FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Věnná města českých královen – Lokalizační modul
Student:	Lukáš Melcher
Vedoucí:	Ing. Petr Pauš, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Projekt Věnná města českých královen se zaměřuje na vývoj programů a aplikací sloužící pro přiblížení historických budov a prvků veřejnosti za využití prostředků virtuální i rozšířené reality. Pro využití v mobilní aplikaci je třeba určit nejpřesněji polohu uživatele v prostoru.

1. Analyzujte stávající řešení lokalizačního modulu.
2. Analyzujte možnosti strojového zpracování obrazu za účelem nalezení korelace mezi vstupními obrazovými daty.
3. Analyzujte možnosti určení polohy na základě zpracování obrazu.
4. Navrhněte řešení lokalizačního modulu dle specifikací daných v předchozích pracích.
5. Implementujte jedno z navržených řešení.
6. Implementované řešení podrobte testům a diskutujte odchylku od očekávané polohy.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Věnná města českých královen – Lokalizační modul

Lukáš Melcher

Katedra softwarového inženýrství
Vedoucí práce: Ing. Petr Pauš, Ph.D.

26. května 2020

Poděkování

Rád bych poděkoval vedoucímu své práce panu Ing. Petru Paušovi, Ph.D. za vedení, rady a pomoc při psaní této bakalářské práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 26. května 2020 . . . . . .

Odkaz na tuto práci

Melcher, Lukáš. *Věnná města českých královen – Lokalizační modul.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce rozebírá možnosti využití metod strojového zpracování obrazu pro určení pozice pomocí knihovny OpenCV. Diskutuje rovněž využitelnost metody pro účely virtuální a rozšířené reality.

Na základě rozpoznání známých objektů a výpočtu vzdálenosti od nich využívá metod analytické geometrie k získání pozice pozorovatele. Díky tomu umožňuje zlepšení uživatelského zážitku při používání aplikací pro virtuální a rozšířenou realitu z důvodu zpřesnění dat získaných ze senzorů zařízení.

Klíčová slova lokalizace, rozpoznání obrazu, rozšířená realita, virtuální realita, analytická geometrie, počítačové vidění, měření vzdálenosti, OpenCV, GPS

Abstract

The thesis describes options of using machine processing of image data for estimation or measuring of the position based on the OpenCV library. Also, it discusses if the method is useful for purposes of virtual and augmented reality.

Based on the recognition of known objects and the calculation of the distance from them, it uses the methods of analytical geometry to obtain the position of an observer. As a result, it enables the refinement of data obtained from device sensors and thus improves the user experience when using applications for virtual and augmented reality.

Keywords localization, image recognition, augmented reality, virtual reality, analytic geometry, computer vision, distance measuring, OpenCV, GPS

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Úvod

Platforma Věnná města českých královen je projekt, který si klade za cíl seznámit (nejen) širší veřejnost s minulostí historický měst. Byť nikdo neumí cestovat časem, aby si mohl prohlédnout historické skvosty středoevropské architektury přímo, přesto, díky tomuto projektu, získají uživatelé možnost nahlédnout do historie příjemným způsobem. Touto možností je využití rozšířené reality v mobilních zařízeních s operačním systémem Android.

Virtuální a rozšířená realita jsou bezpochyby velmi zajímavé obory s velkou perspektivou pro příští léta i desetiletí [2]. Tento projekt se zaměřuje primárně na realitu rozšířenou. Dle volného překladu definice, jež uvádí Encyclopaedia Britannica [3], jde o ‚proces úpravy videa (...), které je částečně překrýváno počítačem generovanými užitečnými prvky' [1] . Definice obsahuje jedno velmi důležité slovo – překrytí. Pro maximalizaci uživatelského zážitku a věrohodnost zobrazení je přesnost překrytí klíčová. Abychom jí dosáhli, je nicméně třeba získat některé informace. Mezi ně patří úhel pozorování objektu, jeho velikost, poloha objektu a pozorovatele a další aspekty. Mnoho z nich je řešitelných pomocí senzorů mobilních zařízení, jako je například GPS, GLONASS, Galileo a další systémy určení zeměpisných souřadnic nebo elektronický kompas pro určování úhlu.

Problémem ovšem zůstává nepříliš vysoká přesnost údajů [4], jež tyto senzory poskytují, pro účely rozšířené reality. Tato práce vzniká za účelem analyzovat a vyzkoušet možnosti strojového zpracování obrazu pro lokalizaci mobilního zařízení.

Kapitola 2. uvádí teoretické pozadí práce. Ve 3. kapitole čtenář nalezne shrnutí procesu implementace včetně některých důležitých učiněných rozhodnutí. Nakonec ve 4. kapitole jsou ukázky z testování výsledné aplikace a rovněž diskuze získaných výsledků.

[1] *Augmented reality, in computer programming, a process of combining or "augmenting" video or photographic displays by overlaying the images with useful computer-generated data.* Vlastní překlad.

Kapitola 1

Cíl práce

Cílů této práce je více. Vzhledem ke značné rozsáhlosti platformy Věnná města českých královen proběhne analýza již implementované či naplánované části projektu a budou vyhodnoceny vhodné způsoby implementace lokalizačního modulu, aby bylo následně možné nově vytvořenou funkcionalitu integrovat. Samotnou integrací modulu do projektu se však tato práce nezabývá.

Rovněž bude uvedena analýza možností strojového rozpoznávání obrazu na základě knihovny OpenCV a budou vybrány z její funkcionality části, jež budou v implementační části využity.

Práce rozebere způsoby využití rozpoznávání obrazu k odhadu, popřípadě výpočtu polohy. Tyto metody budou porovnány a bude diskutována jejich vhodnost.

Po proběhnutí analytické části bude na jejím základě navržen způsob vypočtení polohy, který bude také implementován. Řešení bude podrobeno testům funkčnosti a případná odchylka od referenčních hodnot bude diskutována.

Kapitola 2

Analýza a návrh

Tato kapitola rozebírá teoretické a rešeršní pozadí této bakalářské práce a jeho návaznost na již implementovanou část projektu i na teprve zamýšlenou část implementace.

2.1 Současné řešení problému

Vzhledem k rozsáhlosti platformy Věnná města byl nejprve proveden průzkum hotové části.

2.1.1 Využité technologie

Ing. Jaroslav Štěpán ve své diplomové práci [5], která se zabývá také tímto projektem, již navrhl způsob, jak integrovat modul s požadovanou funkcionalitou – tedy modul lokalizační.

Jeho návrh počítal s implementací v jazyce C++ s využitím Java Native Interface (JNI) a Native Development Kit (NDK) primárně z důvodu vyššího výkonu takového kódu. Aplikace dále obsahuje části kódu v jazycích Java a Kotlin. V případě této dvojice je velkou výhodou kompatibilita tohoto kódu, který se dá kombinovat bez větších problémů. Naopak zmíněný kód v C++ se musí volat skrze definované rozhraní [5].

Aplikace využívá možností, jež skýtá opensource knihovna OpenCV [6]. Tato, v jazyce C++, napsaná knihovna nabízí široké spektrum funkcí a metod pro strojové zpracování obrazu. Mimo jiné jde o tyto:

Klíčové body – struktura, jež v množině popisuje obrazová data pomocí matematických prostředků.

Matice – způsob, jakým knihovna ukládá a spravuje obrazová data.

Deskriptory – struktura, jež popisuje okolí klíčových bodů a jeho odlišnost od bodů dalších.

Algoritmy – pro tuto práci jsou zajímavé primárně algoritmy pracující s klíčovými body. Jde zejména o algoritmy:

1. pro výpočet klíčových bodů,
2. pro párování klíčových bodů na větším počtu obrazových dat.

2.1.2 Princip zjištění polohy

Původní aplikace využívala podobný princip jako člověk. Podle uložené databáze se pokusila určit nejpodobnější obrázek, u nějž měla uloženou polohu, a následně vrátila tuto polohu jako výstup. Tedy stejně, jako by se člověk rozhlédl po náměstí, všiml si orloje a uvědomil si, že tedy asi bude na Staroměstské náměstí v Praze (případně na olomouckém Horním náměstí, pokud by byl orloj modernější).

K této funkcionalitě byla implementována třída `ImageRepository`, jež obsahovala obrázky již známých míst a dokázala mezi nimi nalézt obrázek nejpodobnější, či upozornit na fakt, že velmi pravděpodobně není podobný obrázek k dispozici.

Tento systém pracoval na nepříliš složitém principu. Pro každý nově přidaný obrázek do `ImageRepository` byly spočítány deskriptory obrázku a dopočítány klíčové body. Tento proces sice obsahuje velmi náročné matematické metody, ty jsou ale zapouzdřené ve využité knihovně. K samotnému výpočtu je možné využít více různých algoritmů. Mimo jiné jsou to algoritmy SURF a SIFT, které jsou však chráněny patenty [7], [8]. Byla dána přednost testům s algoritmy ORB, BRISK a AKAZE [9], jež jsou distribuovány volně. Mimo jiné se prací s těmito algoritmy a jejich popisem zabývá Ing. Jaroslav Štěpán v již odkazované diplomové práci [5].

Pro nalezení podobného obrázku byl dodán obrázek vstupní, pro nějž byly rovněž spočítány deskriptory a klíčové body. Pomocí párovacího algoritmu byly následně spojeny body na jednotlivých obrázcích databáze a na obrázku vstupním. Jako nejpodobnější byl vybrán obrázek, jež měl nejlepší poměr mezi počtem spárovaných klíčových bodů a celkovým počtem klíčových bodů na uloženém obrázku.

Pro každý uložený obrázek byla uložena odpovídající poloha ve formě zeměpisné souřadnice. Tato souřadnice byla následně vrácena.

2.2 Rozdělení úlohy

Hned na počátku práce padlo rozhodnutí o rozdělení úlohy na několik menších částí, jež budou propojeny pomocí rozhraní. Díky tomu bude možné jednotlivé části snáze samostatně testovat, postupně vyvíjet a bude snazší udržet hranici mezi různými částmi funkcionality. V kapitolách 2.3, 2.4 a 2.5 jsou rozebrány jednotlivé celky.

2.3 Rozpoznání objektů

Samotné rozpoznávání objektů je velmi náročnou výpočetní úlohou [10]. Tato práce se přesným rozpoznáváním konkrétního objektu nebude zabývat. Využije ale již implementovanou funkcionalitu, aby bylo možné nalézt co nejpodobnější obrázek ku vstupním datům v paměti aplikace.

Obecně lze říci, že než hledat konkrétní objekty, bude aplikace hledat shodnost konkrétní množiny klíčových bodů. Shodnost samotnou samozřejmě nic zaručit nemůže, jde spíše o spárování bodů na základě vzdáleností mezi nimi [5].

Bylo zvažováno více možností porovnávání obrázků. Zde uvádím dvě metody označené jako M_1 *a* M_2.

$$M_1 = \frac{\text{počet spárovaných klíčových bodů}}{\text{počet klíčových bodů vstupního obrázku}}$$

či

$$M_2 = \frac{\text{počet spárovaných klíčových bodů}}{\text{počet klíčových bodů uloženého obrázku}}.$$

Obě tyto možnosti mají své výhody. Zpravidla lze říci, že je lepší dělit počet spárovaných bodů počtem klíčových bodů toho obrázku, jenž zachycuje pouze objekt, jenž je na obrázku významný (tj. budova, socha, specifické okno atd.) a který je specifický pro dané místo.

Protože ale nebylo na počátku projektu jasné, který z těchto poměrů bude výhodnější, bylo navrženo rozhraní, se kterým pracují ostatní části aplikace a na jehož místo je možné dosadit různé třídy, jež budou implementovat jednotlivé možnosti porovnávání.

Obrázek 2.1: Ukázka spočítaných klíčových bodů.

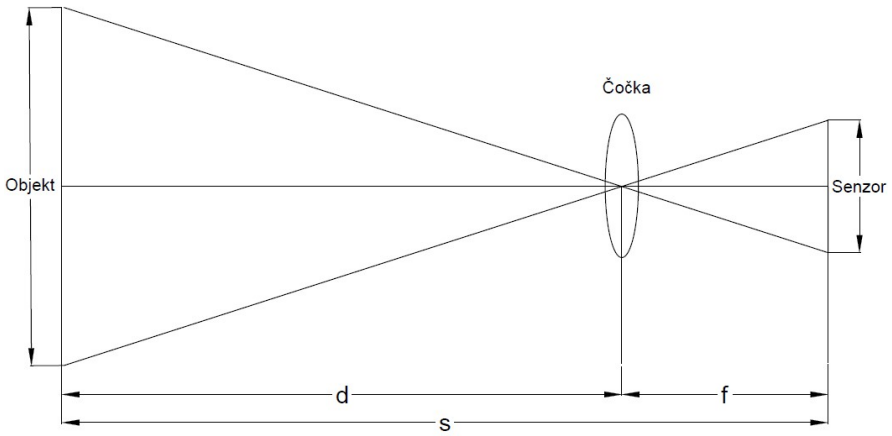
2.4 Výpočet vzdálenosti

Prvním bodem tedy je získat z databáze co nejpodobnější obrázek. Následně je třeba dopočítat, nebo alespoň odhadnout vzdálenost fotoaparátu od objektu.

Dle [11] je možné použít tento vzorec:

$$d = \frac{f h_{sk} h_f}{h_o h_s},$$

kde:
f – ohnisková vzdálenost čočky fotoaparátu [mm],
h_{sk} – skutečná výška objektu [mm],
h_f – výška fotografie [pix],
h_o – výška objektu na fotografii [pix],
h_s – výška senzoru fotoaparátu [mm].



Obrázek 2.2: Schéma měření vzdálenosti.

Z tohoto vzorce vyplývá, že pro zisk vzdálenosti je třeba získat tabulkovou hodnotu velikosti senzoru a čočky. Rovněž je nutné mít k dispozici výšku detekovaného, známého objektu na fotografii.

Velmi důležitým bodem je, že se všemi hodnotami bude nutné pracovat jako s výškami. Tedy i pokud bude potřebné pracovat s vektory, bude nezbytné pracovat pouze s y souřadnicemi.

Důvod k tomuto postupu je takový, že zkreslení způsobené různými pohledy ze stran a krajních úhlů může být velmi velké a mohlo by způsobovat velké nepřesnosti v měření.

Hrozí, že tento problém může nastat i při používání pouze výšek (zmíněných y souřadnic), v případech, kdy dojde k velkému zkreslení vlivem čočky. Jednoduše řečeno, čím více budou na fotce normálně přímé linie ohnuté do oblouků, tím méně se můžeme spolehnout na linearitu plynoucí z výše uvedeného vzorce a tím nepřesnější výsledky lze očekávat.

Hodí se připsat několik poznámek.

- Pokud je o čočce fotoaparátu mobilního telefonu známo, že její ohnisková vzdálenost je kupříkladu 29 milimetrů, stojí za to pečlivě zkontrolovat zdroj, odkud tato hodnota pochází. Zpravidla se totiž uvádí hodnota ekvivalentu 35 milimetrového filmu. Tudíž pokud je zmíněných 29 milimetrů dosazeno do vzorce, vyjdou nesmyslná data. Daleko důvěryhodnější hodnoty ohniskové vzdálenosti čoček jsou v řádech jednotek milimetrů.

- Výpočet velikosti objektu na fotografii je sice pro člověka triviálním úkolem, ale při využití strojového zpracování obrazu rozhodně nejde o jednoduchou úlohu. Pokud totiž nemáme hranici, odkud až kam objekt sahá, ale máme pouze množinu bodů, je nutné určit, se kterými body a jak budeme pracovat.

- Další poznámka souvisí s předchozím bodem. Jestliže je problémem určit, které body budou určovat objekt, pak je samozřejmě problém i určit výšku takového objektu. Příkladem může být situace, kdy je ve vstupních datech například fotografie Orloje na Staroměstském náměstí v Praze, kterému ale budou chybět slavná okénka s apoštoly. Informace, jak vysoký je celý orloj, je sice známá, ale bude nutné dopočítat, jak vysoká je pouze jeho část – část, která je na vstupu.

- Vyhledat velikost senzoru na internetu se ukázalo jako poměrně náročná úloha. Pokud je vůbec dohledatelná, a to spíše u vlajkových lodí větších firem, jedná se pravděpodobně o velikost úhlopříčky, než o její výšku. Je otázkou k zamyšlení (a není to součástí této práce), zda je možné využít poměru stran fotografie k dopočítání délky konkrétních stran. V oddíle realizace je uveden alternativní postup získání odpovídajících dat (konkrétně sekce 3.4).

2.5 Výpočet polohy

K vypočítání polohy je z částí aplikace, popsaných v předchozích kapitolách, k dispozici rozpoznaný obrázek a k němu vzdálenost. Z toho důvodu, pokud je uložena do aplikace informace o obrázku – například zeměpisná souřadnice budovy na něm – je možné získat kulovou plochu či kružnici určenou zeměpisnou souřadnicí jako středem a vzdáleností jako poloměrem.

V optimálním případě by tedy mělo být možné ze tří rozpoznaných objektů dostat souřadnici v prostoru, jenž by měla udávat polohu pozorovatele. Podstatným faktem ale zůstává, že důležitá je především poloha na zemi a ne výška, ze které byla fotografie pořízena. Tedy, postačuje průmět průniku dvou a více kulových ploch na kulovou plochu, nebo ještě lépe, sféroid či geoid, který by znázorňoval planetu Zemi.

Vzhledem k vysoké náročnosti na implementaci, přesnost analytických výpočtů a také vzhledem k problémům práce se zeměpisnými souřadnicemi se tato práce omezuje na výpočet planimetrický.

Jednoduše řečeno, po zjištění vzdáleností od daných zeměpisných souřadnic budou souřadnice převedeny do rovinné kartézské soustavy souřadnic, v níž bude následně pomocí nástrojů analytické geometrie vypočítána poloha.

2.5.1 Metody převodu zeměpisných souřadnic do souřadnic kartézských

Tato úloha se ukázala jako daleko náročnější, než se zdála zpočátku. Celé úskalí tkví v neexistenci univerzálního přesného převodu. Podobně jako dochází ke zkreslení u map, dochází k němu zde. Prostě nelze jednoduchým způsobem převést povrch planety do rovinné reprezentace, a přitom se nesetkat se zkreslením.

Různých projekcí a typů převodů zeměpisných souřadnic do rovinné reprezentace je velké množství. Tato práce tedy obsahuje rozhraní, které je využito v dalších komponentách a tedy je snadno rozšířitelná o další projekce.

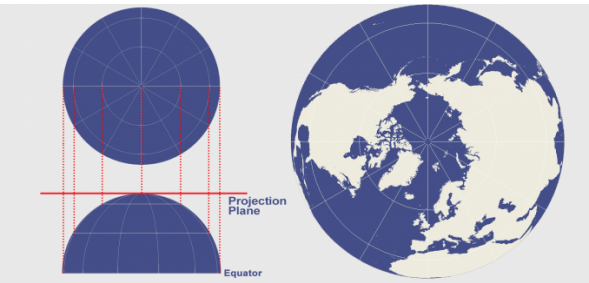
V seznamu níže jsou některé z nich. Jejich popis je velmi neformální. Než budou krátce shrnuty, je třeba stanovit cíle jejich použití a nároky na ně kladené. Například GPS může mít odchylku až v řádu metrů. Požadavkem na projekci tedy bude, aby měla odchylky menší. Dalším požadavkem bude snadná implementace a přijatelná složitost.

Pro testování a odhady spolehlivosti byl vybrán vzorec Haversine [12], který určuje vzdálenost dvou bodů na kouli v závislosti na jejich zeměpisné šířce a délce. Tomuto vzorci se věnuje kapitola 2.5.2. Snahou této práce byla přesnost převodu srovnatelná s přesností vzorce Haversine. Je nezbytné zdůraznit, že předpoklad podobné přesnosti nebude platit u větších vzdáleností. Vzhledem k možnostem rozpoznání obrazu a výběru správného

známého prvku není třeba počítat s většími oblastmi pro zobrazení, než stovkami metrů. Proto padlo rozhodnutí použít tento vzorec.

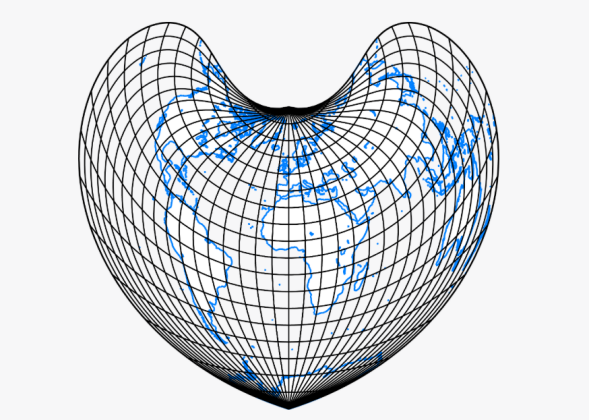
Dle [13], [14] byly zvažovány tyto druhy projekcí (obrázky z [15], [16], [17] a [18]):

Azimutální projekce [19]: Povrch zemského povrchu se promítá na povrch plochy, jež se povrchu dotýká v jednom konkrétním bodě. Zachovává azimuty – odtud jeho jméno. Pro příklad se využívá Lambertovo, Ortografické (viz obrázek 2.3) či stereografické zobrazení.



Obrázek 2.3: Azimutální projekce.

Bonneho projekce [20]: Uvádím spíše pro zajímavost. Zobrazuje povrch Země na těleso vzniklé rotací tvaru srdce okolo svislé osy souměrnosti (viz obrázek 2.4).

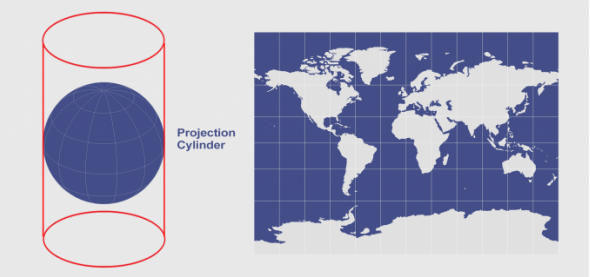


Obrázek 2.4: Bonneho projekce.

Kuželová projekce [21]: Při použití této projekce dochází k zobrazení zemského povrchu na plášť kužele (viz obrázek 2.5). Kružnice dotyku či výška kužele závisí na použití a nastavení. Jako příklady se uvádí zobrazení Albersovo či Ptolemaiovo.

Obrázek 2.5: Kuželová projekce.

Válcová projekce [22]: Pokud využijeme válcovou projekci, vložíme planetu Zemi do pomyslného válce, na jehož plášť následně zobrazujeme zemský povrch (viz obrázek 2.6). Podle výběru parametru rozlišujeme více typů válcových zobrazení. Například Lambertovo, Behrmannovo, Tristanovo, Petersovo, Balthasartovo atd.



Obrázek 2.6: Válcová projekce.

2.5.2 Vzorec Haversine

Je třeba si povšimnout faktu, že tento vzorec pracuje pouze s výpočty na *kulové ploše*. Vzhledem k tomu, že zemský povrch není ideální koulí, bylo by lepší počítat vzdálenosti na geoidu. Zkreslení, které tím vzniká, je však pro potřeby výpočtu na vzdálenostech o maximálně stovkách metrů zanedbatelné. Dle [23] vypadá samotný vzorec takto:

$$d = 2r \arcsin \sqrt{\sin^2(\frac{\phi_2 - \phi_1}{2}) + \cos(\phi_1)\cos(\phi_2)\sin^2(\frac{\lambda_2 - \lambda_1}{2})},$$

kde d (distance) je vzdálenost dvou bodů, r je poloměr Země a ϕ *a* λ jsou zeměpisná šířka a délka obou bodů v radiánech.

Například pomocí Vincentova vzorce [12] by bylo možné dostat vzdálenost přesněji, protože nevyužívá koule, ale sféroidu. Jak ale bylo zmíněno výše, rozdíl bude zanedbatelný.

Jak bylo uvedeno, vzorec Haversine bude využit pro testování. Vzdálenost mezi dvěma body bude spočítána jeho použitím a následně porovnána s hodnotou, jež bude spočítána pomocí Pythagorovy věty, kde využijeme již body v kartézské soustavě souřadnic dopočítané z vybrané projekce.

2.5.3 Ekvidistantní válcová projekce

Pro převedení zeměpisných souřadnic do souřadnic kartézských byla vybrána ekvidistantní válcová projekce. Ta zobrazuje povrch planety na plášť válce, jde tedy o druh válcové projekce. Vyznačuje se jednoduchým vzorcem pro přepočet na kartézské souřadnice i z nich. Dalším plusem je možnost určit centrální poledník (nemusí být shodný s poledníky zemskými, zobrazení umožňuje transformace podle jiných os, než je ta, okolo které se Země otáčí). Tedy přizpůsobit zobrazení přímo pro oblast, ve které se nachází objekty, a tím zvýšit přesnost.

Tedy:

- x, y – souřadnice bodu v kartézské soustavě,
- R – poloměr Země,
- λ, ϕ – zeměpisné souřadnice bodu,
- λ_0 – zeměpisná délka centrálního poledníku zobrazení (v radiánech),
- ϕ_1 – určuje měřítko věrohodnosti zobrazení.

A výpočet souřadnic dle [24] bude vypadat takto:

- $x = R(\lambda - \lambda_0)\cos\phi_1$,

- $y = R\phi$.

Záměr použití vyžaduje rovněž opačný směr převodu:

- $\phi = \frac{y}{R}$,
- $\lambda = \lambda_0 + \frac{x}{R \cos \phi_1}$.

Je ale třeba zdůraznit, že (a je to z uvedených vzorců jasně patrné) ani souřadnice zadaná jako střed zobrazení nevrátí kartézské souřadnice [0,0]. Proto je s výsledky převodu v této práci pracováno jako s ΔA tj. s rozdílem polohy bodu A oproti souřadnicím středu.

2.5.4 Planimetrický výpočet

Jde o klasickou úlohu v analytické geometrii.

Mějme:

- kružnici k – určenou bodem A a poloměrem r_k,
- kružnici l – určenou bodem B a poloměrem r_l,
- $\overrightarrow{BA}$ – vektor určený body A a B.

Než ovšem začneme počítat průnik kružnic, stojí zato nalézt počet řešení [25]. Ten bude (za předpokladu $A \neq B$):

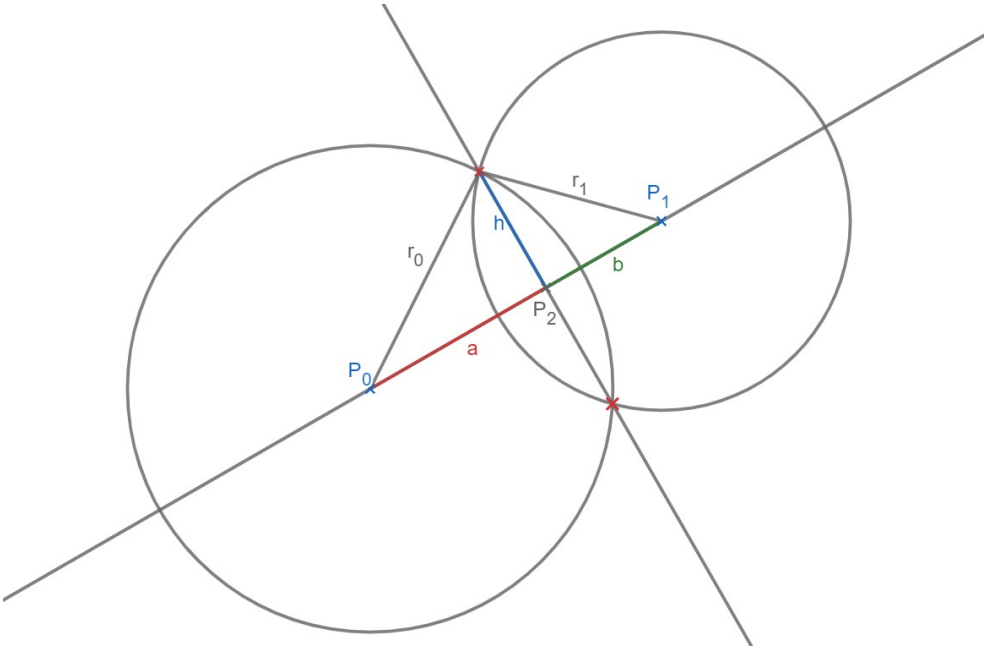
- 0 – v případě, kdy $|\overrightarrow{BA}| > r_k + r_l$,
- 1 – v případě, kdy $|\overrightarrow{BA}| = r_k + r_l$,
- 2 – v případě, kdy $|\overrightarrow{BA}| < r_k + r_l$.

V případě jediného bodu průniku $P_2[x_2, y_2]$, tedy pouze dotyku kružnic využijeme vzorec:

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}, \qquad a = \frac{r_0^2 - r_1^2 + d^2}{2d},$$

$$x_2 = \frac{x_0 + a(x_1 - x_0)}{d}, \qquad y_2 = \frac{x_0 + a(y_1 - y_0)}{d}.$$

A výpočet pro možnost 2 průsečíků $P_2[x_2, y_2]$ *a* $P_3[x_3, y_3]$ (podle pojmenování na obrázku 2.7):



Obrázek 2.7: Výpočet průsečíků kružnic, inspirováno [25].

$$d = a + b, \qquad d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

$$a = \frac{r_0^2 - r_1^2 + d^2}{2d}, \qquad h = \sqrt{(r_0^2 - a^2)},$$

$$x_2 = \frac{x_0 + a(x_1 - x_0)}{d}, \qquad y_2 = \frac{y_0 + a(y_1 - y_0)}{d},$$

$$x_3 = \frac{x_2 + h(y_1 - y_0)}{d}, \qquad y_3 = \frac{y_2 - h(x_1 - x_0)}{d}.$$

Kapitola 3

Realizace

V této kapitole bude rozebrán postup a rozhodnutí učiněná při samotné implementaci aplikace.

V sekci 3.1 je poukázáno na rozdíly vůči práci Ing. Jaroslava Štěpána [5] a budou diskutovány jednotlivé změny. Rovněž budou rozebrány technologie a postupy využité při realizaci úkolu.

V sekci 3.2 je krátký přehled funkčních celků, balíčků a obecně členění aplikace.

Následují sekce 3.3, 3.4 a 3.5, které rozebírají implementace nejdůležitějších funkčních celků.

3.1 Obecný rozbor

Jak bylo rozebráno v kapitole 2.1, původní návrh počítal s využitím velké části kódu v C++ s přístupem přes rozhraní.

Výpočetně nejnáročnější operace však byly a jsou zabezpečeny knihovnou OpenCV, která sice Javu (ať už pro serverové, desktopové nebo android aplikace) podporuje, ale zdaleka (a naštěstí) v Javě napsaná není. Poskytuje však rozhraní, skrz které jsou volané metody v C++ z kódu v Javě.

Vzhledem k několikanásobnému oddělení, nemalé složitosti tohoto postupu a o mnoho nižší čitelnosti takového kódu byl upřednostněn jiný přístup. Celý kód (vyjma kódu OpenCV, kde jsou nároky na výkon enormní) byl napsán v jazyce Java, díky čemuž je možné využít jednodušších konstrukcí. Současně lze využít relativně dobré přenositelnosti tohoto jazyka a lokalizační modul vyvinout jako aplikaci desktopovou, která bude teprve přizpůsobena pro využití v mobilním zařízení (toto přizpůsobení však není obsahem této práce, byť práce v některých případech může naznačovat jednu z možných cest při migraci aplikace).

Nevýhodou tohoto řešení je však nesporné snížení výkonu takového kódu. Přestože operace o nejvyšší výpočetní náročnosti stále probíhají v OpenCV, pokles výkonu může být znatelný [26].

Z toho důvodu byl přepsán kód lokalizačního modulu, který byl již implementován, z jazyka C++ do jazyka Java (o původním řešení pojednává sekce 2.1.2).

Rovněž došlo k migraci od algoritmu BRISK k algoritmu ORB. O důvodech a důsledcích tohoto rozhodnutí blíže pojednává sekce 4.1.1.

3.1.1 Využité technologie

V níže umístěném seznamu jsou sepsány hlavní technologie a knihovny, které byly využity při implementaci včetně krátkého popisu.

OpenCV – Již mnohokrát bylo zmíněno, že k samotnému zpracování obrazových dat je využita knihovna OpenCV. Kapitola 2.1.1 zmiňuje části jež patří v této práci k nejpoužívanější funkcionalitě. Nutno zdůraznit, že verze určená pro desktopovou Javu je upravována s jinou frekvencí a tedy může obsahovat kupříkladu jiné konstrukce příkazů, než verze pro Android.

Návrhové vzory – Nejde přímo o technologii, spíš si je lze představit jako doporučení, či doporučené postupy, jak řešit konkrétní problémy, s nimiž se programátor pravděpodobně ve své praxi potká. Z těchto vzorů byl použit především vzor *Továrna*. Práce zmiňuje možnost využití vzoru *Stavitel*, použit ale není, vzhledem k nepotřebnosti funkcionality v době vývoje a tedy zbytečnému komplikování kódu.

Git – Systém pro verzování kódu (nebo jakéhokoli jiného řádkovaného textu). Byl využíván již v předchozích pracích. Velmi usnadňuje vývoj softwaru a to zvlášť za využití webového repozitáře (v tomto případě služba gitlab).

3.2 Rozdělení na funkční celky

Úloha byla rozdělena na základní tři funkční celky tak, jak uvádí kapitola 2.2. Kromě nich byla implementována další funkcionalita a umístěna do samostatných balíčků (*packages*). Avšak nejde o funkcionalitu, která by byla stěžejní. Spíše o části podpůrné.

- `FileExplorer` – balíček, který zpříjemňuje přístup k souborům a stará se o práci s nimi.
- `ImageComparator` – balíček, vytvořený pro porovnávání jednotlivých kvant obrazových dat. Obsahuje především rozhraní `ImageComparator`, jeho vzorovou implementaci a třídu `ComparingResult`, která představuje výsledek porovnání dat. Taktéž obsahuje výčtový typ popisující možnosti matematického zpracování podobnosti vektoru (práce využívá párování bodů na principu Hammingovy vzdálenosti metodou hrubé síly, více informací v části 4.2).
- `ImageData` – v tomto balíčku lze nalézt hlavní třídu `ImageData`, která zastřešuje informace o obrazových datech včetně jejich matematického (vektorového) popisu. Navíc obsahuje strukturu pro uložení informací o známém objektu a jeho zeměpisné souřadnice `GlobalPosition`.
- `ImagePreprocess` – je z většiny přejatý balíček z práce Ing. Jaroslava Štěpána [5]. Přidána metoda pro vytvoření `ImageData`. Díky tomu je složitější vytvořit `ImageData` bez využití filtrů, či jiných úprav (algoritmy pro výpočet klíčových bodů a deskriptorů často pracují s černobílými obrazovými daty). Zároveň zajišťuje, že na všechna `ImageData` vytvořená pomocí jedné instance `ImagePreprocess` budou použity stejné filtry. To samozřejmě platí pouze, pokud filtry nebudou přidávány průběžně. Pokud by bylo záměrem průběžnému přidávání předejít (což ovšem zatím potřeba není), autor doporučuje implementovat třídu `ImagePreprocess` podle vzoru *Stavitel* (*Builder*) [27].
- `Viewer` – balíček starající se o výpis obrazových dat do souboru v zařízení. Klíčová je však jeho vlastnost zápisu klíčových bodů, či homografie do původního obrázku a tím i vizualizace samotného výsledku procesu.

3.3 Rozpoznání obrazu

Tato část funkcionality je představována především výše zmíněným balíčkem `ImageRepository`. Logická struktura byla z většiny převzata z již implementované části projektu (viz práce Ing. Jaroslava Štěpána [5]). Ovšem při implementaci se ukázalo jako vhodné uskutečnit několik změn.

```
public interface IImageFinder {
    ComparingResult findNearestImage(ImageData inputImage,
                                     ImageComparator comparator
                                    )
                                  throws ImageRepositoryException;
}
```

Výpis kódu 3.1: Rozhraní třídy vyhledávající podobná obrazová data.

Původní implementace sice nabízela metodu pro nalezení nejpodobnějších uložených dat vůči danému vstupu, což byla klíčová funkcionalita, jež byla od tohoto balíčku očekávána. Nicméně jako návratový typ volila pouze instanci třídy `ImageMetaData`, jež obsahovala jen globální pozici a kvaternion pro určení úhlu pohledu. Tato instance musela být vložena spolu s původními obrazovými daty a nebyla upravována.

Tento přístup se jevil jako nešťastný, protože neumožňoval použití již spočítaných deskriptorů a klíčových bodů uloženého obrázku mimo tuto třídu. Vzhledem k velmi značné náročnosti tohoto procesu na systémové prostředky bylo rozhodnuto o zapouzdření obrazových dat a jejich spočítaného matematického popisu do třídy, jež by sloužila jako transportní a současně se sama postarala o vypočtení deskriptorů a klíčových bodů. Jde o `ImageData` zmíněnou v seznamu v kapitole 3.2.

Rovněž nebylo v návrhové fázi zřejmé, jakým způsobem bude nejvhodnější porovnávat uložená obrazová data a data vstupní. Z toho důvodu bylo přidáno rozhraní `ImageComparator`, který potřebnou funkcionalitu garantuje a třídy jež jej implementují mohou být v `ImageRepository` použity.

3.4 Výpočet vzdálenosti

Práce na výpočtech vzdálenosti byla zahájena pokusem o výpočet na vektoru sestávajícího se z dvou bodů jež byly spárovány párovacím algoritmem. Za tímto účelem byla navržena metoda `calculateVectorDistance`. Ta požaduje tedy dva body z uloženého obrázku a dva odpovídající ze vstupních dat plus instance `ImageData` popisující uložený i vstupní obrázek. Jak bylo již uvedeno v sekci 2.4 pracuje tato metoda pouze s y souřadnicemi vektorů.

Níže je uveden rozbor práce se vzorcem pro výpočet vzdálenosti (řazeno dle náročnosti zisku hodnoty proměnné):

- *Výšku fotografie v pixelech* je snadno získatelná z Java objektu `BufferedImage`. Ten je možné získat z objektu `File`, jež je součástí `ImageData`.

- *Výšku objektu v pixelech* lze dopočítat z dvojice bodů pocházejících z vstupního obrázku.

- *Skutečnou výšku objektu* lze získat z `ImageData` uloženého obrázku. V žádném případě však nelze pouze zkopírovat výšku objektu na fotografii. Vzhledem k špatné predikovatelnosti pozice krajních bodů vektoru, který je na vstupu metody nelze předpokládat, že vertikální vzdálenost vstupních bodů bude shodná s výškou objektu. Proto je třeba do úložiště `ImageRepository` ukládat pouze obrázky, u kterých bude známa výšku celé fotografie a v této metodě dopočíst y výšku vektoru v milimetrech pomocí poměru:

  $$l_{vs} = \frac{l_{vp} l_{of}}{l_f},$$

  l_{vs} – skutečná výška vektoru [mm], l_{vp} – výška vektoru [pix],
  l_{of} – výška objektu na fotografii [mm], l_f – výška fotografie [pix].

- *Ohnisková vzdálenost čočky fotoaparátu* je teoreticky velmi jednoduše k nalezení na internetu. Pro účely testování není třeba nic jiného, než jí do kódu dodat jako výchozí konstantu. Nejde ale samozřejmě o univerzální postup. Podobně nelze chtít po uživateli, aby nám ohniskovou vzdálenost dodal na vstupu (což by pro účely testování rovněž stačilo). Naštěstí operační systém Android nabízí rozhraní pro získání dat přímo z konkrétního zařízení:

```
Camera camera = Camera.open();
Camera.Parameters cameraParameters =
    camera.getParameters();
double focalLength =
    cameraParameters.getFocalLength();
```

Výpis kódu 3.2: Příklad získání ohniskové vzdálenosti v OS Android.

- *Výška senzoru v milimetrech* již byla zmíněna v kapitole 2.4. Zjistit tuto hodnotu byl velký problém i pouze pro účely testování. Pro některá konkrétní zařízení sice lze velikost senzoru nalézt, zpravidla ale nejde o výšku senzoru [28]. Proto byla napsána samostatná aplikace pro operační systém Android, která pouze vypíše do `TextView` informace o rozměrech senzoru. Dle [29] by měl mít zadní fotoaparát ID o hodnotě 0.

  Toto řešení je pochopitelně určeno pro zjištění údajů při testování mimo zařízení Android. Při provozu v aplikaci v mobilním zařízení bude vybrán jeden fotoaparát, jímž bude nasnímán vstupní obraz a z něj budou současně načteny jeho parametry.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView textView = findViewById(R.id.textView);
        textView.setText("Camera 0 has sensor: " +
        getCameraResolution(0).toString());
    }

    private SizeF getCameraResolution(int camNum)
    {
        SizeF size = new SizeF(0,0);
        CameraManager manager = (CameraManager)
            getSystemService(Context.CAMERA_SERVICE);
        try {
            String[] cameraIds = manager.getCameraIdList();
            if (cameraIds.length > camNum) {
                CameraCharacteristics character = manager.
                    getCameraCharacteristics(cameraIds[camNum]);
                size =
                character.get(
                    CameraCharacteristics.SENSOR_INFO_PHYSICAL_SIZE
                    );
            }
        }
        catch (CameraAccessException e) {
            Log.e(e.getMessage(), e);
        }
        return size;
    }
}
```

Výpis kódu 3.3: Příklad získání rozměrů senzoru fotoaparátu, kód převzat a upraven z [1].

```
public interface IDistanceCalculator {
    double calculateDistance(ImageData input,
                             ComparingResult result)
            throws DistanceCalculatorException;
}
```

Výpis kódu 3.4: Rozhraní třídy, jež vypočítá vzdálenost pozorovatele od objektu.

Metoda `calculateVectorDistance` je součástí třídy `DistanceCalculator` jež implementuje rozhraní `IDistanceCalculator`, se kterým pracují další komponenty aplikace.

Ovšem je třeba zmínit, jak metodu `calculateVectorDistance` využít. Je třeba nějaké vektory vytvořit. Byly testovány dva způsoby:

1. Byly vybrány všechny spárované body a z nich byly vytvořeny vektory metodou každý s každým. Jejich počet byl omezen shora konstantou a rovněž byly filtrovány jednoduchou podmínkou (byla testována vzdálenost bodů, protože existuje předpoklad o existenci větších odchylek na menších vektorech).

2. Byly nalezeny body, jež byly na obrázku nejvýše a nejníže. Z těchto dvou bodů byl vytvořen vektor, na kterém byla spočítána vzdálenost.

Vzhledem k výpočtu na jediném vektoru ve druhé metodě je tato metoda náchylnější na nepřesnosti oproti metodě první, kde dochází k výpočtu na počtu vektorů daleko větším. Přesto však byla vybrána druhá metoda, protože výsledky získané jejím prostřednictvím byly věrohodnější. Rozptyl výsledků, jež dávala metoda první, byl skutečně závratný.

Pro zpřesnění výsledků je samozřejmě možné opakovat celý výpočet vzdálenosti vícekrát za sebou.

3.5 Výpočet polohy

Tento balíček byl z většiny napsán tak, že pracuje s třídou `Coordinates`, která velmi jednoduše znázorňuje souřadnice v kartézské soustavě souřadnic. Právě kvůli jednoduchosti této třídy byla volena cesta s vytvořením vlastní souřadnice místo využití již implementované třídy `Point`, nebo jí podobné, které jsou v Javě k dispozici.

Protože ale zeměpisné souřadnice jsou uchovávány a k dispozici ve formě zeměpisné šířky a délky (latitude a longtitude) bylo navrženo rozhraní `IProjectionCoordinateCalculator`. Třída, jež jej implementuje, umožňuje pohodlný převod ze zeměpisných souřadnic do kartézské soustavy souřadnic. Podrobnější informace byly uvedeny v kapitole 2.5.1.

```
public interface IProjectionCoordinateCalculator {
    double EARTH_RADIUS = 6378000;
    double PROJECTION_COORDINATE_ALLOWED_DERIVATION = 0.05;
    ArrayList<Coordinates> toCoordinates
                (ArrayList<GlobalPosition> points);
    ArrayList<GlobalPosition> fromCoordinates
                (ArrayList<Coordinates> points);
    Coordinates toCoordinates(GlobalPosition position);
    GlobalPosition fromCoordinates(Coordinates position);
    GlobalPosition getCenter();
}
```

Výpis kódu 3.5: Rozhraní třídy, která převádí mezi soustavami souřadnic.

```
public interface ILocationCalculator {
    int ALLOWED_DEVIATION = 1;
    GlobalPosition findPosition(List<Circle> recognizedObjects)
            throws LocationCalculatorException;
}
```

Výpis kódu 3.6: Rozhraní třídy, jež dopočítá konečnou polohu.

Jako převodník souřadnic mezi soustavami, implementuje zmíněné rozhraní třída `EquirectangularProjectionCalculator`. Díky ní lze převod souřadnic uskutečnit velmi jednoduše a přehledně.

Po nalezení prostředků pro získání kartézských souřadnic rozpoznaných objektů a vzdálenosti pozorovatele od nich (což jsou vstupní data pro tuto komponentu) jde z většiny o planimetrickou úlohu popsanou v sekci 2.5.4. Pro tuto úlohu bylo navrženo rozhraní `ILocationCalculator` a jeho implementace `LocationCalculator`. Jako třída pro uložení potřebných údajů byla implementována třída `Circle`, která ukládá zmíněnou zeměpisnou souřadnici, poloměr kružnice (vzdálenost pozorovatele od objektu) a jméno objektu (pro větší přehlednost).

Ovšem problémem je určení konečné polohy. Stojí za povšimnutí, že získáním údajů o dvou rozpoznaných objektech dojde k nalezení (v mnoha případech) dvou stejně důvěryhodných průniků kružnic. Odhadnout s přijatelnou pravděpodobností, který z nich představuje skutečnou polohu pozorovatele můžou pomoci senzory zařízení – zvláště tedy elektronický kompas, nebo i GPS (či podobný systém) za předpokladu větší vzájemné vzdálenosti takových průsečíků. Toto propojení se senzory však není předmětem této práce.

Níže je rozbor situace, kdy jsou k dispozici data (ve formě instancí třídy `Circle`) o alespoň třech objektech. Postup bude následující:

1. Vytvořit dvojice kružnic metodou každý s každým.
2. Pro tyto dvojice spočíst průniky.
3. Z těchto průniků vytvořit jednu kolekci (konkrétněji – tyto průniky uložit do jedné kolekce).
4. Vypočíst konečnou pozici z této kolekce.

Poslední bod seznamu je velmi obecně formulován. Nejde o triviální úlohu. Je naprosto zřejmé, že použít aritmetický průměr pro průsečíky je velmi naivní (a v mnoha případech naprosto nevyhovující a zcestné) řešení. Příkladem je situace zisku dvou bodů průniku – pak by výsledkem byl střed jejich spojnice (samozřejmě pouze v případě stejného počtu výsledků pro oba body, přesněji by šlo říci, že bod získaný průměrem bude dělit spojnici obou průměrovaných bodů v poměru „hlasů“).

Bylo zvoleno řešení pomocí statistické funkce modus. Jednoduše řečeno, souřadnice, která bude obsažena v kolekci průniků nejčastěji, bude prohlášena za výslednou pozici a komponentou vrácena.

Tento přístup má ovšem jeden háček. Vzhledem k tomu, že nejde o práci s přesnými daty (viz sekce 2.5.1), je nezbytné určit, které průniky budou považovány za obrazy jedné souřadnice. Prakticky to je řešeno tím, že rozhraní `ILocationCalculator` definuje konstantu `ALLOWED_DEVIATION`. Pokud je vzdálenost mezi dvěma dopočtenými průniky menší, než tato konstanta, aplikace je vyhodnotí jako souřadnici jedinou a podle toho s nimi v modu počítá.

Aby bylo možné snadno implementovat funkci modus, byla vytvořena třída `AverageList`, která ukládá jednotlivé pozice a současně nabízí metodu pro získání průměru z uložených pozic.

Při procházení celé kolekce průniků tedy dochází ke třídění do kolekce `AverageList`. Po projití všech průniků se vybere `AverageList` s největším počtem obsažených průniků, vypočítá se z nich průměr a ten se vrátí jako výsledná pozice.

Kapitola 4

Testování

Tato kapitola rozebírá testování aplikace a shrnuje pozorování, jež z testování plynou. Proběhlo více typů testů:

- Byly využívány takzvané *Unit* testy, jež testují jednotlivé funkce a metody. Tím pomáhají zajistit, že aplikace neobsahuje příliš velké množství chyb.
- Byly rovněž napsány *integrační* testy pro zajištění kompatibility jednotlivých částí aplikace. Testování této spolupráce nebylo příliš komplikované díky rozdělení již v počátku vývoje. Tato část testování probíhala „black box“ metodou – tedy docházelo k testování oproti rozhraní a testy byly psány „bez znalosti implementace“.
- Testování hlavních částí aplikace rovněž probíhalo manuálně, protože automatizace těchto testů by byla náročná a není předmětem této práce.

4.1 Rozpoznání obrazu

V rámci testování části pro rozpoznávání obrazu byly vytvořeny sady obrázků a na nich bylo testováno, zda z dané sady budov nalezne aplikace správný.

Testování bylo komplikováno prací s relativně malými obrázky. V databázi obrazových dat je využíváno obrázků o maximálních rozměrech 480 x 640 pixelů. Důvod je jednoduchý – rychlost zpracování. Pokud má být systém využitelný v budoucnu pro určování polohy v mobilním zařízení, musí být zpracování přijatelně rychlé. Dalším důvodem je nezbytná potřeba rozsáhlé databáze obrazových dat a tedy značná náročnost na vnitřní úložiště zařízení. Vnitřní úložiště mobilních zařízení každým rokem stoupá, ale přesto nelze předpokládat, že by uživatele potěšilo, že modul pro pouhé upřesnění polohy zabírá řádově GB z paměti jeho zařízení.

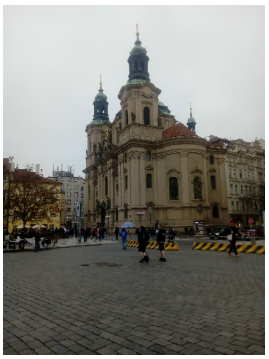
1

2

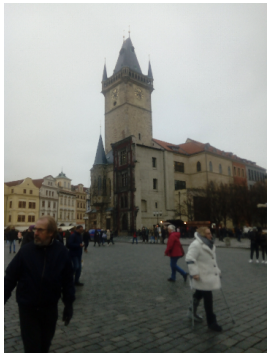
3

4

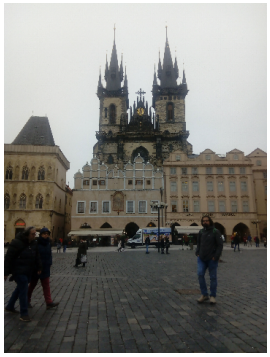
5



6



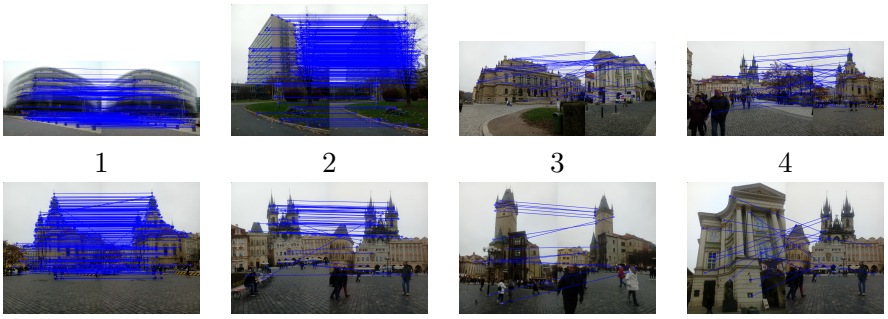
7



8

9

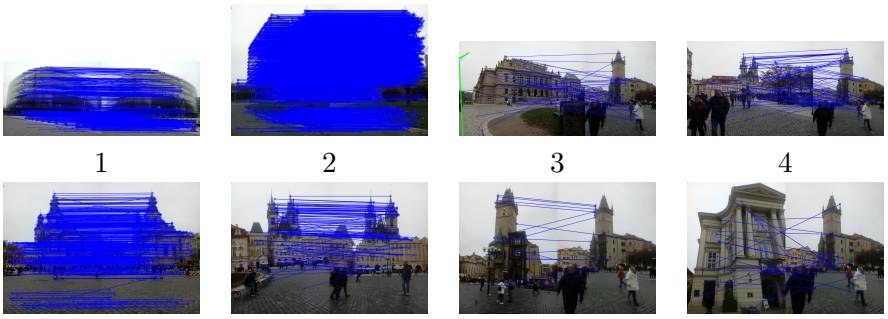
Tabulka 4.1: Testovaný soubor obrazových dat.



1 2 3 4

5 6 7 8

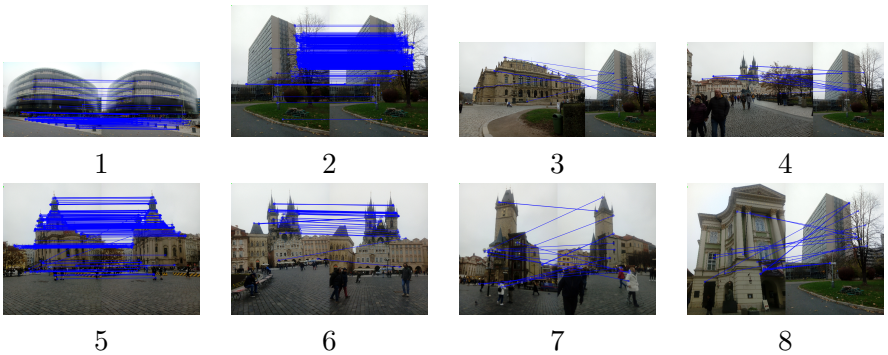
Tabulka 4.2: Výsledky párování algoritmem AKAZE.



1 2 3 4

5 6 7 8

Tabulka 4.3: Výsledky párování algoritmem BRISK.



1 2 3 4

5 6 7 8

Tabulka 4.4: Výsledky párování algoritmem ORB.

Nutno ovšem přiznat, že pokud je jako vstupní data zadána fotografie skutečně známé a uložené budovy (objektu apod.), která ovšem bude vyfocena z dálky, snadno dojde k problémům s rozpoznáním. K velkým problémům docházelo především, pokud se na obrazových datech vyskytl strom (stejný efekt by byl pozorovatelný i u jiných objektů, ale speciálně u stromů byl v průběhu testování opakovaně pozorován). Algoritmus pro výpočet klíčových bodů detekoval mnoho hran a prudkých změn na obrázku v oblasti větví a listů a klíčových bodů v těchto místech vygeneroval opravdu mnoho. Proto následně `ImageComparator` nevyhodnotil podobnost jako zvlášť vysokou, protože poměr spárovaných bodů nebyl vysoký.

Jako řešení se jeví možnost vyhodnocovat poměr z počtu spárovaných klíčových bodů vůči celkovému počtu klíčových bodů z uloženého obrázku. To omezí efekt špatného rozpoznávání obrazových dat obsahujících členité objekty, jež nejsou vhodné pro porovnávání. Předpokládá to však velmi pečlivě připravenou databázi známých objektů.

4.1.1 Výsledky

Výsledky v tabulkách 4.2, 4.3 a 4.4 obsahují porovnání výsledků manuálních testů části rozpoznání obrazu se zvýrazněním spárovaných klíčových bodů. Zajímavé je, že testované algoritmy mají na testovacím vzorku obrazových dat prakticky shodnou úspěšnost rozpoznání budov. Nelze ale zapomenout na parametr množství klíčových bodů potřebných pro rozpoznání objektu. Zejména při testování obrázků číslo 2 či 5 je výhodnost zvoleného algoritmu ORB vidět.

Obrázky číslo 3 a 8 nespároval správně ani jeden z algoritmů. Jako nejvýznamnější důvod tohoto faktu se jeví neúplnost databáze, neboť v obou případech došlo ke značné změně úhlu pohledu na danou budovu.

Zajímavý je případ testování obrázku číslo 4. Týnský chrám (jenž je také na obrázku 6) je zde vyfocen ze značné vzdálenosti. Ani jeden z algoritmů jej nebyl schopen správně rozpoznat – možná proto, že na samotné fotografii zabírá malou oblast. Pokud by nedošlo ke zlepšení při využití většího rozlišení obrázku, mohlo by to naznačovat omezení funkčnosti celého modulu. Pokud totiž nebudeme schopni rozeznat budovu (architektonický prvek apod.), není možné dostat korektní výsledky i při předpokladu správné funkce částí dalších (což je ovšem při principu výpočtu vzdálenosti velmi smělý předpoklad).

4.2 Výpočet vzdálenosti

Pro výpočet vzdálenosti bylo vytvořeno několik sad obrazových dat a tato část byla testována na více typech objektů. Mimo jiné šlo o:

- Testy na skutečných budovách. Mohly však být použity pouze ty, u kterých byla dohledatelná alespoň přibližná výška.
- Testy funkčnosti v umělém, připraveném prostředí pro optimalizaci a zjednodušení výpočtu.

V průběhu testování bylo dbáno na to, aby nedošlo k problémům s rozpoznáním obrazu pomocí funkce `assert`.

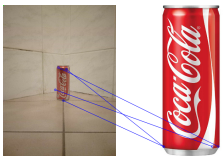
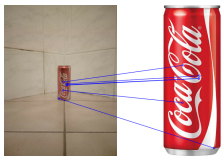
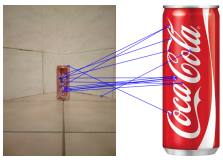
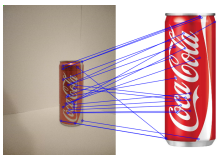
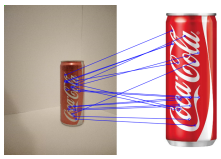
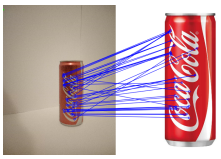
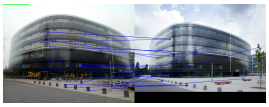
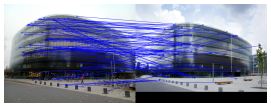
Probíhaly rovněž testy různých párovacích algoritmů. OpenCV nabízí tyto algoritmy:

1. `FLANNBASED`
2. `BRUTEFORCE`
3. `BRUTEFORCE_L1`
4. `BRUTEFORCE_HAMMING`
5. `BRUTEFORCE_HAMMINGLUT`
6. `BRUTEFORCE_SL2`

Ukázalo se však, že původní výběr metody `BRUTEFORCE_HAMMING` dává stejně jako `BRUTEFORCE_HAMMINGLUT` nejlepší výsledky. Zobrazená data jsou tedy výsledkem využití prvního zmíněného algoritmu.

4.2.1 Výsledky

Tabulka 4.5 ukazuje výsledky párování klíčových bodů a naměřenou vzdálenost (vzdálenosti jsou zaokrouhlené na celé centimetry – s výjimkou případu, kdy bylo naměřeno méně, než centimetr; obrázky napravo převzaty z [30], [31]). Při porovnávání výsledků je třeba si všimnout kromě správnosti spárování klíčových bodů také jejich počtu. Pokud se podaří algoritmu rozpoznat obraz (což se předpokládá v části 4.1) s menším počtem klíčových bodů, bude zpracování rychlejší – což je jen pozitivum. Samozřejmě proto, že je využívána metoda výpočtu z nejvyššího a nejnižšího bodu. Pokud by byla použita metoda využívající více vektorů, jejich větší počet (plynoucí z většího počtu spárovaných bodů) by mohl být výhodný.

AKAZE	BRISK	ORB
		
0,3 cm	80 cm	66 cm
Skutečná vzdálenost byla 50 cm.		
		
35 cm	34 cm	34 cm
Skutečná vzdálenost byla 25 cm.		
		
3 267 cm	3 351 cm	3 700 cm
Odhadovaná vzdálenost 33 metrů.		

Tabulka 4.5: Ukázka párování klíčových bodů vstupních dat na uložený obrázek včetně dopočítané vzdálenosti.

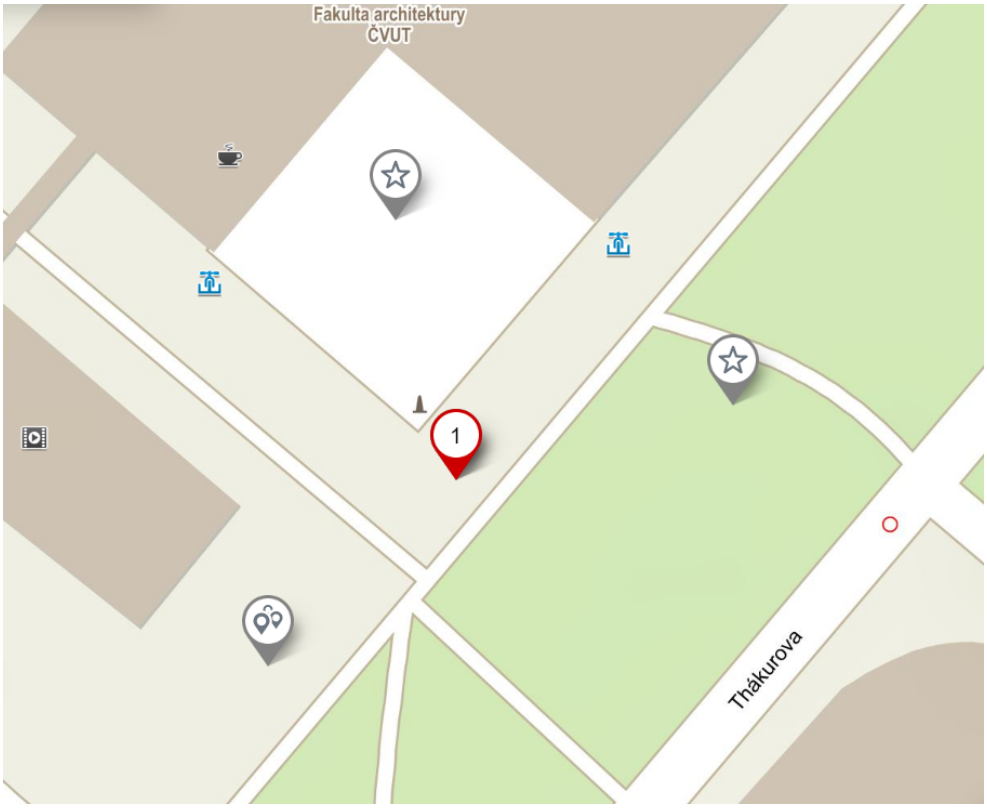
4.3 Výpočet polohy

Při testování komponenty pro výpočet polohy bylo využito mimo jiné vzorce Haversine, jak bylo uvedeno v kapitole 2.5.2. Po převedení dvou souřadnic do kartézské soustavy byla spočítána jejich vzdálenost pomocí Pythagorovy věty a testováno, zda výsledek odpovídá výsledku výpočtu vzdálenosti z původních dvou zeměpisných souřadnic právě skrz vzorec Haversine.

Dalším způsob testování tkvěl v simulaci skutečného stavu. Třídě pro výpočet polohy byly dodány tři body a vzdálenosti těchto bodů od pozorovatele a testován byl výstup.

Testovací případy byly vytvářeny pomocí internetových map [32] a jejich funkcionality, jež například nabízí možnost měřit vzdálenosti. Žel citovaní stránka, jež jsme využívali podporuje měření se zaokrouhlením na metry. U testovacího případu na obrázku 4.1 bylo dosaženo přesnosti 80 cm vůči výsledku vzorce Haversine, což lze považovat za velký úspěch vzhledem k přesnosti měření. Dá se předpokládat, že za použití přesnějšího měření na mapách by bylo možné dosáhnout o mnoho přesnějších výsledků.

Pomocí těchto testů bylo možné ověřit použitelnost této části a její správnou funkčnost.



Obrázek 4.1: Ukázka testování proti rozhraní ILocationCalculator.

4.4 Diskuze odchylky

Zdrojů odchylky je hned několik (řazeno dle částí aplikace):

1. Špatné rozpoznání objektu na vstupních datech. To je fatální problém, další operace nejsou třeba a výsledek není možné získat. Tomuto problému je možné předcházet pouze vhodně vytvořenou databází známých objektů.

   Pro detekci tohoto problému je možné použít omezující podmínku pro poměr klíčových bodů všech a spárovaných. Nejde však o univerzální řešení, snadno najdeme příklad, kdy tato detekce nepomůže a naopak může zkomplikovat situaci.

2. Výpočet vzdálenosti je klíčovou částí systému. Pokud nedojde k fatální chybě v části pro rozpoznávání obrazu, je hlavním zdrojem nepřesností a odchylek. Největším problémem je zde nedostatečně spolehlivé párování klíčových bodů (viz tabulky 4.2, 4.3 a 4.4). Žel detekovat chyby

v této části je velmi obtížné. Jedna z možností je porovnávat výsledky lokalizačního modulu s daty ze senzorů zařízení.

Jako další vylepšení výpočtů vzdálenosti by bylo možné vyzkoušet uplatnění statistických matematických postupů (konfidenční intervaly, ...) na větší množství vygenerovaných vektorů, či se pokusit pomocí vypočtené homografie ověřovat změny polohy vektoru složeného z nejvyššího a nejnižšího bodu. Možností by bylo rovněž vyzkoušet funkcionalitu knihovny OpenCV týkající se kontur.

Z tabulky 4.5 je vidět, že odchylka současného měření je i u algoritmu ORB, jež dosahuje nejlepších výsledků, velmi značná (až 36%). Ovšem z testování této části rovněž vyplývá, že výsledek je velmi silně závislý na spárování konkrétních bodů. Na základě toho vidí autor velký potenciál ve výše zmíněných optimalizačních metodách.

3. Odchylka vznikající převodem mezi soustavami souřadnic se jeví jako nejméně závažná. Průměrně vzniklá odchylka (mezi vzdáleností vypočítanou Pythagorovou větou a Haversine vzorcem v testech) vychází přibližně 1,2 cm. Největší naměřená odchylka odpovídala necelým 5 cm při vzdálenosti bodů přes 130 m.

   Větší problém by mohl představovat mechanizmus výběru výsledné souřadnice ze souboru průniků „kružnic“. Bude třeba experimentálních testů pro zjištění optimálního nastavení konstanty `ALLOWED_DEVIATION`.

   Upřesňující data by mohlo být možno získat ze senzorů zařízení. Případnou neschopnost modulu získat polohu z daných dat by bylo možno rovněž detekovat nemožností vypočíst modus (respektive určení modu by nebylo jednoznačné).

4. Na celkový výsledek by mohl mít rovněž nemalý vliv fakt, že při současné implementaci je kupříkladu budova určena jedinou zeměpisnou souřadnicí, což je jednoduše řečeno nepřesné.

   Tento zdroj nepřesnosti by mělo být možné redukovat používáním menších objektů v databázi. Další možností je při zachycení celé budovy ukládat přibližný střed její fasády.

Závěr

Cíli stanovenými v úvodu této práce bylo analyzovat již zpracovanou část projektu a možnosti strojového zpracování obrazu. Na tomto základě analyzovat, navrhnout a implementovat lokalizační modul. Ten poté podrobit testům a diskutovat výsledky.

Všechny tyto body byly zpracovány na základě již vypracovaných částí projektu. Strojové zpracování obrazu bylo analyzováno primárně v kontextu s již využívanými technologiemi, jež projekt nabízí.

Jako vhodné bylo shledáno využití rozdělení problému na tři části, jež byly řešeny odděleně. Byly rovněž diskutovány odchylky měření, jejich možné původy a rovněž byly navrženy možné způsoby jejich redukce.

Tyto poznatky mohou být základem pro navazující práce. Jde především o vybudování vhodné obrazové databáze a stanovení doporučení pro rozpoznávání obrazu optimálním způsobem. Dále o zpřesnění a vylepšení komponenty pro výpočet vzdálenosti. A konečně o propojení a integrace lokalizačního modulu do prostředí OS Android.

Autor práci shledává jako úspěšnou. Všechny cíle se podařilo splnit a byl položen značný základ pro uvedení sepsaných technologií do praxe a běžného života.

Bibliografie

1. *How to get Camera Sensor Size in android device?* [online]. 1961 [cit. 2020-03-30]. Dostupné z: https://stackoverflow.com/questions/8104252/how-to-get-camera-sensor-size-in-android-device.
2. BRICKEN, Meredith. Virtual Reality Learning Environments: Potentials and Challenges. *SIGGRAPH Comput. Graph.* [online]. 1991, roč. 25, č. 3, s. 178–184 [cit. 2020-03-23]. ISSN 0097-8930. Dostupné z DOI: 10.1145/126640.126657.
3. HOSCH, William L. *Augmented reality* [online]. Encyclopædia Britannica, inc., 2020 [cit. 2020-03-23]. Dostupné z: https://www.britannica.com/technology/augmented-reality.
4. ZANDBERGEN, Paul A. Accuracy of iPhone Locations: A Comparison of Assisted GPS, WiFi and Cellular Positioning. *Transactions in GIS* [online]. Roč. 13, č. s1, s. 5–25 [cit. 2020-03-23]. Dostupné z DOI: 10.1111/j.1467-9671.2009.01152.x.
5. JAROSLAV, Štepán. Věnná města českých královen – Modul rozpoznání obrazu. [online]. 2019 [cit. 2020-03-23]. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/82620/F8-DP-2019-Stepan-Jaroslav-thesis.pdf?sequence=-1&isAllowed=y.
6. *OpenCV modules* [online] [cit. 2020-03-23]. Dostupné z: https://docs.opencv.org/4.2.0/.
7. *SURF Algorithm* [online] [cit. 2020-03-26]. Dostupné z: http://people.ee.ethz.ch/~surf/index.html.
8. *SIFT: SCALE INVARIANT FEATURE TRANSFORM* [online] [cit. 2020-03-26]. Dostupné z: https://uilo.ubc.ca/sift-scale-invariant-feature-transform.

9. TAREEN, S. A. K.; SALEEM, Z. A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In: *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)* [online]. 2018, s. 1–10 [cit. 2020-03-23]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8346440.

10. UIJLINGS, Jasper RR; VAN DE SANDE, Koen EA; GEVERS, Theo; SMEULDERS, Arnold WM. Selective search for object recognition. *International journal of computer vision*. 2013, roč. 104, č. 2, s. 154–171.

11. *Subject distance* [online] [cit. 2020-03-23]. Dostupné z: https://www.scantips.com/lights/subjectdistance.html.

12. MAHMOUD, H.; AKKARI, N. Shortest Path Calculation: A Comparative Study for Location-Based Recommender System. In: *2016 World Symposium on Computer Applications Research (WSCAR)* [online]. 2016, s. 1–5 [cit. 2020-03-23]. ISSN null. Dostupné z DOI: 10.1109/WSCAR.2016.16.

13. *Kartografická zobrazení podle konstrukční osy* [online] [cit. 2020-03-23]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=59996.

14. DORAISWAMY, Selvarajkumar; D., Michael; OLSON, George. *50 Map Projections Types: A Visual Reference Guide [BIG LIST]* [online]. 2020 [cit. 2020-03-23]. Dostupné z: https://gisgeography.com/map-projection-types/.

15. BREUSS, Martin. *Azimuthal Projection: Orthographic, Stereographic and Gnomonic* [online]. 2020 [cit. 2020-03-23]. Dostupné z: https://gisgeography.com/azimuthal-projection-orthographic-stereographic-gnomonic/.

16. ČÁBELKA, Miroslav. *Projection detection* [online] [cit. 2020-03-26]. Dostupné z: https://web.natur.cuni.cz/~bayertom/index.php/projection-analysis/projection-analysis.

17. *Conic Projection: Lambert, Albers and Polyconic* [online]. 2020 [cit. 2020-03-23]. Dostupné z: https://gisgeography.com/conic-projection-lambert-albers-polyconic/.

18. *Cylindrical Projection: Mercator, Transverse Mercator and Miller* [online]. 2020 [cit. 2020-03-23]. Dostupné z: https://gisgeography.com/cylindrical-projection/.

19. *Azimuthal Equidistant Projection* [online] [cit. 2020-03-23]. Dostupné z: https://mathworld.wolfram.com/AzimuthalEquidistantProjection.html.

20. *Bonne Projection* [online] [cit. 2020-03-23]. Dostupné z: https://mathworld.wolfram.com/BonneProjection.html.

21. *Conic Projection* [online] [cit. 2020-03-23]. Dostupné z: https://mathworld.wolfram.com/ConicProjection.html.

22. *Cylindrical Projection* [online] [cit. 2020-03-23]. Dostupné z: https://mathworld.wolfram.com/CylindricalProjection.html.

23. *Haversine vzorec - Haversine formula* [online] [cit. 2020-03-23]. Dostupné z: https://cs.qwe.wiki/wiki/Haversine_formula.

24. *Cylindrical Equidistant Projection* [online] [cit. 2020-03-23]. Dostupné z: https://mathworld.wolfram.com/CylindricalEquidistantProjection.html.

25. *Circle-circle intersection points* [online] [cit. 2020-03-23]. Dostupné z: https://stackoverflow.com/questions/3349125/circle-circle-intersection-points.

26. SANGAPPA, Sudhir; PALANIAPPAN, K.; TOLLERTON, Richard. Benchmarking Java against C/C++ for Interactive Scientific Visualization. In: *Proceedings of the 2002 Joint ACM-ISCOPE Conference on Java Grande* [online]. Seattle, Washington, USA: Association for Computing Machinery, 2002, s. 236 [cit. 2020-03-23]. JGI '02. ISBN 1581135998. Dostupné z DOI: 10.1145/583810.583848.

27. *Design Patterns and Refactoring* [online] [cit. 2020-03-30]. Dostupné z: https://sourcemaking.com/design_patterns/builder.

28. *The Phone Camera Sensor: A Simple Introduction* [online]. 2019 [cit. 2020-03-30]. Dostupné z: https://thesmartphonephotographer.com/phone-camera-sensor/.

29. *Documentation android.hardware.camera2* [online] [cit. 2020-03-30]. Dostupné z: https://developer.android.com/reference/android/hardware/camera2/package-summary.html?authuser=1.

30. *Coca-Cola Can 330ml* [online] [cit. 2020-04-04]. Dostupné z: https://homeline.gr/wp-content/uploads/2018/05/coco_cola_330ml.jpg.

31. *Národní technická knihovna v Praze* [online]. 2016 [cit. 2020-04-04]. Dostupné z: https://www.cka.cz/cs/svet-architektury/seznam-architektu/ing-arch-lesek-petr/narodni-technicka-knihovna-v-praze/.

32. *Mapy* [online] [cit. 2020-04-04]. Dostupné z: https://mapy.cz/zakladni?x=14.4016000&y=50.1051000&z=11.

Příloha **A**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS