ algoritmus` ..... zdrojový kód konzolové aplikace
  - `Zkompilovaný program` ..... konzolová aplikace, instalátor APK pro Android