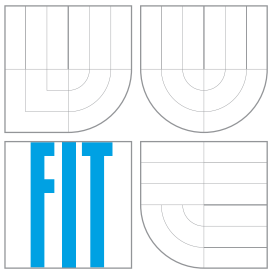
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZPRACOVÁNÍ GPS ZÁZNAMŮ

POSTPROCESSING OF GPS LOGS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROMAN JAŠKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ POLOK

BRNO 2016

Abstrakt

Cieľmi tejto práce boli analýza chýb v GPS záznamoch aktivít, identifikácia algoritmov vhodných na ich korekciu a návrh riešenia tohoto problému spojený s implementáciou Android aplikácie. Prvá kapitola sa zaoberá všeobecným popisom, ukážkou problému a motiváciou pre voľbu tejto témy. Druhá kapitola sa venuje podrobnejšiemu rozkladu tohoto problému. Konkrétne v nej rozoberám zdroje nepresností, formát vstupných dát, existujúce riešenia, matematické metódy vhodné pre tento problém, API využiteľné pre riešenie a odôvodnenie voľby implementačnej platformy. Kapitola číslo tri popisuje konkrétny návrh aplikácie pre systém Android ako riešenia tohoto problému. Sú v nej popísané požiadavky na výslednú aplikáciu, detaily funkcionality, konkrétne využitie Kalmanovho filtra, jednotlivých API, knižníc ako i návrh užívateľského rozhrania aplikácie. Predposledná kapitola s číslom štyri približuje vybrané detaily implementácie ako samotný Kalmanov filter, či problematiku manuálneho presúvania bodov trasy. Taktiež sa v tejto kapitole nachádza popis zmien návrhu počas implementácie. Posledná kapitola zhodnocuje dosiahnuté výsledky Kalmanovho filtra a upínania úsekov trasy cesty pomocou *Google Maps Roads API*. V tejto kapitole popisujem i možnosti ďalšieho rozšírenia aplikácie.

Abstract

This thesis aims to analyse common errors in recorded GPS activities, identify algorithms suitable for correction of these problems and design and subsequently implement an Android application. The first chapter contains an introduction to the problem, demonstration of a few affected examples and explains the motivation behind the choice of this subject for my thesis. Second chapter decomposes the established problem. I specifically elaborate on the source of the errors, the format of input data, existing solutions, mathematical methods suitable for this problem, available APIs and reasons behind the selection of the final platform. Chapter number three describes the actual solution of the problem employed in the final application. The requirements for the final application are laid out as well as the specific details of the functionality, such as the usage of Kalman filter, individual APIs, libraries as well as the design of the user interface. The penultimate chapter reveals selected parts of the implementation, such as the Kalman filtering itself or the problem of manual track correction. This chapter also contains description of design changes executed during the implementation. The final chapter reviews the achieved results of Kalman filtering and snapping of points to roads using *Google Maps Roads API*. Ultimately, I elaborate on possibilities of continued development of the application.

Klíčová slova

GPS, trasy, filtrovanie, chyby, nepresnosť, záznamy, Kalmanov filter, Android, Roads API, upínanie na cesty

Keywords

GPS, tracks, filtering, errors, inaccuracy, logs, Kalman filter, Android, Roads API, snap to road

Citace

JAŠKA, Roman. *Zpracování GPS záznamů*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Polok Lukáš.

Zpracování GPS záznamů

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pana Ing. Lukáša Poloka. Uviedol som všetky literárne pramene, publikácie a internetové zdroje, z ktorých som čerpal.

.....
Roman Jaška
15. května 2016

Poděkování

Chcel by som sa poďakovať vedúcemu mojej práce, Ing. Lukášovi Polokovi, za to, že mi poskytol možnosť zvoliť si vlastné zadanie práce a za nasmerovanie v hľadaní riešenia problému. Taktiež ďakujem Kristíne Jankovičovej za priebežné testovanie užívateľkého rozhrania, morálnu podporu a typografickú i gramatickú kontrolu.

Obsah

Kapitola 1

Úvod

V súčasnosti je k dispozícii široký repertoár zariadení schopných zaznamenávania pohybu pomocou technológie *„Global Positioning System“* (ďalej len *GPS*). Medzi tieto zariadenia patria napríklad profesionálne GPS prijímače, dedikované športové zariadenia, cyklistické počítače či v neposlednom rade mobilné telefóny. Aktuálne mobilné telefóny predstavujú podstatný podiel zariadení využívaných pre športové účely. Hlavným dôvodom je ich všeobecné rozšírenie a skutočnosť, že takmer každý komerčne dostupný smartphone obsahuje integrovaný GPS senzor.

Vďaka týmto faktom nevzniká potreba zakúpenia dodatočného GPS zariadenia a tvorba záznamov je bežnému užívateľovi omnoho prístupnejšia ako v minulosti. Softwarových možností pre zaznamenávanie aktivít je taktiež veľké množstvo, pričom väčšina z dostupných aplikácií poskytuje minimálne základnú funkcionalitu zdarma.

1.1 Popis problému

Vstavané GPS zariadenia v mobilných zariadeniach často dosahujú značne podpriemerné výsledky. Zatiaľ čo za ideálnych podmienok sú záznamy relatívne uspokojivé, ich kvalita rázne klesá v mestskom prostredí. Signál je na takýchto miestach značne skreslený a znehodnotený okolitými budovami a rôznymi zdrojmi rušenia. Vo výsledku často krát vznikne záznam aktivity plný nepresností. V lepšom prípade sú tieto nepresnosti relatívne malé a postrehnuteľné až pri dlhom zázname, keďže sa na seba postupne nabaľujú. Pri väčších nepresnostiach sa zaznamenané pohyby javia ako chaotické, keďže obsahujú rôzne vybočenia či extrémne skoky, často krát rádovo v desiatkach metrov od skutočnej trasy.

Pri pohľade na výstup takto znehodnoteného záznamu je i neskúsenému pozorovateľovi jasné, že niečo nieje v poriadku. V opačnom prípade by to znamenalo, že osoba počas záznamu dosahovala častokrát nadľudských rýchlostí alebo sa nezmyselne pohybovala zo strany na stranu na zdanlivo rovnej ulici. Takéto chybné údaje v konečnom dôsledku vedú k skresleniu výsledných štatistík odvodených zo získaného záznamu. Príkladom takto znehodnotených údajov sú priemerná či maximálna rýchlosť, prípadne celková prekonaná vzdialenosť.

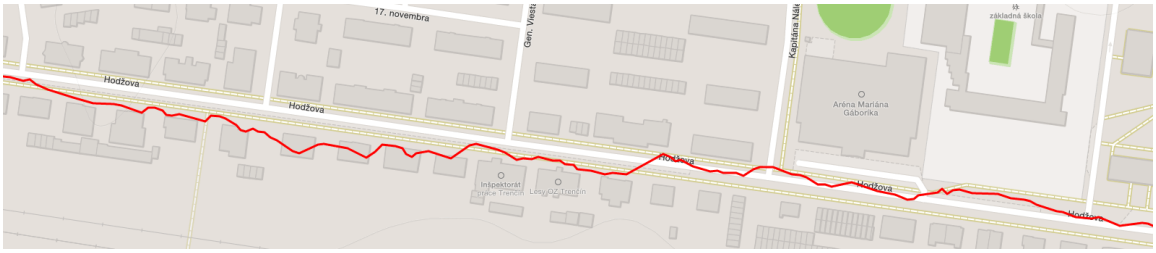
Túto tému som si zvolil, keďže som sa vo svojom dennodennom živote často stretával so spomenutým problémom a nenašiel som žiadne existujúce a uspokojivé riešenie dodatočného spracovania záznamov. Pri písaní tejto práce predpokladám, že čitateľ má základnú predstavu o princípe fungovania GPS.

1.2 Ukážka problému

Problém je demonštrovateľný na dátach zobrazených pomocou spojníc bodov na mape. Na takto zobrazenej trase je pri dostatočnom priblížení možné jasne pozorovať nedostatky v kvalite záznamu. Chyby je možné rozdeliť do dvoch skupín.

1.2.1 Zanášanie

Zanášanie, pozorovateľné na obrázku 1.1, je najbežnejším problémom. Jedná sa o kľučkovanie trasy v dôsledku neustále sa meniacej presnosti vzoriek zachyteného polohy. Tento jav je prítomný v každom zázname, avšak jeho závažnosť závisí vo veľkej miere na kvalite prijímača. Nekvalitný senzor vykazuje toto správanie i za ideálnych podmienok. Naopak, pri kvalitnom senzore sú výkyvy minimálne, avšak prítomné. Rýchlosť pohybu alebo vzorkovania taktiež ovplyvňuje mieru výskytu zanášania. Čím rýchlejší je zaznamenávaný pohyb, teda čím nižší je počet vzoriek na danom úseku, tým menej chýb sa prejaví.



Obrázek 1.1: Zanášanie v zaznamenanej trase pri pešej chôdzi rovnou ulicou.

1.2.2 Skok

Skok nastáva pri náhlom znížení kvality signálu, alebo pri opätovnom nadobudnutí signálu po jeho strate. Tomuto popisu zodpovedá predovšetkým prechod tunelom či podjazdom, ale i vstup do oblasti so zhoršenou kvalitou signálu, akou je napríklad rušná križovatka. Ďalším zdrojom tohoto javu môže byť i vyťaženie zaznamenávajúceho zariadenia. Ku skokom dochádza, keď už telefón nestíha zaznamenávať aktuálnu polohu v pravidelných intervaloch v dôsledku nedostatku systémových zdrojov. Ukážka skoku sa nachádza na obrázku 1.2.

Obrázek 1.2: Skok v zaznamenanej trase pri jazde na bicykli na rušnej mestskej križovatke.

Kapitola 2

Analýza problému

Táto kapitola sa zaoberá popisom jednotlivých častí problému filtrovania GPS záznamov a identifikáciou vhodných riešení.

2.1 Zdroje nepresností

Nepresnosti sa do záznamov dostávajú z viacerých dôvodov. Najvýznamnejšími dôvodmi sú:

- Nekvalitný GPS prijímač
- Rušenie signálu okolím
- Nedostatok viditeľných GPS družíc
- Preťaženie záznamového zariadenia

Kvalita GPS prijímača ovplyvňuje predovšetkým mieru zanášania vo výslednom zázname. Je to spôsobené veľkými výkyvmi v chybe nameraných údajov a teda v presnosti jednotlivých vzoriek. Nekvalitný GPS prijímač je náchylnejší na stratu signálu a z pravidla mu trvá omnoho dlhšie, kým zachytí prvotnú polohu.

Vplyv okolia na GPS signál taktiež nie je zanedbateľný. Okolité budovy, vozidlá, ale i oblačnosť skresľujú signál. Toto je obzvlášť badateľné pri menej kvalitných prijímačoch.

Určenie polohy pomocou GPS sa spolieha na 21 navigačných satelitov na obežnej dráhe. Počet viditeľných družíc ovplyňuje presnosť určenej polohy, pričom minimum pre určenie polohy sú tri družice. Čím viac družíc zariadenie „vidí“, tým presnejšia bude určená poloha.

Záznamové zariadenia, v našom prípade mobilné telefóny, taktiež nemajú k dispozícii neobmedzené systémové zdroje. V prípade, že popri GPS zázname telefón spracúva niekoľko ďalších náročných úloh bežiacich v reálnom čase, môže sa stať, že dôjde ku strate vzoriek. Príkladom môže byť súbežné zaznamenávanie polohy a videa na pozadí sprevádzané navigačnou aplikáciou v popredí.

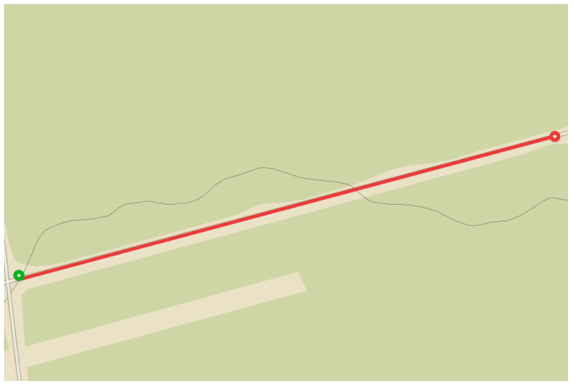
2.2 Vstupný formát

S rozšírením dostupnosti osobných GPS zariadení vzniklo taktiež veľké množstvo formátov pre ukladanie záznamov. Rôzny výrobcovia používali rôzne formát čo často viedlo ku

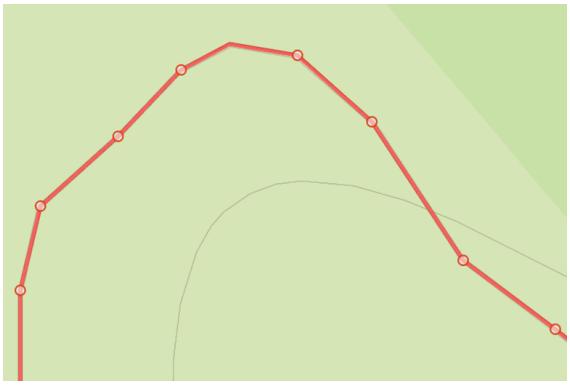
komplikáciám pri zdieľaní takýchto záznamov. Preto v roku 2004 vznikol formát *„GPS Exchange Format“*, skrátene *GPX*. Tento formát vychádza z rozšíriteľného značkovacieho jazyka XML. Formát GPX sa od roku 2004 rýchlo rozšíril a dnes je využívaný prakticky každým moderný softwarom a hardwarom pracujúcim s aktivitami zaznamenanými pomocou GPS. Formát je schopný ukladať viaceré trasy v jednom súbore, pričom trasa môže, avšak nemusí, obsahovať časové údaje o jednotlivých bodoch. Zjednodušene by sa dal tento formát popísať ako zoznam párov desatinných hodnôt, jedna pre zemepisnú šírku a jedna pre zemepisnú dĺžku. Každý záznam v tomto zozname môže ďalej obsahovať informácie ako čas, nadmorskú výšku, okamžitú rýchlosť, presnosť GPS polohy a podobne. Pre potreby tejto práce však budem rátať len s dvoma základnými prípadmi, ktorými sú:

- Naplánovaná trasa bez časových údajov
- Záznam aktivity s časovými údajmi

Filtrovanie trás bez časových údajov bude menej presné, keďže môžu byť vzorkované veľmi nerovnomerne. Ako príklad nám môže poslúžiť takto uložená naplánovaná trasa, kde bude dlhý rovný úsek reprezentovaný dvoma bodmi, obrázok 2.1, zatiaľ čo ztáčajúca sa časť trasy na obrázku 2.2 si vyžiada podstatne viac bodov.



Obrázek 2.1: Nízke vzorkovanie jednoduchej trasy.



Obrázek 2.2: Vyššie vzorkovanie zložitej trasy.

Pri filtrovaní záznamov aktivít, teda pri prítomnosti časových údajov, môžeme očakávať pravidelnejšie vzorkovanie ako v minulom prípade. Pravidelnosť vzorkovania však nie je zaručená a môže sa meniť z rôznych dôvodov. Najčastejšími z týchto dôvodov sú vyťaženie zariadenia počas záznamu alebo strata družicového signálu.

Obrázek 2.3: Pravidelné vzorkovanie záznamu skutočnej aktivity

2.3 Existujúce riešenia

Existujúce aplikácie je možné rozdeliť do dvoch skupín:

- Záznamové aplikácie
- Analyzačné aplikácie

Záznamové aplikácie sa vo väčšine prípadov zameriavajú na športové aktivity. Bývajú napojené na webové služby, ktoré poskytujú detailné informácie o zaznamenaných aktivitách, pričom sú zároveň sociálnymi sieťami. Aplikácií tohoto typu je neúrekom a sú bežne používané. Patria medzi ne napríklad *Strava*, *RunKeeper*, *Endomondo*, *Runtastic* a mnohé ďalšie. Tieto aplikácie súbor so záznamom aktivity odosielajú na vzdialený server, kde je ďalej spracovaný. Tento proces býva pred užívateľom skrytý a prístup k samotným GPX súborom je v extrémnych prípadoch nemožný alebo prípadne dostupný len cez webové rozhranie danej služby. Dodatočné spracovanie záznamov na webe spočíva len vo výpočte rôznych údajov zo vstupných dát bez akéhokoľvek filtrovania. Ako dôkaz som uskutočnil jednoduchý experiment:

1. Zaznamenal som aktivitu pomocou aplikácie *Strava*.
2. Z webu služby som stiahol spracovaný GPX súbor.
3. Hodnoty som náhodne pozmenil.
4. Upravený súbor som nahral ako novú aktivitu.
5. Po spracovaní aktivity som opäť stiahol GPX.
6. Porovnal som upravený GPX s novo stiahnutým GPX.

Vďaka tomuto experimentu som sa uistil, že na strane servera sa trasa nijakým významným spôsobom nefiltruje.

Menšiu skupinu tvoria jednoduché aplikácie slúžiace na čistý záznam aktivity. Tieto aplikácie sa nespoliehajú na vzdialený server na dodatočné vyhodnotenie údajov a často produkujú priamo GPX súbory. Medzi tieto aplikácie patria napríklad *Google My Tracks*, *Route Tracker* alebo *GeoTracker*.

Druhou skupinou sú aplikácie pre spätné zobrazovanie a analýzu GPX súborov. Príkladom sú aplikácie ako *GPX Viewer*, *GPS Essentials* alebo *MAPinr*. Tieto aplikácie vyžadujú internetové pripojenie len pre zobrazenie podkladových máp a slúžia predovšetkým pre rýchlu vizualizáciu trás či zmenu základných metadát daných súborov.

Po rozsiahlom testovaní rôznych aplikácií dostupných na systéme Android som žiadne existujúce riešenie so zameraním na filtrovanie, optimalizáciu a editáciu zaznamenaných trás nenašiel.

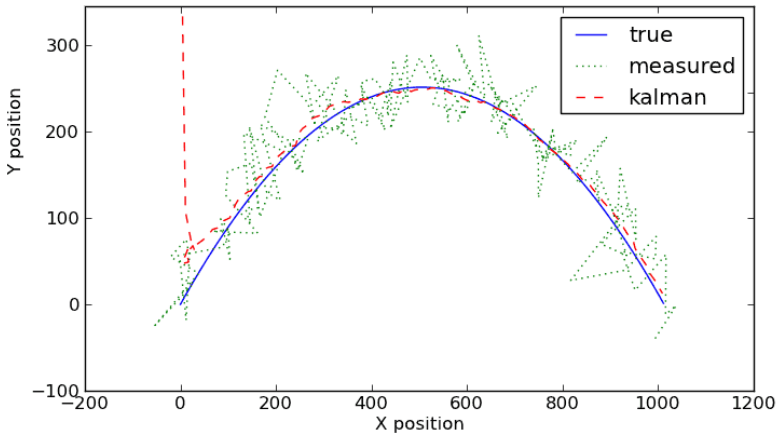
2.4 Vhodné algoritmy

Podľa odporučenia vedúceho práce som si naštudoval základy a posúdil vhodnosť dvoch metód filtrovania bodov.

2.4.1 Kalmanov filter

Jedným z najznámejších a najpoužívanejších nástrojov pre stochastický odhad stavu systému z nepresných meraní je *Kalmanov filter*. Filter nesie meno Rudolfa E. Kálmána, amerického matematika narodeného v Maďarsku, ktorý v roku 1960 publikoval prácu popisujúcu rekurzívne riešenie problému filtrovania diskrétnych údajov. Kalmanov filter je sada matematických rovníc, ktoré vykonávajú odhad a korekciu sledovaných parametrov. Tento filter je *optimálny* v zmysle, že minimalizuje odhadovanú kovarianciu chýb.[12]

Kalmanov filter je diskrétnym filtrom, čo znamená, že sa spolieha na merania získané v pravidelných časový intervaloch. Beh filtra prebieha rekurzívne, čiže odhad stavu budúcnosti sa spolieha na stav v súčasnosti (pozícia, rýchlosť, zrýchlenie atď.) a prípadné informácie o úkonoch, ktoré systém vykonal pre ovplyvnenie stavu (zatáčanie, zmena výkonu motoru atď.). Jeden krok filtra pozostáva z odhadu stavu v budúcnosti, získania nameraných hodnôt, vzájmného porovnania týchto hodnôt a úpravy stavu na ich základe. Čím presnejší matematický model sledovaného systému použijeme, tým viac bude Kalmanov filter konvergovať ku skutočnému priebehu javu popísaného modelom.[5]



Obrázek 2.4: Demonštrácia Kalmanovho filtra na simulovanom príklade vystrelenej delovej gule. Úvodný odhad je zámerne chybný pre zdôraznenie rýchlej konvergencie filtra.

Pred samotným spustením Kalmanovho filtra je potrebné ho vhodným spôsobom inicializovať. Inicializácia Kalmanovho filtra si vyžaduje niekoľko vstupov:

- Matematický popis sledovaného systému reprezentovaný maticami A, B a H.

  A – Stavovo–tranzitívna matica pre získanie predpovede budúceho stavu.

  B – Ovládacia matica obsahujúca popis ovládacích faktorov stavu.

  H – Matica pozorovania.

- Úvodný odhad hodnôt stavu systému, vo forme vektora $\vec{x}$.
- Úvodný odhad hodnôt chyby systému, vo forme matice P.
- Odhady všeobecnej chyby samotného procesu a jeho merania, reprezentované maticami Q a R.

Po spustení filtra je potrebné na začiatku každého kroku dodať dva vstupy. Prvým vstupom je vektor $\vec{u}$ obsahujúci najnovší stav ovládacích prvkov. Jedná sa o hodnoty úkonov ktoré systém vykonal pre ovplyvnenie stavu (napr. miera zatočenia volantom). Druhým vstupom je vektor $\vec{z}$ obsahujúci najnovšie hodnoty pozorovania. Na základe týchto vstupov prebehnú výpočty popísané rovnicami 2.1 až 2.7. Výsledkom týchto výpočtov je nový odhad stavu systému (x_n) a nový odhad celkovej chyby systému (P_n). Nasledujúce rovnice predstavujú krok Kalmanovho filtra.

$$x_{predict} = Ax_{n-1} + Bu_n \tag{2.1}$$
$$P_{predict} = AP_{n-1}A^T + Q \tag{2.2}$$
$$\tilde{y} = z_n - Hx_{predict} \tag{2.3}$$
$$S = HP_{predict}H^T + R \tag{2.4}$$
$$K = P_{predict}H^T S^{-1} \tag{2.5}$$
$$x_n = x_{predict} + K\tilde{y} \tag{2.6}$$
$$P_n = (I - KH)P_{predict} \tag{2.7}$$

Pričom platí,že $x_{predict}$ je predvídaný stav v budúcnosti, $P_{predict}$ je predvídaná hodnota kovariancie, $\tilde{y}$ je hodnota inovácie stavu na základe merania a predpokladaného stavu, S je kovariancia inovácie, K je Kalmanovo zolisnenie, teda miera úpravy stavu na základe inovácie, x_n je nový stav po inovácii a P_n je nová hodnota odhadu kovarianacie.

Pre vstupné hodnoty platí, že $\vec{u}$ je vektor ovládania, ktorý popisuje hodnoty akými bol systém ovplyvnený v danom kroku a $\vec{z}$ je vektor merania, ktorý obsahuje namerané hodnoty daného kroku.
Výstupmi sú x_n a P_n a predstavujú aktualizovaného hodnoty stavu a chyby stavu.
Konštantami vo výpočte sú:

- A – Stavovo–tranzitívna matica – Produktom A a stavu je po pripočítaní ovládacích prvkov nasledujúci stav.
- B – Ovládacia matica – Obsahuje linárne rovnice popisujúce ovládacie faktory stavu.
- H – Matica pozorovania – Po vynásobení stavového vektora touto maticou získame vektor merania.
- Q – Odhad kovariancie chyby procesu
- R – Odhad kovariancie chyby merania

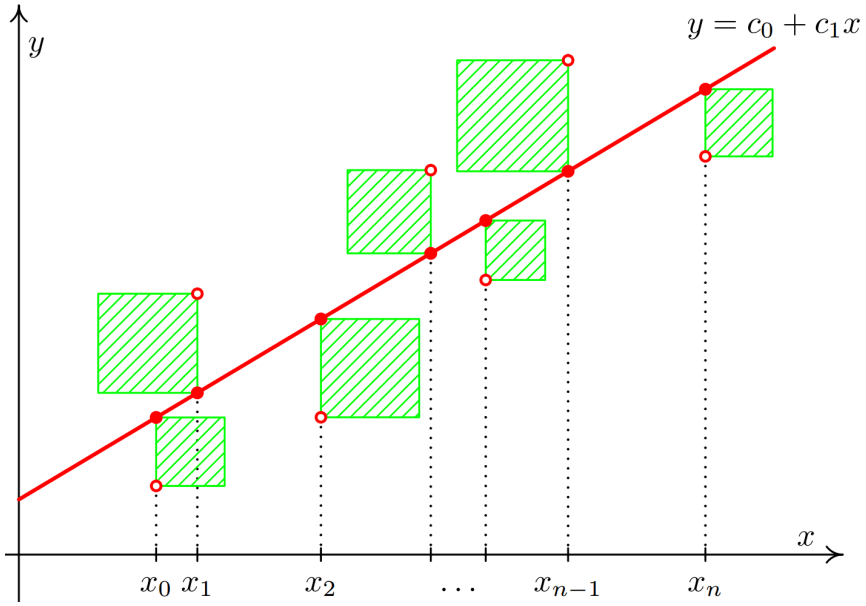
Kalmanov filter sa mi javí ako vhodné riešenie problému filtrovania GPS záznamov. Demonštračné výsledky vyzerajú veľmi uspokojivo a čo sa týka implementácie je tento algoritmus veľmi dobre zdokumentovaný.

2.4.2 Nelineárna metóda najmenších štvorcov

Metóda najmenších štvorcov, anglicky „least squares“, je bežne používaným prístupom k aproximácii riešenia preurčených systémov, teda systémov, kde je počet rovníc vyšší

než počet neznámych. Názov metódy plynie z faktu, že cieľom riešenia tejto metódy je minimalizácia celkového súčtu obsahov štvorcov, ktorých strana predstavuje chybu daného merania. V prípade bežnej metódy najmenších štvorcov je cieľom preloženie nameraných dát priamkou (obrázok 2.5), parabolou alebo všeobecným polynómom predom určeného stupňa. Všetky spomenuté varianty sú prípadom lineárnej regresie.

I keď metóda najmenších štvorcov bola prvý krát popísaná francúzskym matematikom menom Adrien-Marie Legendre v publikácii z roku 1805, zásluhy za jej definíciu sa pripisujú primárne Carlovi Friedrichovi Gaussovi, ktorý vo svojom popise metódy výpočtu obežníc vesmírnych telies z roku 1809 tvrdí, že táto metóda mu bola známa už od roku 1795. Toto viedlo k sporu, avšak Gauss bol schopný dokázať spojitosť tejto metódy s princípmi pravdepodobnosti a normálnej distribúcie.



Obrázek 2.5: Príklad preloženia bodov priamkou pomocou metódy najmenších štvorcov.[7]

Nelineárna varianta tejto metódy, ktorá sa využíva na preloženie sady m pozorovaní matematickým modelom, ktorý je nelineárny v n neznámych parametroch, pričom platí, že $m > n$. Základom tejto metódy je aproximácia cieľa lineárnym modelom. Po získaní úvodnej aproximácie prebehne subsekvetné iteratívne upresňovanie argumentov tohoto modelu. Nosný princíp tejto metódy vychádza z bežnej metódy najmenších štvorcov.

Nelineárna metóda najmenších štvorcov sa mi javí ako menej vhodná metóda pre potreby mojej aplikácie. Informácie o jej implementácii sa vyskytujú na internete pomenej než informácie Kalmanovom filtri a hodlám ju hlbšie preskúmať iba ak narazím na problémy s Kalmanovym filtrovaním. Vedúci práce je navyše členom vývojového tímu, pracujúceho na C++ knižnici zvanej *SLAM++* ktorá túto metódu implementuje a diskutovali sme o využití tejto knižnice v mojej aplikácii v prípade potreby.[15] Pre použitie by však bolo potrebné do aplikácie zaviezť C++ kód, o čom viac píšem v sekcii 2.6.

2.5 Využiteľné API

Pre tvorbu aplikácie pre prácu s GPS trasami, obzvlášť na platforme Android, je vhodné použiť dostupné aplikačné rozhrania (API) od spoločnosti Google. Tri najpoužiteľnejšie API sú nasledovné:

- Maps API – Mapy podkladu a vizualizácia údajov.
- Roads API – Viazanie nameraných bodov k cestám na mape.
- Directions API – Získanie trasy pre rôzne typy dopravných prostriedkov.

2.5.1 Maps API

Google Maps API je ideálne pre zabezpečenie mapovej funkcionality v Android aplikácii. Vďaka tomuto aplikačnému rozhraniu nie je potrebné ručne implementovať vykresľovanie máp pomocou *OpenGL* či riešiť sieťovú komunikáciu spojenú s dynamickým načítaním mapových údajov z internetu. Nevýhodou je nutnosť prítomnosti aplikácií „*Google Maps*“ a „*Google Play services*“ na cieľovom zariadení, avšak obe spomenuté aplikácie sú v dnešnej dobe pribalené na každom zariadení so systémom Android.

2.5.2 Roads API

Google Maps Roads API je relatívne nové internetové aplikačné rozhranie, ktoré bolo sprístupnené verejnosti v roku 2015. Slúži na získanie trasy „pripnutej“ na cesty z nepresnej vstupnej trasy. Vstupom je séria nepresných nameraných údajov, ktoré sa na vzdialenom serveri upnú na najpravdepodobnejšiu trasu na okolitých cestách a odošlú sa späť v odpovedi. Počas upínania je možné vyžiadať interpoláciu bodov, čo zaručí plynulé vzorkovanie zložitých trás, v zmysle obrázku (2.2). Takto interpolovaná odpoveď navyše obsahuje indexy pôvodných bodov, čo umožňuje rozlíšiť tieto body od pôvodných bodov odoslaných na server.[11] Popis konkrétneho využitia v tejto aplikácii sa nachádza v sekcii 3.5.

2.5.3 Directions API

Google Maps Directions API je taktiež internetové aplikačné rozhranie. Slúži pre získanie trasy prechádzajúcej danými bodmi. Na server sa odošle požiadavka so zoznamom priechodzích bodov, typom dopravného prostriedku a prípadnými ďalšími upresňujúcimi voľbami. V odpovedi dostaneme trasu či trasy splňujúce dané parametre.

2.6 Implementačná platforma

Keďže v dobe písania tejto práce vlastním už v poradí tretí mobilný telefón so systémom Android ktorý zároveň používam pre zaznamenávanie GPS aktivít, zvolil som si pre tento projekt práve zmienený systém. Pri implementácii aplikácie pre systém Android sa na prvý vynára nutnosť programovania celého riešenia v programovacom jazyku Java, tak ako to tvorcovia systému zamýšľali. Po krátkom prieskume sa však začnú vynárať ďalšie možnosti, ako napríklad komplexné riešenie *Xamarin*, ktoré slúži na vývoj mobilných aplikácií v jazyku C# vo vývojom prostredí *Microsoft Visual Studio* s možnosťou zostavenia výslednej aplikácie pre všetky tri hlavné mobilné platformy, teda *Android*, *iOS* i *Windows Phone*.[17]

Ďalšou možnosťou je použiť „*Native Development Kit*“ (ďalej len *NDK*), teda sadu nástrojov pre implementáciu zväčša výpočetne náročných algoritmov pomocou jazyka C++, respektíve C. Pomocou NDK je takto implementovaný program prevedený do natívneho kódu, čo umožňuje efektívnejší beh algoritmu. [9]

Mojim plánom pre tento projekt je vyhnúť sa zbytočným komplikáciám a najskôr vyskúšať implementáciu výhradne v jazyku Java. Ostatné možnosti zvážim až po objavení prípadných problémov pri vývoji či pri nadmernej výpočetnej náročnosti použitých algoritmov.

Konkrétne zariadenie na ktorom bude aplikácia implementovaná a testovaná je mobilný telefón *Prestigio Multiphone 5044* (obrázok 2.6) s verziou systému `4.1.2`, respektíve *Android API 17*, teda *Jelly Bean*. Jedná sa o dva roky staré zariadenie, reprezentujúce priemerný mobilný telefón. Pre účely aktuálnosti bude aplikácia taktiež testovaná a zariadení *Lenovo A7000* (obrázok 2.7) so systémom verzie `6.0` (*Android API 23*), zvaným *Marshmallow*.

Obrázek 2.6: Prestigio Multiphone 5044

Obrázek 2.7: Lenovo A7000

Kapitola 3

Návrh aplikácie

V aktuálnej kapitole sa budem venovať popisu návrhu samotnej aplikácie pre operačný systém Android, ktorá je predmetom tejto práce.

3.1 Požiadavky na aplikáciu

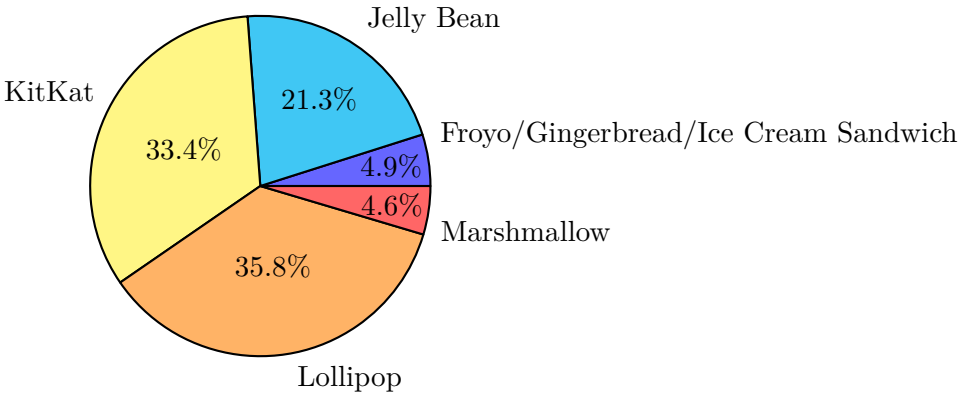
Pri návrhu aplikácie je vhodné najskôr jasne pomenovať jej jednotlivé aspekty. Výsledná aplikácia bude spĺňať nasledovné:

- Aplikácia bude pracovať s aktivitami predom zaznamenanými pomocou GPS.
- Vizualizácia trasy bude používať Google Maps.
- Vstupy i výstupy budú realizované vo formáte GPX. Užívateľ učiní pri otváraní rozhodnutie o prevzorkovaní vstupného súboru.
- Aplikácia bude umožňovať filtrovanie vstupných údajov.
    1. Užívateľ bude schopný ovládať silu filtra.
    2. Užívateľ bude schopný filtrovať celú trasu, i úsek trasy.
    3. Užívať bude mať možnosť definovať úsek trasy.
    4. Filtrovanie bude zabezpečovať Kalmanov filter.
    5. Užívateľ bude mať možnosť upnúť úsek trasy na cestu na mape. Upínanie bude využívať Roads API.
    6. Užívateľ učiní rozhodnutie či výsledky filtrovania zahrnúť do trasy, alebo zahodiť.
- Užívateľ bude mať možnosť definovať bod a manuálne ho presunúť.
    1. Presúvanie ovplyvní okolité body.
    2. Užívateľ bude mať možnosť nastaviť počet bodov ovplyvnených posúvaním.
- Aplikácia bude poskytovať možnosť kroku späť a kroku vpred.

3.1.1 Cieľová verzia Android API

Operačný systém Android je neustále sa vyvíjajúcim systémom. V dôsledku rýchleho vývoja systému Android a rapídne sa zvyšujúceho výkonu mobilných zariadení je reálna distribúcia jednotlivých verzií systému značne fragmentovaná. Mnoho majiteľov mobilných telefónov používa zariadenia so zastaranou verziou systému. Dôvodom tohoto javu je tendencia výrobcov ukončiť podporu zariadení už po relatívne krátkej dobe z technických, ekonomických či marketingových dôvodov.

Spoločnosť Google na svojich stránkach pre Android vývojárov poskytuje údaje (Obrázek 3.1) o percentuálnych podieloch aktívnych verzií systému. Tieto údaje boli získané ku dňu 4.4.2016, pričom najstaršou používanou verziou systému presahujúcou podiel 5% je verzia *Jelly Bean*.[2]



Obrázek 3.1: Distribúcia verzií systému Android ku dňu 4.4.2016

Verzia	**Názov**	**API**	**Distribúcia**
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.2%
4.1.x	Jelly Bean	16	7.8%
4.2.x		17	10.5%
4.3		18	3.0%
4.4	Kitkat	19	33.4%
5.0	Lollipop	21	16.4%
5.1		22	19.4%
6.0	Marshmallow	23	4.6%

Tabulka 3.1: Distribúcia verzií systému Android ku dňu 4.4.2016

Na základe týchto údajov som zvolil ako minimálnu verziu API úroveň 16, teda prvú variantu verzie *Jelly Bean*. Vývoju prospeje i fakt, že môj aktuálny telefón obsahuje Android verzie `4.2.1`, teda API 17. Touto voľbou moja aplikácia, z pohľadu kompatibility, pokrýva 95.1% aktuálne aktívnych zariadení so systémom Android. Naviac na základe faktu, že zvyšných 4.9% aktívnych zariadení používa staršiu verziu systému Android ako je *Jelly Bean* môžeme predpokladať, že ide o dosluhujúce zariadenia s nízkym výkonom.

3.1.2 Vstupy a výstupy

Vstupy i výstupy aplikácie budú realizované vo formáte GPX. Aplikácia bude transformovať vstupný záznam pomocou kombinácie filtra a manuálnych úprav. Spracovaný vstup bude možné uložiť pre ďalšie použitie.

3.1.3 Funkcie

Hlavnou funkciou aplikácie je filtrovanie GPS záznamu. Po načítaní vstupu užívateľ nastaví silu filtra a spustí filtrovanie. Filtrovanie je možné spustiť viac krát nanovo s inou silou. Po vyznačení úseku trasy je možné filtrovať len daný úsek. Taktiež je možné označiť jeden bod, a ten ručne presúvať. Po dosiahnutí uspokojivého výsledku má užívateľ možnosť vyfiltrovanú trasu uložiť do súboru. Funkcie aplikácie sa teda dajú vymedziť nasledovne:

- Načítanie súboru.
    1. Voľba súboru v súborovom systéme.
    2. Rozhodnutie o prevzorkovaní.
- Prevzorkovanie súboru.
- Vizualizácia údajov na mape.
    1. Vizualizácia hlavnej trasy.
    2. Vizualizácia výsledku filtrovania.
    3. Vizualizácia zvoleného úseku.
    4. Vizualizácia manuálne určených bodov.
- Filtrovanie trasy.
    1. Voľba úseku trasy.
        (a) Umiestnenie bodu na vybrané miesto na trase.
    2. Voľba metódy filtrovania.
        (a) Kalmanov filter.
            i. Voľba sily filtra
            ii. Spustenie filtra.
        (b) Upínanie úseku trasy na cestu na mape.
            i. Voľba úseku.
            ii. Spustenie filtra.
    3. Integrácia výsledku filtrovania do hlavnej trasy.
- Manuálna korekcia trasy.
    1. Voľba bodu korekcie.
    2. Nastavenie počtu okolitých bodov ktoré budú ovplyvnené.
    3. Voľba konečného miesta daného bodu.
    4. Plynulé posunutie okolitých bodov v danom rozmedzí.
- Krok späť a krok v pred.
- Uloženie hlavnej trasy do súboru.

3.2 Spracovanie vstupov a výstupov

3.2.1 Prístup k súborom

Pre potreby tejto práce predpokladám, že užívateľ aplikácie má pripravené zaznamenané súbory vo formáte GPX na karte SD, prípadne v pamäti telefónu. Keďže Android API úrovne 16 neposkytuje štandardné riešenie pre dialóg voľby súboru, najjednoduchším riešením bude využiť triedu `Intent`. Táto trieda, ktorá je súčasťou Android API, umožňuje definovať úlohu, na ktorej splnenie je možné použiť inú nainštalovanú aplikáciu. Po vyvolaní *intentu* získania obsahu systém skontroluje dostupnosť aplikácií, ktoré dokážu túto úlohu splniť. Ak je vyhovujúcich aplikácií viac, užívateľ zvolí jednu z nich.

V prípade, že v systéme nie je vhodná aplikácia pre prácu so súbormi, použije sa jednoduchý vstavaný dialóg výberu súbor. Pre tieto účely bude potrebné túto dialóg implementovať, alebo nájsť vhodnú knižnicu. Po krátkom hľadaní som narazil na knižnicu „FileChooser“. [13] Táto malá knižnica je voľne dostupná a perfektne spĺňa úlohu záložného spôsobu voľby súboru. Vďaka kombinácii využitia *intentov* a vstavaného súborového manažéra ako alternatívy, bude zabezpečená funkčnosť v prípade absencie externých súborových manažérov, ale i zaručená možnosť použiť súborový manažér, na ktorý je užívateľ zvyknutý.

3.2.2 Spracovanie formátu GPX

Keďže formát GPX vychádza z formátu XML, bolo by možné jeho spracovanie implementovať pomocou bežne používanej triedy *XmlPullParser*, ktorá je súčasťou platformy Android. Na internete však existuje Java knižnica s názvom „GPXParser“, licencovaná pod LGPL, ktorá implementuje parsovanie formátu GPX kompletne podľa jeho popisu, vrátane všetkých jeho záľudností. Knižnica taktiež poskytuje základné objekty vyplývajúce z formátu GPX, akými sú trasa, bod trasy, metadáta súboru či GPX súbor samotný.[1] Bude teda vhodné túto knižnicu využiť nielen na spracovanie formátu GPX, ale i pre vnútornú reprezentáciu spracovávaných údajov.

3.3 Vizualizácia údajov na mape

Jednoznačne najvhodnejším spôsobom prezentácie GPS záznamov je vizualizácia na mape pomocou spojníc bodov. Pochopenie takto prezentovaných údajov je pre užívateľa jednoduché, intuitívne.

3.3.1 Návrh využitia Maps API

Najjednoduchším a pravdepodobne i najlepším dostupným spôsobom ako zaistiť mapovú funkcionalitu v aplikácii na systéme Android je použitie *Google Maps API*, popísaného v sekcii 2.5.1. *Maps API* poskytuje všetky potrebné funkciem, ktoré aplikácia popísaná v tejto práci využíva. Okrem zobrazenia samotnej podkladovej mapy poskytuje i možnosť umiestniť na mapu užívateľské body, tzv. „Markers“, a čiary zostavené z viacerých bodov, zvané „Polylines“. Možnosť umiestnenia *markerov* bude využitá pri vyznačovaní úseku trasy, prípadne voľbe bodu manuálnej korekcie. Polyline zasa bude použitý pre vizualizáciu trás na mape.

3.3.2 Výpočet dĺžky trasy

Súčasťou aplikácie bude i indikátor celkovej dĺžky nefiltrovanej trasy a trasy filtrovanej. Užívateľ tak jasne uvidí vzdialenostný rozdiel, ktorý vzniká v dôsledku filtrovania. Pre účely tejto funkcie je potrebné zaručiť spoľahlivý spôsob určenia vzdialenosti medzi dvoma bodmi.

Výpočet dĺžky trasy prebieha iteratívne. Výsledná dĺžka je súčtom vzdialeností jednotlivých po sebe idúcich bodov. Pre získanie tejto vzdialenosti existuje metóda `distanceTo` triedy `Location`, ktorá je súčasťou Android API. Počas experimentovania s touto metódou som však narazil na problémy s presnosťou výsledkov. Tomuto problému som venoval niekoľko dní a prišiel som k záveru, že bude spoľahlivejšie výpočet vzdialenosti medzi dvoma súradnicami implementovať ručne.

Vo svete navigácie sa často používa „haversin“. Je to trigonometrická funkcia, ktorá je polovicou „versinu“, teda rozdielu jednotky a kosínu. Prvé tabuľky s hodnotami *haversinu* boli publikované už v roku 1805[3], avšak samotný výraz „*haversine*“ bol zavedený až v roku 1835 Jamesom Inmanom, pre navigačné potreby britských námorníkov.[14] Haversin umožňuje výpočet vzdialenosti ľubovoľných dvoch bodov na povrchu gule, pomocou nasledujúcej rovnice:

$$a = sin^2\left(\frac{\Delta\varphi}{2}\right) + cos\varphi_1 \cdot cos\varphi_2 \cdot sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{3.1}$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{1-a}) \tag{3.2}$$

$$d = R \cdot c \tag{3.3}$$

Pričom platí, že φ je zemepisná šírka, λ je zemepisná dĺžka, R je polomer gule, d je výsledná vzdialenosť a všetky uhly sú v radiánoch.[16]

3.4 Kalmanov filter

Po preskúmaní náročnosti, efektivity a dostupnosti materiálov jednotlivých metód som sa rozhodol, že v aplikácii bude filtrovaciu funkcionalitu zabezpečovať Kalmanov filter popísaný v sekcii 2.4.1.

3.4.1 Knižnica JKalman

Pri prvých pokusoch o implementáciu Kalmanovho filtra v jazyku Java som narazil na knižnicu *JKalman*. Autorom knižnice je Ing. Petr Chmelař z UIFS FIT VUT. Jedná sa o knižnicu, ktorá zapuzdruje stav Kalmanovho filtra a poskytuje základné metódy pre inicializáciu filtra a uskutočnenie odhadu i korekcie.[4] Použitie knižnice mi umožní zamerať sa na samotné vstupy a model problému, bez nutnosti implementácie jednotlivých medzikrokov rovníc.

3.4.2 Problém vzorkovania

Keďže Kalmanov filter je diskrétny algoritmus, vstupné dáta by mali byť vzorkované v pravidelných, fixných intervaloch. GPS záznamy vytvorené na mobilných zariadeniach, na ktorých tento projekt vyvíjam, sú vzorkované približne po jednej sekunde. Toto vzorkovanie však nieje konštantné a v dôsledku javov, popísaných v sekcii 2.1, sa môže časová vzdialenosť dvoch bodov v sekvencii líšiť v extrémnych prípadoch i o desiatky sekúnd. Pre tieto účely by bolo vhodné vstupné dáta prevzorkovať doplnením priemerovaných bodov, medzi po sebe idúce body, ktorých časový rozdiel je väčší ako jedna sekunda.

Prevzorkovanie však prináša mnohé nevýhody. Týmito nevýhodami sú napríklad zvýšená réžia pri otváraní súboru, keďže je nutné vypočítať veľké množstvo „falošných“ bodov. Ďalším problémom je fakt, že ak bolo zaznamenávanie pozastavené a opätovne spustené, vytvorí sa obrovský počet zbytočných bodov. Takmer všetky záznamové aplikácie možnosť pozastavenia poskytujú a mnohé z nich, ako napríklad *Strava*, pozastavujú niektoré typy aktivít pri detekcii zastavenia automaticky. Detekcia týchto zastavení vo vstupnom zázname by si vyžadovala inteligentné stanovene časovej hranice, po ktorej prekročení by sa chýbajúce vzorky brali ako prerušenie a neboli by doplňované. Za týmto účelom by bolo potrebné zaviesť ďalší ovládací prvok do užívateľského rozhrania a Kalmanov filter by bolo následne nutné púšťať viacnásobne nad jednotlivými úsekmi.

Druhou možnosťou je prevzorkovať vstup podľa najväčšieho rozdielu vzoriek, a teda zahodiť veľký počet bodov v mene pravidelného filtrovania. Tento postup by síce urýchlil filtrovanie i celkový chod aplikácie, avšak presnosť výsledku, obzvlášť pri filtrovaní komplexných trás, by bola nenávratne znížená.

Treťou, dokázateľne menej korektnou avšak oveľa efektívnejšou možnosťou, je „prižmúriť oči“ a prehlásiť vstupné body filtru za pravidelne vzorkované a prípadné nezrovnalosti vo filtrovanej trase korigovať manuálne. Táto možnosť je prípustná, pretože nezrovnalosti vo vzorkovaní v bežnom súbore z pravidla nebývajú príliš veľké a pri spracovaní záznamu s dlhými zastaveniami sa záznam nenafúkne o tisíce nepotrebných bodov alebo naopak, pri dlhej pauze sa nezahodí veľa bodov. Chýbajúce vzorky by sa však stali neprehliadnuteľným problémom v prípade že trasa obsahuje veľmi veľké skoky, ktoré by užívateľ chcel manuálne upraviť.

V prospech tretej možnosti hrá i fakt, že užívateľ bude schopný riadiť silu filtra, a teda vyberie vždy najvhodnejšiu silu, podľa vizualizovaných výsledkov filtrovania. Taktiež bude mať k dispozícii nástroje pre filtrovanie manuálne zvoleného úseku trasy či korekciu konkrétneho bodu trasy. Ako riešenie som teda zvolil kompromis medzi prvým a tretím popísaným postupom a to vo forme dialógu, ktorý sa užívateľa pri otvorení súboru spýta či sa má súbor prevzorkovať. Užívateľ teda môže pri relatívne bezproblémových záznamoch vzorkovanie preskočiť, ale má k dispozícii i kvalitnejšiu a korektnejšiu možnosť na úkor rýchlosti spracovania a veľkosti výsledného súboru.

3.4.3 Model pohybu

Pre použitie Kalmanovho filtra je potrebné formulovať model pohybu v podobe matice. Keďže je mojim cieľom aby aplikácia bola schopná filtrovať ľubovoľný typ GPS záznamu či už športovej aktivity alebo jazdy motorového vozidla, musí byť model dostatočne všeobecný. Filtrovanie by malo byť schopné produkovať uspokojivé, predvídateľné a opakovateľné výsledky nezávisle od druhu záznamu. Aplikácia musí byť taktiež responzívna, teda filtrovanie trasy by nemalo trvať dlhšie ako niekoľko sekúnd. Model pohybu Kalmanovho filtra teda nesmie byť výpočetne náročný a spracovanie priemernej trasy by malo byť v ideálnom prípade

zdanlivo okamžité.

Vstupné údaje sú zaznamenané vo forme zemepisnej šírky a dĺžky, čo by mohlo na prvý pohľad viesť k predpokladu, že ide o dvojrozmerné súradnice. Opak je však pravdou a korektná práca so zemepisnými údajmi si vyžaduje pokročilejšie výpočty, ktorým sa v mene výpočetnej náročnosti chcem vyhnúť. Pre potreby implementovanej aplikácie môžeme teda problém zjednodušiť a naozaj pracovať so vstupmi ako so súradnicami x a y v dvojrozmernom priestore. Keďže aplikácia je určená na filtrovanie každodenných záznamov bežných športových aktivít, skreslenie je zanedbateľné. Rýchlosť bude v modeli pre jednoduchosť reprezentovaná ako Δx a Δy medzi dvoma po sebe idúcimi vzorkami. Z týchto dôvodov som sa rozhodol použiť jednoduchú sústavu rovníc pre výpočet polohy v dvojrozmernom priestore:

$$x(t) = x_0 + V_{0x}t \tag{3.4}$$
$$V_x(t) = V_{0x} \tag{3.5}$$
$$y(t) = y_0 + V_{0y}t \tag{3.6}$$
$$V_y(t) = V_{0y} \tag{3.7}$$

Tieto rovnice môžeme prepísať rekurentne, kde Δt bude časový krok vzorkovania.

$$x_n = x_{n-1} + V_{xn-1}\Delta t \tag{3.8}$$
$$V_{xn} = V_{xn-1} \tag{3.9}$$
$$y_n = y_{n-1} + V_{yn-1}\Delta t \tag{3.10}$$
$$V_{yn} = V_{yn-1} \tag{3.11}$$
$$\tag{3.12}$$

Ak rovnice ďalej prepíšeme do tvaru matice, začnú sa podobať na rovnicu 2.1 popísanú v sekcii 2.4.1.

$$\begin{bmatrix} x_n \\ V_{xn} \\ y_n \\ V_{yn} \end{bmatrix} = \begin{bmatrix} x_{n-1} & + & V_{xn-1}\Delta t & & \\ & & V_{xn-1} & & \\ & & & y_{n-1} + V_{yn-1}\Delta t & \\ & & & & V_{yn-1} \end{bmatrix} \tag{3.13}$$

Po následnom odstránení premenných získame maticu A, popísanú v sekcii 2.4.1, ktorá je potrebná pre získanie predikcie nasledujúceho stavu, zo stavu súčasného. (x_n v tomto prípade už predstavuje stav vo vzorku n a nie samotnú súradnicu x). Výsledná rovnica neobsahuje maticu ovládania, pretože v tomto prípade máme k dispozícii len dve súradnice a čas, teda žiadne informácie o zatáčaní či zrýchlení.

$$x_n = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} x_{n-1} \tag{3.14}$$

3.4.4 Úvodný odhad

Nasledujúcim nevyhnutným krokom pri inicializácii Kalmanovho filtra je nastavenie úvodného odhadu. Jedná sa o maticu, ktorá obsahuje približné hodnoty všetkých premenných stavu systému. Úvodný odhad nemusí byť príliš presný, pretože Kalmanov filter už po niekoľkých iteráciách nadobudne uspokojivé hodnoty.

V našom prípade bude voľba úvodných hodnôt celkom jednoduchá. Ako úvodné súradnice x a y zvolím zemepisnú šírku φ a dĺžku λ prvého bodu záznamu, pričom rýchlosti Δx a Δy nastavím na hodnotu 0.

3.4.5 Spracovanie meraní

Filtrovaná trasa sa pre potreby filtra skladá z jednotlivých meraní. Meraním rozumieme štvoricu x, Δx, y a Δy. V kroku filtra sa tieto hodnoty predajú filtru vo forme vektora, pričom budú použité na korekciu odhadnutého stavu. Hodnoty x a y budú predávané do filtra priamo zo vstupných záznamov. Hodnoty Δx a Δy bude potrebné v každom kroku vypočítať. Poslúži k tomu nasledujúci vzťah:

$$\Delta x = \frac{x_n - x_{n-1}}{\Delta t} \tag{3.15}$$

3.4.6 Kovariancia chyby merania

Matica kovariancie chýb merania je hlavným prvkom ovplyvňujúcim vnímanú silu filtrovania. Ak nastavíme filtru vyššiu hodnotu kovariancie, efektívne tým hovoríme, že danej premennej veríme menej a výstupy by mali byť bližšie k hodnotám odhadu ako k hodnotám merania. Naopak nulová hodnota kovariance spôsobí, že meraniu veríme absolútne a výsledná trasa bude identická so vstupom.

V aplikácii všetky namerané hodnoty pochádzajú z jedného zdroja a preto budú mať spoločnú hodnotu kovariancie. Užívateľovi bude táto hodnota sprístupnená pomocou posuvníka, prezentovaného ako „sila filtra“. V tejto chvíli máme definované a pripravené všetky vstupy Kalmanovho filtra potrebné pre implementáciu aplikácie.

3.5 Využitie Roads API

Rozhranie *Google Maps Roads API*, popísané v sekcii 2.5.2, je možné využiť pre zarovnanie nameraných bodov na podľa okolitých ciest. Rozhranie predpokladá, že vstupné body sú nepresné merania, tak ako v našom prípade.

3.5.1 Popis všeobecného postupu použitia

Predpokladom práce s *Roads API* je API kľúč. Tento kľúč je vyžiadateľný na vývojárskych stránkach spoločnosti Google. Po získaní kľúča je možné zasielať požiadavky na servery *Roads API*. Aplikácia musí pripraviť požiadavku so zoznamom bodov určených na umiestnenie na cesty. Požiadavka a odpoveď sú prenášané medzi serverom a zariadením pomocou metódy `RoadsApi.snapToRoads`. Odpoveď zo servera obsahuje zoznam bodov triedy `SnappedPoint`. Každá položka zoznamu obsahuje bod umiestnený na cestu, pričom zahŕňa i pôvodný index bodu v odoslanej postupnosti.

Práve vďaka prítomnosti pôvodného indexu je možné v aplikácii následne jasne určiť odpovedajúci bod pôvodného záznamu, čo umožňuje zachovať časové údaje bodov. *Roads API* taktiež umožňuje výslednú trasu interpolovať. Interpolácia síce zvýši zdanlivú presnosť trasy kopírujúcej cesty pridaním dodatočných bodov, avšak túto možnosť používať nebudem, pretože interpolované body neobsahujú časové údaje a nezodpovedajú pôvodným bodom trasy. Využitie interpolácie bez dodatočného priemerovania časov by viedlo k nežiadúcim časovým skokom v zázname.

Samotné žiadosti odosielané na server musia spĺňať isté predpoklady. Na využitie bezplatnej varianty *Roads API* sa vzťahujú následujúce obmedzenia[10]:

- Maximálny počet 2500 žiadostí za deň.
- Maximálny počet 10 žiadostí za sekundu.
- Maximálny počet 100 bodov na žiadosť.

Prvé obmedzenie by pre demonštračnú aplikáciu nemalo byť problémom, keďže 2500 žiadostí za deň predstavuje až 250 000 bodov, čo pri vzorkovaní jeden krát za sekundu predstavuje takmer sedemdesiat hodín alebo necelých 700 kilometrov záznamu pri priemernej rýchlosti 10 kmh^{-1}.

Druhé obmedzenie, teda 10 žiadostí za sekundu, taktiež nepredstavuje pri mobilnej aplikácii problém, keďže spracovanie bodov trvá istý čas. Pre korektnosť riešenia však tento limit bude zohľadnený vo finálnej aplikácii.

Tretie obmedzenie si vyžaduje zvýšenú pozornosť, keďže v aplikácii filtrovať budeme i úseky presahujúce dĺžku 100 bodov. Riešenie je však jednoduché; stačí rozdeliť daný úsek na menšie úseky o maximálnej dĺžke 100 bodov a po prijatí odpovede ich zasa spojiť. Vývojári projektu odporúčajú pri takomto skladaní trás odosielať aj istý počet prelínajúcich sa bodov na začiatku a konci úseku, pre lepšie výsledky algoritmu.[8]

3.5.2 Návrh použitia v aplikácii

V aplikácii bude *Roads API* použité pre manuálne filtrovanie lokálnych častí trasy. Užívateľovi bude sprístupnené tlačidlo, po ktorého stlačení bude aktuálne zvolený úsek trasy upnutý na okolité cesty. Upínanie na cesty je vhodné len na niektorých úsekoch, keďže zaznamenaná aktivita mohla prebiehať i mimo známych ciest či v nesúlade s okolitými cestami. Ďalším dôvodom prečo nie je vždy vhodné využiť upínanie je fakt, že *Roads API* v súčasnej podobe neumožňuje určiť typ dopravného prostriedku a teda upína údaje na cesty v súlade s pravidlami cestnej premávky. To znamená, že pri trase smerujúcej po chodníkoch bez prístupu áut či v opačnom smere jednosmerných ciest, nebude možné trasu upnúť.

Ako vhodný príklad reálneho a efektívneho využitia v aplikácii nám môže poslúžiť záznam aktivity v mestskom prostredí kopírujúci cestu, ktorý však v dôsledku nízkej kvality

záznamu prebieha zo strany na stranu. V tomto prípade je v porovnaní s filtrovaním pomocou Kalmanovho filtra efektívnejšie daný úsek pripnúť na blízku cestu.

Postup použitia v aplikácii bude nasledovný:

1. Užívateľ zvolí filtrovaný úsek.
2. Body zahrnuté v úseku sa rozdelia na menšie úseky podľa obmedzení API.
3. Na server sa postupne posielajú jednotlivé časti.
4. Po prijatí všetkých častí zo strany servera sa opätovne poskladá filtrovaný úsek.
5. Úsek v pôvodnej trase nahradíme výsledkom.

3.6 Využitie Directions API

Rozhranie *Google Maps Directions API*, popísané v sekcii 2.5.3 je možné do istej miery použiť tam, kde *Roads API* zlyháva, teda pri upínaní na rôzne chodníky či vedľajšie cesty.

3.6.1 Popis všeobecného postupu použitia

Directions API funguje na podobnom princíme ako *Roads API*. Pre využitie plného potenciálu bezplatnej verzie je potrebné získať API kľúč. Je však možné použiť existujúci kľúč z *Roads API*. Aplikácia využívajúca toto API odošle požiadavku vo forme zoznamu bodov a rôznych upresňujúcich parametrov, akými sú napríklad typ dopravného prostriedku či čas odchodu a server odpovie trasou ktorá cez tieto body prechádza a zároveň spĺňa všetky voliteľné parametre. Na požiadavky sa vzťahujú nasledujúce obmedzenia:

- Maximálny počet 2500 žiadostí za deň.
- Maximálny počet 10 žiadostí za sekundu.
- Maximálny počet 8 bodov na žiadosť bez API kľúča.
- Maximálny počet 23 bodov na žiadosť s API kľúčom.

Na prvú nevýhodu v porovnaní s *Roads API* narážame už pri maximálnom počte bodov v žiadosti. API má síce možnosť pracovať bez kľúča, avšak počet spracovaných bodov je drasticky obmedzený. Ak by sme chceli vyjadriť denné limity rovnakým spôsobom ako v popise predchádzajúceho API, tak sa pri ôsmich bodoch na žiadosť dostaneme na hodnoty takmer 5.5 hodín záznamu pri vzorkovaní po jednej sekunde, alebo 55 kilometrov pri priemernej rýchlosti 10 kmh^{-1}. Tieto hodnoty sú príliš nízke pre bežné použitie a preto musíme použiť variantu s kľúčom. Použitie kľúča umožňuje odosielať takmer tri krát viac bodov, avšak stále zaostáva za *Directions API*, ktoré umožňuje odosielať viac ako štvornásobne väčší počet bodov. S kľúčom sa teda dostávame k hodnotám takmer 16 hodín, alebo 160 kilometrov.

Obmedzenia tohoto API sú striktnejšie z jednoduchého dôvodu, ktorým je fakt, že API je určené pre získanie trasy *prechádzajúcej* danými bodmi, v porovnaní s *Roads API*, ktoré je určené priamo na upínanie bodov. Neznamená to však, že *Direction API* sa nedá použiť na upínanie bodov. Tu nastáva problém.

3.6.2 Návrh použitia v aplikácii

Keďže sa jedná o službu pre plánovanie trás *prechádzajúcich* danými bodmi, každý bod odoslaný na server sa z definície musí vyskytnúť vo výslednej trase. Nie je teda možné odosielať na server všetky body daného úseku, pretože výsledkom by bola trasa obsahujúca kombináciu pôvodných bodov a trasy ktorá ich spája.

V rámci experimentovania s týmto API som skúsil odosielať len každý n-tý bod, pričom som testoval rôzne hodnoty n. Výsledky boli uspokojivé, ak odosielané body ležali priamo na ceste či chodníku, alebo neďaleko od nich. Ak však odosielaný bod ležal pri križovatke a bol bližšie ku križujúcej ceste ako ku skutočnej, výsledná trasa odbočila do danej križovatky a prešla úsek po nasledujúcu križovatku po úplne inej ulici.

Ďalším problémom je, že body ktoré prídu ako odpoveď neobsahujú časové údaje, keďže žiadame server len o trasu. Priama integrácia takýchto bodov do pôvodnej trasy by znehodnotila časové údaje záznamu.

Riešením týchto problémov by mohol byť nasledujúci postup:

1. Užívateľ zvolí filtrovaný úsek.
2. Na server sa odošle žiadosť obsahujúca iba hraničné body úseku a typ aktivity.
3. Vhodným spôsobom zaručíme aby počet bodov v odpovedi zodpovedal počtu pôvodných bodov úseku.
   (a) V prípade že je bodov v odpovedi menej, pridáme do nej ďalšie priemerované body.
   (b) Ak je naopak bodov v odpovedi viac, nejaké z nich zahodíme.
4. Priradíme časové údaje z pôvodných bodov do zodpovedajúcich bodov odpovede v pomere $1 : 1$.
5. Úsek v pôvodnej trase nahradíme výsledkom.

Toto riešenie zaručuje len to, že výsledné body získané z odpovede budú obsahovať *nejaké* časové údaje. Pri rozvážnom použití na vhodnom úseku by však časové hodnoty pôvodných bodov mali byť dostačujúce.

3.7 Návrh užívateľského rozhrania

Keďže sa primárne jedná o technickú aplikáciu s jediným účelom, užívateľské rozhranie tomu bude zodpovedať. Bude jednoduché, ideálne s jedinou obrazovkou na ktorej umožní prístup ku všetkým funkciám aplikácie. Rozhranie by však malo byť použiteľné i bežným užívateľom bez potreby vysvetľovania ovládania.

3.7.1 Požiadavky na užívateľské rozhranie

Základné požiadavky na užívateľské rozhranie implementovanej aplikácie by sa dali vymedziť nasledovne:

- Musí umožňovať jednoduchý prístup k vstupným súborom.
- Musí jasným spôsobom vizualizovať trasu na mape.

- Musí umožňovať jednoduchý prístup k všetkým funkciám.
- Musí byť responzívne, užívateľovi musí byť jasné čo sa deje.

3.7.2 Mockup užívateľského rozhrania

Pre potreby popisu návrhu rozhrania aplikácie je vhodné vytvoriť jeho vizualizáciu, teda *Mockup*. Hlavné prvky aplikácie vyobrazené na obrázku 3.2 sú popísané v nasledujúcich štrnástich bodoch:

1. Mapa s vizualizáciou pôvodnej a filtrovanej trasy.
2. Tlačidlo načítania vstupného GPX súboru.
3. Tlačidlo uloženia výstupného GPX súboru.
4. Posuvník pre nastavenie sily filtrovania.
5. Posuvník pre nastavenie počtu bodov ovplyvnených manuálnym posunom bodu.
6. Výpis dĺžky nefiltrovanej a filtrovanej trasy.
7. Rozbaľovací zoznam pre voľbu typu mapy.
8. Tlačidlo zahodenia pripnutia upravovaného úseku na cesty.
9. Tlačidlo kroku späť.
10. Tlačidlo kroku vpred.
11. Dialóg informujúci o priebehu aktuálnej úlohy.
12. Tlačidlo zahodenia filtrovaných výsledkov.
13. Tlačidlo pripojenia filtrovaných výsledkov do trasy.
14. Bod (Marker) manuálnej úpravy.

Obrázek 3.2: Mockup užívateľského rozhrania

3.7.3 Popis správania prvkov užívateľského rozhrania

- Tlačidlá sú stlačiteľné iba ak to pri aktuálnom stave aplikácie dáva zmysel.
- Po stlačení tlačidiel 2 a 3:
  1. Vyvolá sa intent pre výber súboru externou aplikáciou.
  2. V prípade absencie externej aplikácie sa použije vstavaný súborový dialóg popísaný v sekcii 3.2.1.
  3. Spustí sa zodpovedajúca úloha.
- Po zmene hodnoty posuvníka 4 sa spustí úloha filtrovania nad vyznačeným úsekom trasy. Ak nie je vyznačený žiadny úsek, trasa sa filtruje celá.

- Po kliknutí do mapy 1:
  (a) Ak mapa neobsahuje žiaden bod 14 a zároveň prechádza kliknutým miestom trasa, umiestni sa v tomto mieste bod.
  (b) Ak mapa obsahuje práve jeden bod:
    1. Ak prechádza kliknutým miestom trasa, umiestni sa v tomto mieste bod.
    2. Vyznačí sa úsek medzi oboma bodmi.
    3. Sprístupní sa tlačidlo upínania na cesty.
  (c) Ak kliknutým miestom neprechádza trasa alebo mapa obsahuje práve dva body, odstránia sa všetky body.
- Po dlhom podržaní a ťahaní bodu 14 sa zmení jeho poloha. Počet okolitých bodov, ktoré sú ovplyvnené zodpovedá hodnote posuvníka 5.
- Po spustení úloh ako filtrovanie či práca so súbormi sa zobrazí dialóg 11. Dialóg sa dá zrušiť stlačením tlačidla späť, ale len v prípade, že sa jedná o sieťovú úlohu.
- Po skončení filtrovania, presúvania či upínania sa zobrazí výsledok modrou farbou a zároveň sa zobrazí spodný panel s tlačidlami 12 a 13, ktorý umožní užívateľovi zmenu zachovať alebo zahodiť.
- Po stlačení tlačidla 8 sa spustí úloha upínania úseku na cestu.
- Po stlačení tlačidla 9 sa zvráti posledná aplikovaná zmena v zázname.
- Po stlačení tlačidla 10 sa opätovne vykoná posledná zvrátená zmena.
- Po stlačení ponuky 7 sa užívateľovi sprístupní voľba typu mapy.

3.7.4 Ikony užívateľského rozhrania

Pre zvýšenie intuitívnosti a odľahčenie objemu text v užívateľskom rozhraní som sa rozhodol väčšinu tlačidiel reprezentovať relevantnými ikonami. Napríklad na tlačidlo pre otvorenie súboru je reprezentované ikonou priečinka, tlačidlo pre uloženie disketou a tak ďalej. Za účelom efektívneho využitia zdrojov sú tieto ikony uložené v špeciálnom, voľne distribuovateľnom písme zvanom *Font Awesome*.[6] Toto písmo je bude zdrojových súborov aplikácie.

Obrázek 3.3: Ukážka niekoľkých znakov písma *Font Awesome*, ktoré budú použité ako ikony tlačidiel v užívateľskom rozhraní aplikácie

Kapitola 4

Implementácia

Táto kapitola popisuje z môjho pohľadu zaujímavé implementačné detaily a zmeny či rozhodnutia učinené počas implementácie aplikácie.

4.1 Finálna platforma a nástroje

Mojim plánom načrtnutým v kapitole 2 bolo pokúsiť sa o implementáciu výhradne pomocou jazyka Java. Počas implementácie som nenarazil na žiadne problémy týkajúce sa platformy a podarilo sa mi dosiahnuť požadované výsledky bez nutnosti využitia alternatívnych riešení popísaných v sekcii 2.6. Aplikáciu som úspešne implementoval v oficiálnom integrovanom vývojovom prostredí *Android Studio 2.0* za pomoci knižníc popísaných v kapitole 3.

4.2 Zmeny v návrhu

Počas implementácie som bol nútený vykonať niekoľko menších zmien návrhu riešenia problému i užívateľského rozhrania. Najpodstatnejšou zmenou je vypustenie využitia *Google Directions API*, popísaného v sekcii 3.6. Dôvodom bolo, že *Google Roads API* splňovalo účel upínania bodov na cestu dostatočne dobre a adaptácia *Directions API* pre potreby upínania by si vyžadovala nadmerné úsilie, keďže oficiálne slúži mierne odlišnej úlohe plánovania trasy.

Zmeny užívateľského rozhrania sú vylepšením návrhu. Majú najmä podobu skrývania ovládacích prvkov, ktorých použitie v daný moment nedáva zmysel. Príkladom je skrývanie panela obsahujúceho tlačidlá pre zahodenie, alebo integráciu výsledku filtrovania do hlavnej trasy v situáciách keď filtrovanie neprebieha. Druhým príkladom je skrytie posuvníka pre nastavenie počtu bodov ovplyvnených posunom v prípade, že posun práve neprebieha.

4.3 Vybrané detaily implementácie

4.3.1 Implementácia úloh

Počas implementácie som si všimol, že v požiadavkách na funkcionalitu sa niekoľko krát vyskytujú úlohy, počas ktorých užívateľ nemôže vykonávať žiadne iné úkony a mal by byť informovaný o ich priebehu. Konrkétnymi úlohami sú:

- Otváranie súboru a spracovanie vstupu.

- Beh Kalmanovho filtra.
- Komunikácia so serverom *Roads API*.
- Ukladanie výsledkov do súboru.

Všetky tieto úlohy naviac musia prebiehať asynchrónne, pretože sa jedná o časovo náročné úkony. Ak by boli vykonávané v hlavnom vlákne, aplikácia by po dobu behu úlohy „zamrzla“ a systém Android by ju dokonca mohol ukončiť. So zavedením asynchrónnosti je zároveň potrebné vyriešiť predávanie výsledkov z asynchrónnych úloh.

Mojim riešením tohoto problému je rozhranie `IAsyncResponse`. Hlavná aktivita implementuje toto rozhranie, ktoré vyžaduje definíciu metód pre spracovanie výsledkov, ktoré budú volané ako delegáti z jednotlivých úloh. Každá asynchrónna úloha je následne obsiahnutá vo vlastnej triede, ktorá rozširuje triedu `AsyncTask`.

4.3.2 Implementácia úlohy Kalmanovho filtrovania

Pre implementáciu Kalmanovho filtrovania bolo najskôr potrebné vytvoriť triedu obsahujúcu stav filtra triedy `JKalman` a zároveň konkrétny matematický model popísaný v sekcii 3.4. Výsledkom je trieda `KalmanFilter`, ktorá obsahuje inštanciu triedy `JKalman` a poskytuje metódy pre inicializáciu filtra a vykonanie kroku filtra. Inicializácia prebieha zadaním prvotného odhadu a sily filtra, zatiaľ čo metóda `step` na základe zadaného merania pozície a rýchlosti vykoná krok filtra a vráti nové súradnice.

Samootná úloha kalmanovho filtrovania následne inicializuje filter prvým bodom úseku a následne iteratívne volá metódu `step` nad každým bodom trasy. V každom kroku sa zo súradníc naviac vypočítavajú rýchlosti Δx a Δy. Výsledok kroku filtra sa ukladá ako bod do novej trasy, pričom mu je priradená časová hodnota body z ktorého filtrovaním vznikol. Nová trasa je následne po skončení úlohy vrátená do hlavnej aktivity aplikácie.

Špeciálnym prípadom tejto úlohy je filtrovanie úseku, kde sa vyžaduje aby prvý i posledný bod zostali nezmenené, aby sa zaručila možnosť nadpojenia na pôvodnú trasu. Riešením tohoto problému je v mojej aplikácii spustenie filtra oboma smermi a následná kombinácia výsledkov pomocou váženého priemeru závislého na vzdialenosti od začiatku trasy. Týmto je zaručené že prvý i posledný bod trasy zostali nezmenené a do trasy nebol zavedený žiaden skok. Nevýhodou tejto metódy je fakt, že filtrovanie sa spúšťa dva krát, avšak riešenie bolo implementačne jednoduché, pretože využíva základnú variantu filtrovania a úseky sú väčšinou oveľa kratšie než celková trasa, takže dvojnásobný beh filtra nie je časovo citeľný.

4.3.3 Implementácia manuálneho presúvania bodu trasy

Predpokladom pre možnosť presunúť bod trasy je nutnosť implementácie voľby bodu na trase. Pre tento účel som musel implementovať handler metódy `onMapClickListener` a `onMarkerDragListener`. Listener kliknutia na mapu sa používa pre získanie zemepisných súradníc miesta na mape na ktoré užívateľ klikol. Zo získaných súradníc vypočítam ich vzdialenosť od každého bodu trasy na súradnice bodu ktorý je najbližie sa umiestni Marker, ktorý sa nastaví ako ťahateľný. Pokial vzdialenosť najbližieho bodu presahuje istú konštantnú hodnotu, žiaden bod sa nepridá.

Druhá spomenutá metóda, teda `onMarkerDragListener` sa zavolá keď mapa detekuje ťahanie jedného z markerov. Počas ťahania v reálnom čase získavam novú polohu bodu. Pri každej zmene polohy ťahaného bodu sa vypočíta jeho posun v smere x a y. Následne sa

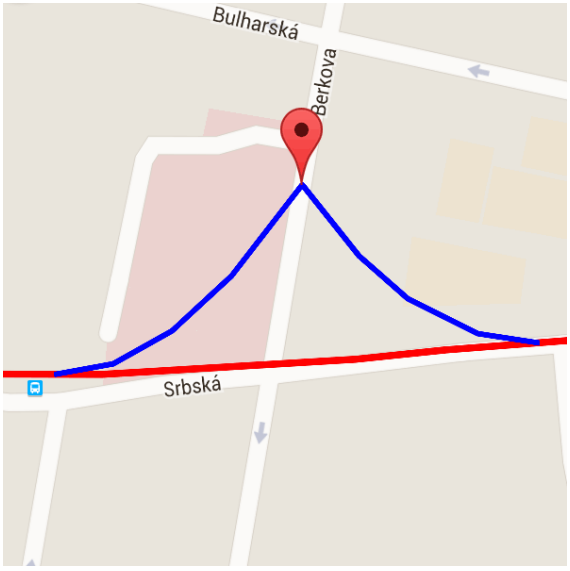
iteratívne prejde n bodov pred i za presúvaným bodom. Hodnota n je nastavená užívateľom pomocov posuvníka. Ku každému z týchto bodov je následne pripočítaný posun v smere x a y vynásobený pomerom zavislým na indexovej vzdialenosti od presúvaného bodu. Pomer je definovaný nasledujúcou rovnicou:

$$p_i = \left(1 - \frac{|i_{mid} - i|}{n}\right)^2 \tag{4.1}$$

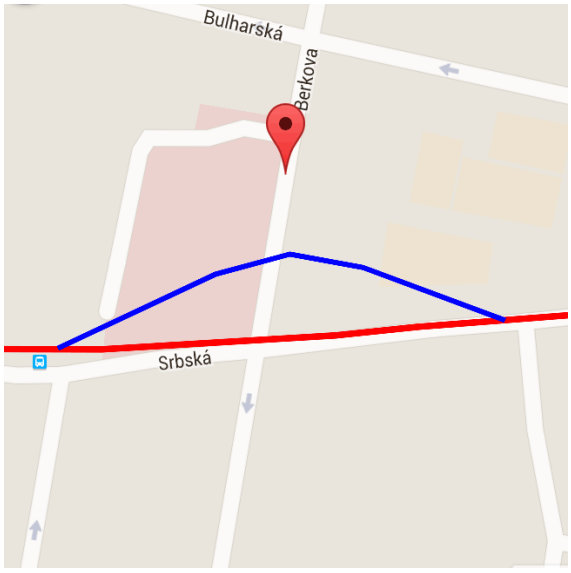
Jedná sa o modifikovanú rovnicu paraboly, kde pláti nasledovné; i je index aktuálne spracúvaného bodu, p_i je pomer posunu bodu s indexom i, i_{mid} je index hlavného presúvaného bodu a n je celočíselná hodnota určujúca počet okolitých bodov na ktoré sa presun vzťahuje. Nové súradnice aktuálne spracúvaného bodu okolia sa následne získajú takto:

$$x_{new} = x_{old} + \Delta x_{mid} * p_i \tag{4.2}$$
$$y_{new} = y_{old} + \Delta y_{mid} * p_i \tag{4.3}$$

Dôsledkom takéhoto presunu je, že čím je bod z intervalu ovplyvnených bodov bližšie k bodu presúvanému, tým viac ho presun ovplyvní, pričom miera ovplyvnenia rastie exponenciálne s klesajúcou vzdialenosťou indexu. Príkladom môžeme pozorovať na obrázku 4.1. Na výsledok presunu je zároveň možné aplikovať i Kalmanov filter. Výsledok je následne dostatočne zjemnený, tak ako je viditeľné na obrázku 4.2.



Obrázek 4.1: Výsledok manuálneho presunu bodu s ovplyvnením okolia na relatívne rovnej trase.



Obrázek 4.2: Výsledok manuálneho presunu bodu kombinovaného s Kalmanovým filtrovaním výsledku.

Kapitola 5

Záver

5.1 Dosiahnuté výsledky

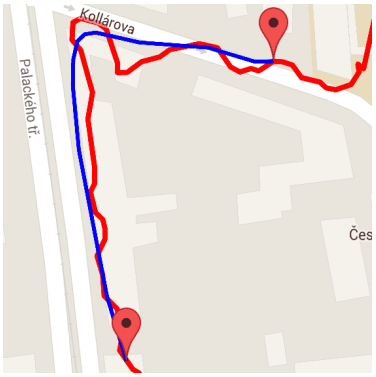
Aplikácia, ktorá je výsledkom tejto práce je použiteľná na filtrovanie bežných GPS aktivít, akými sú beh či cyklistika. Mieru filtrovania riadi užívateľ, pričom kvalita výsledného prefiltrovaného záznamu je priamo úmerná úsiliu ktoré je užívateľ ochotný vynaložiť. Ak užívateľ aplikuje filter na trasu ako celok, proces bude rýchly avšak chyby stále v menšej miere prítomné. Ak sa užívateľ naopak rozhodne venovať samotnej práci s aplikáciou aspoň 10 minút, môže kombináciou celkového filtrovania, manuálnymi úpravami nepresností a lokálnym filtrovaním dosiahnuť takmer ideálne výsledky, ktoré presne kopírujú reálnu trasu. Aplikáciu som počas vývoja priebežne používal a musím priznať, že moje očakávania splnila.

5.1.1 Analýza výsledkov Kalmanovho filtra

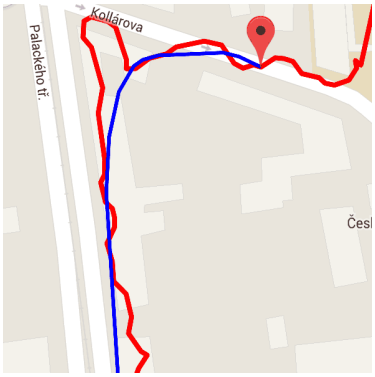
Časti trasy filtrované Kalmanovým filtrom implementovaným v tejto aplikácii sú vyhladené a už pri nízkej sile filtrovania zaniká väčšina drobných skokov či zanášaní, tak ako môžeme vidieť na obrázku 5.1.

Pri nadmerne vysokej sile filtrovania sú však ostré zabočenia až príliš zjemnené, čím vznikajú veľmi pozvoľne smerujúce zákruty ako na obrázku 5.2.

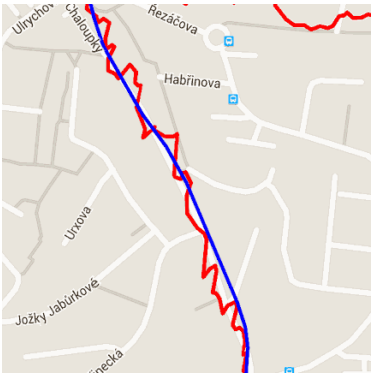
Vysoká sila filtrovania je vhodná najmä na rovné úseky bez prudkých zmien smeru s vysokou mierou chybných údajov, ako na obrázku 5.3.



Obrázek 5.1: Ľahké Kalmanovo filtrovanie prudkého zabočenia.



Obrázek 5.2: Silné Kalmanovo filtrovanie prudkého zabočenia.



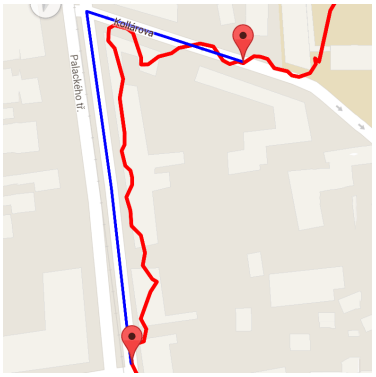
Obrázek 5.3: Silné Kalmanovo filtrovanie na rovnej trase.

5.1.2 Analýza výsledkov Roads API

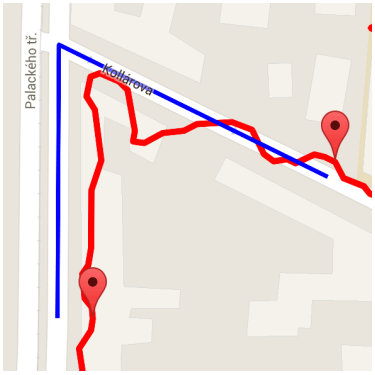
Pri použití funkcie upínania trasy na cesty pomocou *Roads API*, popísanej v sekcii 3.5, je možné dosiahnuť ideálne výsledky. Pod pojmom *ideálne* v tomto prípade myslím trasu, ktorá presne kopíruje cestu na mape, na ktorej bola aktivita zaznamenaná. Pri vhodne zvolenom začiatočnom a koncovom bode, tak ako na obrázu 5.4, sú výsledky bezchybné a kopírujú skutočnú trasu. Vhodne zvoleným začiatočným a koncovým bodom rozumieme body nachádzajúce sa *na* ceste a nie vedľa nej. V prípade že takéto body nie sú k dispozícii a celá dĺžka trasy, ktorú chceme upnúť sa nachádza mimo cesty, môžeme pomocou manuálnej úpravy ťahaním dva body presunúť aby tento predpoklad splňovali.

Pri nevhodne zvolených hraničných bodoch, teda mimo cesty, prebehne upínanie v poriadku, avšak na miestach kde je filtrovaná oblasť napojená na pôvodnú trasu vznikne skok, tak ako môžeme na hraniciach úseku na obrázku 5.5.

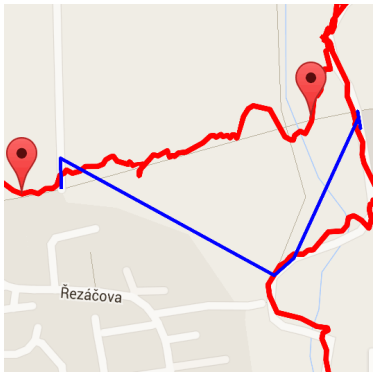
V prípade že upínanie aplikujeme na nevhodný úsek, ktorý neobsahuje skutočnú cestu výsledky sú nepredvídateľné a chaotické. Príkladom je obrázok 5.6.



Obrázek 5.4: Vhodne zvolené body úseku upínania na cesty.



Obrázek 5.5: Nevhodne zvolené body úseku upínania na cesty.



Obrázek 5.6: Upínanie na nevhodnom úseku je nepredvítaľné.

5.2 Možnosti rozšírenia

Rozšírenie aplikácie je realizovateľné vo viacerých smeroch. Keďže cieľom tejto práce bolo primárne vytvoriť jednoduchú, funkčnú aplikáciu, užívateľské rozhranie bolo skôr účelné. Rozhranie by sa dalo rozhodne zlepšiť a bude pre mňa prioritou v ďalšom vývoji.

Čo sa týka funkcionality, aplikácii by prospela možnosť prihlásenia sa pomocou API bežne používaných služieb, ktoré som spomínal v sekcii 2.3. Užívateľ by bol pri takomto spojení úplne odtienený od práce s GPX súbormi, ktoré by mohli byť vyžiadané a opätovne odovzdané na server takpovediac „pod kapotou“. Neposlednou možnosťou rozšírenia je zavedenie istej formy umelej inteligencie, ktorá by odľahčila prácu užívateľovi či už rozpoznávaním podliehajúcich ciest za pomoci aplikačných rozhraní spomínaných v sekcii 2.5 či inteligentným aplikovaním lokálneho filtrovania.

Literatura

[1] AlternativeVision: GPXParser. Online.
URL http://gpxparser.alternativevision.ro/

[2] Android Developers: Dashboards. Online.
URL http://developer.android.com/about/dashboards/

[3] van Brummelen, G. R.: *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton University Press, 2013, ISBN 9780691148922.

[4] Chmelař, P.: *JKalman*. Online.
URL https://sourceforge.net/projects/jkalman/

[5] Czerniak, G.: *Kalman Filters for Undergrads 1*. Online.
URL http://greg.czerniak.info/guides/kalman1/

[6] Dave Gandy: Font Awesome. Online.
URL http://fontawesome.io/

[7] Fajmon, B.; Hlavičková, I.; Novák, M.; aj.: *Numerická matematika a pravděpodobnost*. Online, 2014.
URL http://www.vutbr.cz/www_base/priloha.php?dpid=82874

[8] Google Developers: Advanced Concepts. Online.
URL http://developers.google.com/maps/documentation/roads/advanced

[9] Google Developers: Android NDK. Online.
URL http://developer.android.com/tools/sdk/ndk/index.html

[10] Google Developers: Google Maps Roads API Usage Limits. Online.
URL http://developers.google.com/maps/documentation/roads/usage-limits

[11] Google Developers: Introduction to the Google Maps Roads API. Online.
URL http://developers.google.com/maps/documentation/roads/intro

[12] Greg Welch, G. B.: *An Introduction to the Kalman Filter*. University of North Carolina at Chapel Hill Department of Computer Science, 2001.
URL
http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf

[13] Hira, M. I. H.: android-file-chooser. Online.
URL http://github.com/imranhasanhira/android-file-chooser

[14] Inman, J.: *Navigation and Nautical Astronomy: For the Use of British Seamen*. W. Woodward, C. & J. Rivington, 1821.

[15] Polok, L.; Šolony, M.; Ila, V. S.: *SLAM++*. Online.
URL http://sourceforge.net/p/slam-plus-plus/wiki/Home/

[16] Veness, C.: *Calculate distance, bearing and more between Latitude/Longitude points*. Online.
URL http://www.movable-type.co.uk/scripts/latlong.html

[17] Xamarin: *Mobile Application Development to Build Apps in C#*. Online.
URL https://www.xamarin.com/platform

Přílohy

Seznam příloh

Příloha A

Obsah CD

Priložené CD obsahuje nasledujúce priečinky súbory:

- BP_elektronicka_forma.pdf – Elektronická forma tejto bakalárskej práce.
- BP_listinna_forma.pdf – Listinna forma tejto bakalárskej práce pripravená na tlač.
- GPSFilter.apk – samotná aplikácia pripravená na inštaláciu do Android zariadenia s minimálnou verziou systému 4.1.
- Source – Priečinok obsahujúci celý projekt pre IDE Android Studio 2.0.
- example_input.gpx – Vzorový GPX súbor so záznamom aktivity v podpriemernej kvalite pre demonštráciu schopností aplikácie.
- app_demonstration.mp4 – Video s demonštráciou aplikácie.