VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

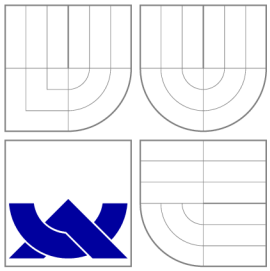
APLIKACE PRO MOBILNÍ TELEFON ZPRACOVÁVAJÍCÍ DATA Z GPS (SYMBIAN)

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

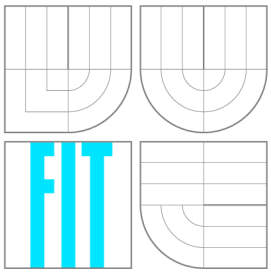
AUTOR PRÁCE
AUTHOR

ONDŘEJ KLUBAL

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

APLIKACE PRO MOBILNÍ TELEFON ZPRACOVÁVAJÍCÍ DATA Z GPS (SYMBIAN)

APPLICATION FOR MOBILE PHONE WHICH WILL PROCCESS GPS DATA (SYMBIAN)

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ KLUBAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN HORÁČEK
SUPERVISOR

BRNO 2010

Zadání bakalářské práce/9846/2009/xkluba00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2009/2010

Zadání bakalářské práce

Řešitel: **Klubal Ondřej**
Obor: Informační technologie
Téma: **Aplikace pro mobilní telefon zpracovávající data z GPS (Symbian)**
Application for Mobile Phone Which Will Proccess GPS Data (Symbian)
Kategorie: Vestavěné systémy

Pokyny:

1. Seznamte se s operačním systémem Symbian a možnostmi vývoje aplikací pro tento operační systém.
2. Navrhněte aplikaci, která z GPS přijímače zaznamená prošlou cestu zařízení a tu dále analyzuje (průměrná rychlost, celková délka, atd.). Aplikace by měla sloužit jako osobní tréninkový deník.
3. Navrženou aplikaci implementujte.
4. Promyslete další možná rozšíření Vaší aplikace.
5. Vytvořte poster prezentují vaše dílo.

Literatura:

- Stránky zabývajací se vývojem pod OS Symbian: http://www.symbian.org
- Babin, S.: Developing Software for Symbian OS :an introduction to creating smartphone application in C++, John Wiley & Sons, Chichester, 2006, ISBN 0-470018-45-3
- Aubert, M.: Quick recipes on Symbian OS :mastering C++ smartphone development, John Wiley, Hoboken, 2008, ISBN 978-04-709-9783-3
- Hojgr, R.: GPS :praktická uživatelská příručka, Computer Press, Brno, 2007, ISBN 978-80-251-1734-7
- Další dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

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

Vedoucí: **Horáček Jan, Ing.**, UITS FIT VUT
Datum zadání: 1. listopadu 2009
Datum odevzdání: 19. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato bakalářská práce pojednává o návrhu aplikace pro mobilní platformu Symbian S60 5th Edition. Hlavním účelem této aplikace je sběr dat z mobilního GPS modulu, jejich záznam ve standardním formátu a analýza zaznamenaných tras. Zvláštní pozornost je věnována přesnosti dat a filtrování trasových bodů. V aplikaci je dále možné porovnávat zaznamenané trasy. Aplikace také umožňuje vizualizaci GPS dat na displeji mobilního telefonu ve formě grafu a zobrazení náhledu trasy bez mapového podkladu.

Abstract

This bachelor thesis describes a design of an application for Symbian S60 Platform 5rd Edition. Main purpose of this application is to collect data from cellphone's GPS module, record them in standard format and analyze recorded traces. Particular attention is mainly focused on data accuracy and track points filtering. In this application it is possible to compare recorded traces. It also allows the visualization of the GPS data on cellphone's display as chart and previewing the route.

Klíčová slova

Symbian S60, Symbian 5rd edition, Nokia, Symbian C++, GPS, GPX, porovnávání tras

Keywords

Symbian S60, Symbian 5rd edition, Nokia, Symbian C++, GPS, GPX, comparing traces

Citace

Ondřej Klubal: Aplikace pro mobilní telefon zpracovávající data z GPS (Symbian), bakalářská práce, Brno, FIT VUT v Brně, 2010

Aplikace pro mobilní telefon zpracovávající data z GPS (Symbian)

Prohlášení

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

.....
Ondřej Klubal
17. května 2010

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé práce Ing. Janovi Horáčkovi za poskytnutí odborných rad, cenných připomínek a vstřícnou spolupráci. Dále bych chtěl poděkovat rodině a nejbližším za psychickou podporu.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V dnešní době se stále častěji setkáváme s mobilními telefony, které disponují vestavěným GPS modulem. Z vlastní praxe při používání mobilního telefonu s GPS navigací a zjištění některých nedostatků jsem se pokusil o jejich odstranění. Cílem této práce je vytvoření aplikace, jejímž úkolem je zaznamenání údajů o prošlých trasách v databázích, které jsou zpracovány tak, aby byly optimálně velké, minimálně zatěžovaly paměť telefonu a zaznamenávaly jen nezbytný počet trasových bodů. Další funkcí této aplikace je zpracování zaznamenaných dat a jejich vizualizace – např. graf nadmořské výšky, délky trasy, rychlosti aj. Dále lze tato data analyzovat a poskytnout přehledné statistiky. Mnohé z těchto funkcí jsou spíše doménou specializovaných GPS zařízení, které často neumožňují zobrazení grafových dat – umožňují maximálně zobrazit profil tratě a neposkytují tak podrobné statistiky. Zaznamenaná data jsou ukládána v souborech ve formátu GPX[17], který je standardem v oblasti ukládání GPS dat. Uživatel tedy bude moci tato data dále zpracovat i jiným způsobem v PC či jiném zařízení. Samozřejmostí je podpora více jazykových lokalizací.

V 2. kapitole se věnuji teoretickému pozadí, zejména volbě cílového programovacího jazyka a jeho specifičnosti oproti zažitým standardům, ale i krátkému popisu vývojového prostředí. Dále pak kapitola pojednává o globálním navigačním systému GPS a o formátu GPX. V následující kapitole 3. se zabývám popisem použitých algoritmů pro zpracování a vizualizaci GPS dat. Kapitola 4. se zabývá implementačními detaily aplikace a jejich závislosti na použité platformě. Poslední 5. kapitola je věnována vyhodnocení výsledků práce a možnostem dalšího rozšíření.

Kapitola 2

Teoretické pozadí

Za cílovou platformu jsem zvolil Symbian S60 5rd edition, která se oproti předchozím verzím vyznačuje zejména použitím dotykové obrazovky s vysokým rozlišením [10]. Ačkoli vlastním mobilní telefon, který podporuje aplikace 3rd i 5rd, rozhodl jsem se pro vyšší verzi zejména kvůli možnosti dotykového ovládání.

Pro vývoj aplikace jsem vybral Symbian C++ [8], který umožňuje tvorbu nativních aplikací pro cílovou platformu a dobrý přístup k hardwaru telefonu. Oproti jazyku Python nevyžaduje na koncovém zařízení dodatečnou instalaci podpůrných knihoven. Také je možné vytvářet aplikace pomocí QT toolkitu, ale v době započetí psaní této práce ještě nebyla vydána finální verze, která by umožňovala přístup k službám získávání polohy, tedy jej nebylo možné pro vývoj aplikace použít.

2.1 Symbian C++

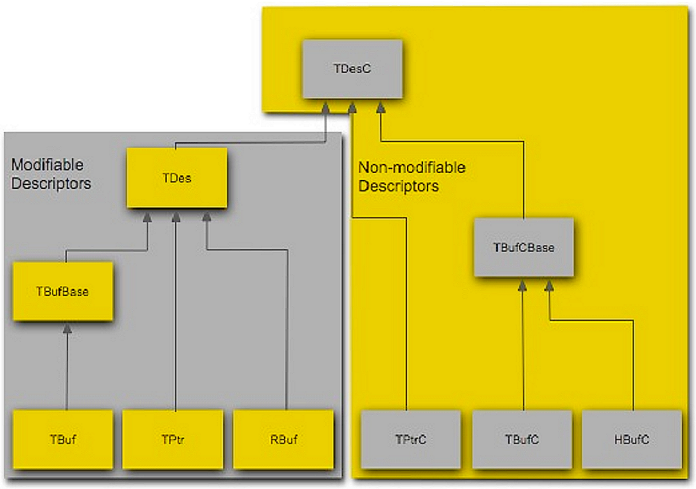
Symbian C++ se od standardního C++ liší v mnoha faktorech, zejména pak absencí STL (*Standard Template Library*). Setkáváme se zde také s odlišným značením datových typů např.: `TInt` - `int`, `TReal` - `real`. Pro uchovávání textových řetězců zde nenalezneme datový typ `string`, ale místo něho se používají takzvané deskriptory[13]. Deskriptory mají pevně danou maximální velikost. Existují i dynamické deskriptory, jejichž obsluha není triviální a často je nelze použít se všemi vestavěnými funkcemi.

Další výraznou odlišností je jiný způsob obsluhy výjimek. Využívá se zejména makra `TRAP` a jeho variací `TRAP_IGNORE`. Metody, které mohou způsobit výjimku, se vyznačují příponou `L`. Takovéto metody se nemohou vyskytnout v konstruktoru ani v destruktoru, aniž by byly zachyceny makrem `TRAP`. Při konstrukci objektů se využívá takzvané dvoufázové konstrukce, kdy standardní konstruktor nesmí obsahovat žádný kód, který může způsobit výjimku, proto se používá výhradně pouze k inicializaci proměnných objektů. Kód, který může způsobit výjimku, je skryt v metodě `ConstructL`.

Nová instance objektu se vytváří pomocí metod objektu `NewL` či `NewLC`. Pokud zavoláme metodu, která může způsobit výjimku v destruktoru bez zachycení makrem `TRAP`, dojde k vyvolání neošetřitelné výjimky zvané *Panic*. Tyto chyby jsou také vyvolávány neoprávněným přístupem do paměti, poškozeného *resource* souboru aj. nepříliš častými a odvoditelnými zdroji.

V Symbian C++ se také vyskytují dohodnuté konvence, které pomáhají odlišit jednotlivé identifikátory [8]. Třídy se odlišují počátečním písmenem a mohou být těchto typů:

- `T` - Třídy datových typů (*Data type classes*) slouží k zapouzdření datových typů jako



Obrázek 2.1: Deskriptory – převzato z [13]

`TInt`, ale také jednoduchých tříd, které mají manipulační metody `TBuf`, `TFixedArray` a další. Nemají destruktor, tudíž nemohou obsahovat dynamicky alokovaný obsah.

- `C` - Dynamicky alokované třídy na haldě odvozené od třídy `CBase` (*Heap allocated classes derived from* `CBase`). Jedná se o nejběžnější druh třídy. Má konstruktor i destruktor. Instance třídy se vytváří pomocí `NewL(C)`. Jedná je např. o `CAknView`, `CCoeControl`.
- `H` - Dynamicky alokované třídy na haldě, které nejsou odvozeny od `CBase`. Nejsou příliš časté. Typickým zástupcem je `HBufC`.
- `R` - Třídy zprostředkovávající přístup ke zdrojům (*Resource classes*). Nejčastěji obsahují *handle*. Data jsou umístěna na vzdálené straně. Příkladem může být třída `RPositionServer`, která slouží k získávání dat z GPS. Do této kategorie také spadají soubory `RFile` či dokonce dynamická pole `RArray`. Velmi často se využívá koncepce klient – server. Inicializace těchto objektů se provádí pomocí `Open()`, ukončení práce pomocí `Close()`.
- `M` - Třídy rozhraní (*Interface classes - mixins*). Jedná se o abstraktní třídy, jejichž účelem je rozšíření funkčnosti třídy. Využívá se vícenásobné dědičnosti.
  Např. `MRemConCoreApiTargetObserver`.

Další prefixy, se kterými se můžeme setkat, jsou například:

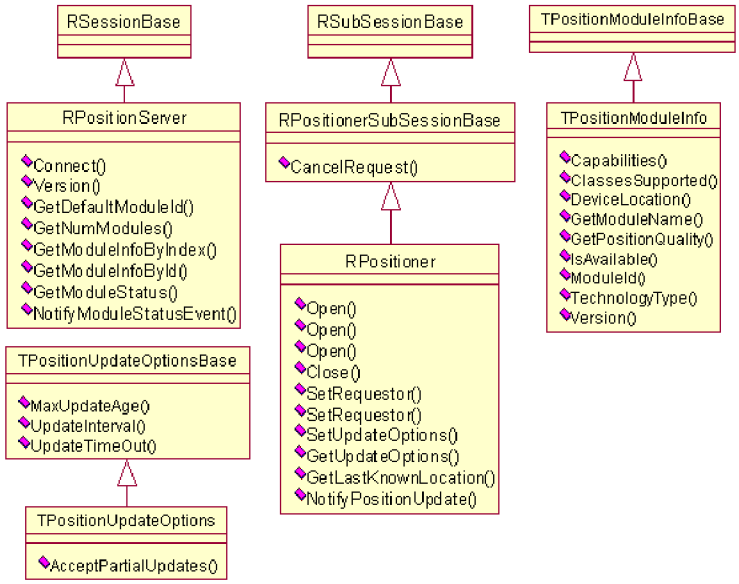
- `K` - konstanty
- `i` - členské proměnné
- `a` - argumenty funkcí
- `E` - prvek výčtu

Existují také postfixy:

- `L` - Metoda, která může způsobit zachytitelnou výjimku. Např. `NewL()` – objekt nemusí být vytvořen z důvodu nedostatku paměti aj.
- `C` - Metoda, která po svém skončení zanechá jednu nebo více položek na úklidovém zásobníku (*cleanup stack*). Např. `NewLC()` zanechá na zásobníku ukazatel na nově vytvořený objekt. Může se také jednat o třídy, které obsahují konstantní (nemodifikovatelný) obsah (`TDesC`).
- `D` - Metoda, po jejímž vykonání dojde ke zrušení objektu. Např. `ExecuteLD()`.

2.1.1 Location Acquisition API

Slouží k získání GPS dat. Jak je na platformě S60 zvykem, využívá se principu klient–server. Nejprve je třeba se pomocí třídy `RPositionServer` připojit k pozičnímu serveru. Potom lze získat informace o dostupných GPS modulech, o jimi poskytovaných službách a jejich aktuálních stavech. Také slouží k přepínání aktivních GPS modulů.



Obrázek 2.2: Třídy Location API

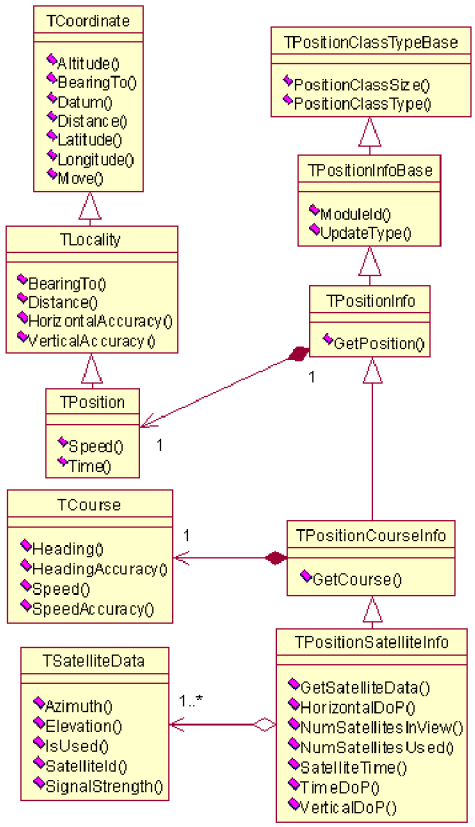
K získávání informací o pozici je zapotřebí pomocí třídy `RPositioner` navíc vytvořit *subsession*, díky které můžeme provádět další akce:

- Získání aktuální pozice
- Získání rozšířených informací o pozici
- Zjištění poslední známé pozice
- Zrušení požadavku na zjištění pozice
- Získávání periodických aktualizací

- Aktualizace pozice s časovým limitem

Když máme vytvořeno kompletní spojení, nastavíme parametry získávání dat. To provedeme zavoláním metody `SetUpdateOptions()`, která očekává `TPositionUpdateOptions`. `SetUpdateInterval()`nám říká, v jakém intervalu bude server poskytovat aktualizace pozice. `UpdateTimeOut` určuje, za jak dlouho bude požadavek na získání pozice zrušen. Poslední z časových intervalů je `SetMaxUpdateAge()`, který určuje maximální stáří zasílané aktualizace. Také můžeme povolit získávání neúplných aktualizací, které jsou poskytovány v době výpadku signálu a obsahují poslední známou pozici.

Nyní je vše nastaveno. Můžeme zahájit příjem poslední známé pozice `GetLastKnownPosition()`, či získat aktuální `NotifyPositionUpdate()`.



Obrázek 2.3: Datové třídy Location API

K uchovávání dat slouží třída `TCoordinate`, která obsahuje pouze základní informace o poloze. Třída `TLocality` ji rozšiřuje údaji o přesnosti, `TPosition` o časovou značku pozice dle času mobilního telefonu. K předávání dat z pozičního serveru slouží třída `TPositionInfo`, ze které lze získat `TPosition`.

První pozici musíme vyžádat pouze se standardními informacemi typu `TPositionInfo`, protože použijeme-li přímo `TPositionSatelliteInfo`, získání pozice se nezdaří a skončí chybou. Jakmile obdržíme výsledek, zjistíme aktuálně použitý modul `ModuleId` z přijatých dat. Dále zjistíme, zda došlo ke změně od minulé odpovědi. Pokud se `ModuleId` liší, určíme, které druhy pozičních dat umí poskytovat. Zavoláním metody `GetModuleInfoById()` v rámci session požádáme server o zaslání

`TPositionModuleInfo` použitého modulu. Prověřením `EPositionSatelliteInfoClass & ClassesSupported(EPositionInfoFamily)` třídy `TPositionModuleInfo` si můžeme zjistit, zda modul poskytuje `TPositionSatelliteInfo` a nastavit příjem těchto informací.

Mezi rozšířené informace patří údaj o aktuální rychlosti *Speed* a směru pohybu *Heading*, které jsou mnohem přesnější než údaje, které vypočteme z uložených pozic. Poslední úrovní je `TPositionSateliteInfo`, kde lze získat hodnoty *DOP*, počet aktivních či viditelných družic a přesný satelitní GMT čas.

2.1.2 Aktivní objekty a vlákna

Na platformě Symbian lze paralelní procesy provádět *aktivními objekty* (AO) [14] a *vlákny* [15]. Aktivní objekty jsou využívány pro asynchronní programování. Jsou preferovány před používáním vláken. Umožňují provádění několika asynchronních činností v rámci jednoho vlákna, tím odpadá nutnost přepínání kontextu vlákna. Využívány jsou zejména při dlouhých činnostech – například načítání souborů. Při krátkých činnostech (mohou trvat různou dobu) je využíváme například při žádosti na poziční server.

- Synchronní funkce je prováděna okamžitě a řízení je předáno zpět po jejím dokončení.
- Asynchronní funkce okamžitě předá řízení zpět a činnost je vykonána později. Volajícímu signalizuje úspěšnost dokončení operace signálem (*event*). Volající mezi tím může provádět další akce – například zobrazit dialog oznamující čekání.

Všechny objekty, které volají nějakou asynchronní činnost, jsou odvozeny od třídy `CActive`. Správa a časování AO je řízena `CActiveScheduler`. Při vytvoření aktivního objektu musíme tedy definovat prioritu AO (např. `EPriorityStandard`) a přidat AO do časovače aktivních objektů (`CActiveScheduler::Add()`).

Před tím, než provedeme asynchronní požadavek, je třeba zkontrolovat, zda již požadavek neprobíhá, protože jeden AO může provádět v daném okamžiku právě jeden asynchronní požadavek. V opačném případě by došlo ke vzniku kritické chyby. Té musíme předejít tak, že zrušeníme stávající požadavek. Můžeme vyslat nový požadavek, nebo počkáme na dokončení a požadavek provedeme později. Chceme-li odeslat požadavek na poskytovatele služby, musíme mu předat stavové slovo `iStatus`, které asynchronní služba nastaví na `KRequestPending` ještě před započetím asynchronní události. Dále nastavíme (`SetActive()`) AO jako čekající.

Jakmile služba dokončí asynchronní požadavek je zavolán `RequestComplete()`. Časovač AO pak provede metodu `RunL()` AO, kde můžeme otestováním stavového slova `iStatus` zjistit úspěšnost asynchronní operace. AO také umožňuje zrušení právě probíhajícího požadavku zavoláním `Cancel()` a současně musí být naimplementována metoda `DoCancel()`, ve které lze definovat dodatečné akce v případě zrušení požadavku. Při zrušení AO musí být zrušen probíhající požadavek.

Požadujeme-li provedení dlouhé synchronní operace asynchronně, není možné přímo použít AO, ale kombinaci AO s vláknem `RThread`. V konstruktoru AO také vytvoříme nové pozastavené vlákno. Při spuštění AO nastavíme `iStatus` na `KRequestPending`, aktivujeme AO a spustíme provádění vlákna. Při zrušení požadavku pozastavíme vlákno. To jako takové je reprezentováno statickou metodou, které můžeme předat jeden parametr – ten použijeme jako ukazatel na AO a budeme díky němu komunikovat s okolím. V těle vlákna definujeme nekonečný cyklus, ve kterém zavoláme naši synchronní operaci. Po jejím provedení zavoláme `RequestComplete()` a nastavíme `iStatus` na `KErrNone`, čímž AO

signalizujeme, že operace proběhla v pořádku. Nesmíme zapomenout uspat vlákno, jinak by došlo k zacyklení. Uzavření do nekonečného cyklu je nutné, protože v případě úplného ukončení vlákna ho již nelze znova spustit.

2.2 Carbide C++

Carbide C++ je vývojové prostředí založené na Eclipse. Jeho součástí je UI Designer, který slouží ke grafickému návrhu uživatelského rozhraní aplikace.

Pro vývoj aplikace jsem použil verzi 2.0.0 (v době psaní tohoto dokumentu je dostupná již verze 2.4), která obsahuje nezanedbatelné množství chyb. Jedná se zejména o nemožnost upravovat některé vlastnosti ovládacích prvků pomocí UI Designeru. Jedním z příkladů může být chybějící tlačítko pro přidání položky do `AknEnumeratedTextPopupSettingItem`. To slouží v nastavení k výběru z předem definovaných textových položek. Tento nedostatek jde obejít ručním upravením souboru `uidesign`.

Dále není možné definovat menu, které by mělo zaškrtávací či přepínací položky. Editace *resource* souboru je sice možná, ale při další úpravě přes UI Designer, dojde k přegenerování dané části *resource* souboru a tím ke ztrátě provedených změn. Nastavení parametrů za běhu programu není zcela spolehlivé. Jediným řešením je neoficiální patch[16].

2.3 GPS

GPS neboli *Global Positioning System*[7, 11, 3] je nejpoužívanějším navigačním družicovým systémem, který je provozovaný armádou Spojených států amerických. Díky tomuto systému lze určit polohu a přesný čas kdekoli na Zemi. Základy systému byly položeny v roce 1973, plného operačního stavu 24 družic bylo docíleno v roce 1994. V současné době je na oběžné dráze 31 družic [4] z 32 systémem umožňujících.

Systém GPS se skládá ze tří základních segmentů:

- Řídící – slouží ke sledování a řízení satelitů (korekce údajů, drah).
- Kosmický – skládá se ze soustavy družic, umístěných na oběžných drahách ve výšce 20200 km s dobou oběhu přibližně 12 hodin.
- Uživatelský – GPS přijímače, které přijímají signál z družic. Pro výpočet 2D polohy potřebujeme mít aktivní signál alespoň ze 3 družic. Chceme-li navíc určit i nadmořskou výšku, potřebujeme mít k dispozici signál alespoň ze 4 družic. Také je poskytován údaj o přesném čase s přesností $< 10^{-6}$s (SPS).

GPS je poskytována ve dvou úrovních přesnosti:

- PPS - *Precise Positioning Service*, která slouží výhradně autorizovaným uživatelům, zejména armádě USA a jejím spojencům. Příjem této služby vyžaduje speciální přijímač a dekódovací klíče pro dekódování P(Y) kódu.
- SPS - *Standard Positioning Service*, která je určena pro veřejný sektor a je volně dostupná. Využívá se C/A kód na frekvenci L1. V plánu jsou také nové veřejné signály L1C a L2C.

C/A kód je pseudonáhodná 1023 bitová sekvence blízká šumu (PRN kód). Každá družice má svůj vlastní PRN kód a je možné rozlišit až 32 družic.

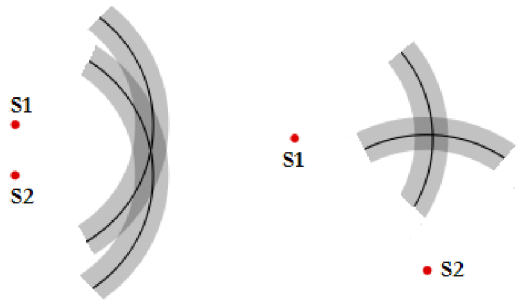
P(Y) kód je dlouhá pseudonáhodná sekvence. Vyšší přesnost této služby je dosažena díky dlouhému a rychlejšímu kódu. Vysíláním na L1 a L2 lze potlačit vliv ionosferické refrakce. Je dostupný pouze pro autorizované uživatele.

Frekvenční pásmo L1 je vysíláno na frekvenci 1575,42 MHz, L2 na frekvenci 1227,60 MHz.

2.3.1 Zdroje chyb v GPS

Na příjem GPS signálů má vliv mnoho faktorů[2]:

- *Selektivní dostupnost (SA)*. Do roku 2000 to byl jeden z největších zdrojů chyb. Jedná se o uměle zavedenou chybu do veřejně dostupného C/A signálu. Deaktivací SA došlo ke zvýšení přesnosti z 45m na 6,3m.
- Vliv polohy satelitů (*Satellite geometry*). Největší chyba nastane, když se aktivní satelity nacházejí v řadě za sebou na horizontu. Nejlepším stavem je, když jsou k sobě družice natočeny pod úhlem 90° (obr. 2.4).



Obrázek 2.4: *Satellite geometry*: vlevo – satelity v řadě, vpravo – ideální případ

  Špatná aktuální poloha satelitů může vést k chybě až 100m, výjimečně i 150m. Vliv polohy satelitů lze vyjádřit hodnotou rozptylu přesnosti – zvaného zkráceně DOP (Dilusion of precision). Jedná se o číslo v rozsahu 1-50. Existují tyto druhy:

  - GDOP – celkový DOP, $GDOP = \sqrt{PDOP^2 + TDOP^2}$
  - TDOP – časový DOP
  - PDOP – poziční DOP, 3D, $GDOP = \sqrt{HDOP^2 + VDOP^2}$
  - HDOP – horizontální DOP, 2D
  - VDOP – vertikální DOP, výška

- Odrazy – Chyba způsobená odrazem signálu od budov a jiných výškových převýšení. Odchylka v řádu několika metrů.
- Atmosférické jevy – Ionosferická a troposférická refrakce (lom rádiového signálu). Způsobuje chybu v rámci +-5m. Výrazně je lze omezit příjmem signálu na obou frekvencích L1 a L2 a díky tomu spočítat korekci. V aktuální době je toto možné jen v rámci PPS. V budoucnosti ale bude možné korekci vypočítat z L1C a L2C i v rámci SPS.

- Dráhy satelitů na orbitě – Ačkoli jsou satelity umístěny na velmi stabilních drahách, dochází k jejich posunu vlivem měnící se gravitace Země, Měsíce aj. vesmírných těles. Korekce je vysílána v rámci zprávy přijímači. Může způsobit chybu maximálně 2 metry.
- Díky vlivu zaokrouhlovacích chyb a nepřesnosti hodin vznikají další chyby (+-3m).

Existují také systémy evropský EGNOS a americký WAAS, které využívají pozemní stanice o známé poloze a vysílají korekční informace. Díky tomu lze eliminovat vliv atmosferických jevů, nepřesnosti hodin a drah satelitů. Příjem těchto informací však nemusí přijímač podporovat.

SPS s aktivním SA	±100 m
SPS s deaktivovaným SA	±15 m
PPS - Diferenciální GPS (DGPS)	±3 - 5 m
S korekčními daty WAAS/EGNOS	±1 - 3 m

Tabulka 2.1: Výsledná typická přesnost GPS

2.3.2 A-GPS

Assisted GPS slouží k rychlejšímu počátečnímu výpočtu pozice. Jedná se o dodatečné informace o polohách satelitů, součástí může být také přesný čas. Typicky se pro získání těchto informací využívá datového spojení mobilního operátora [6].

2.3.3 GPS modul telefonu Nokia 5800

Pro příjem GPS dat v aplikaci využívám interní modul telefonu. Bez využití služby *A-GPS* může trvat počáteční výpočet pozice i několik desítek minut v závislosti na kvalitě okolních signálů (uvnitř budovy, úzká ulice, les). Za příhodných podmínek lze očekávat ustálení během jedné minuty. Pokud se k výpočtu použijí aktuální A-GPS data, dojde k ustálení do 30s. Rychlost výpočtu závisí na počtu aktivních satelitů.

Informace o aktuální poloze jsou k dispozici až v době přítomnosti 4 aktivních družic. Výhodou tohoto je, že ihned po ustálení pozice, máme k dispozici všechny poskytované údaje včetně nadmořské výšky. Dojde-li během příjmu GPS signálu k poklesu aktivních družic na 3, přestane být dostupný údaj o aktuální nadmořské výšce. V případě poklesu pod 3 aktivní družice přestává GPS modul poskytovat informace o aktuální pozici a čase.

Modul poskytuje také rozšířené informace o aktuální rychlosti a kurzu. Výhodou těchto informací je velice dobrá přesnost poskytovaných údajů, nevýhodou je jejich častá nedostupnost.

2.4 GPX

GPX formát (*GPS eXchange Format*) slouží k uchovávání GPS dat. Je založen na XML schéma. Může obsahovat několik druhů informací – jednotlivé trasové body (*waypoints*), trasy (*routes*, zaznamenané – *tracks*) aj.

V této aplikaci slouží GPX soubory k ukládání zaznamenaných tras. Příklad zaznamenané trasy s jedním trasovým bodem:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<gpx xmlns="http://www.topografix.com/GPX/1/1" creator="GPSTraces"
     version="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.topografix.com/GPX/1/1
     http://www.topografix.com/GPX/1/1/gpx.xsd">
<metadata><time>2010-02-06T12:48:52Z</time></metadata>
<trk><name>10-ÚNOR-06 12:48:52 ODP.</name><trkseg>
<trkpt lat="49.393278" lon="15.393269">
<ele>746.00</ele>
<time>2010-02-06T11:27:28Z</time></trkpt>
</trkseg></trk></gpx>
```

Z příkladu je vidět, že se jedná o XML soubor s daty ve formátu GPX 1.1. `<Metadata>` obsahují element `<time>`, kde je specifikován datum a čas vytvoření souboru ve formátu ISO 8601. Dále se v souboru nachází element `<trk>`, který představuje zaznamenanou trasu. Jako název trasy `<name>` jsem zvolil aktuální čas.

Samotné trasové body `<trkpt>` obsahují údaje o zeměpisné délce a šířce bodu (`lat` a `lon`) ve stupních a jsou uzavřeny v sekvenci `<trkseg>`. Každý trasový bod obsahuje také element `<ele>` (*elevation*) – nadmořská výška a element `<time>`, který obsahuje přesný UTC čas vytvoření bodu. Volitelně lze do souboru také ukládat informace o přesnosti pozice jako je `<hdop>`, tt ¡vdop¿ a `<pdop>` a také počet satelitů `<sat>`. Dle standardní definice GPX není možné uchovávat údaj o aktuální rychlosti, ačkoli v dokumentaci verze GPX 1.0 byl element `<speed>` zmíněn. Přesto se můžeme setkat s aplikacemi, které tento nestandardní element umožňují zpracovat. Mezi ně patří i naše aplikace. Generování GPX souborů s tímto elementem je volitelné.

Kapitola 3

Návrh aplikace

Aplikace se skládá ze dvou základních bloků:

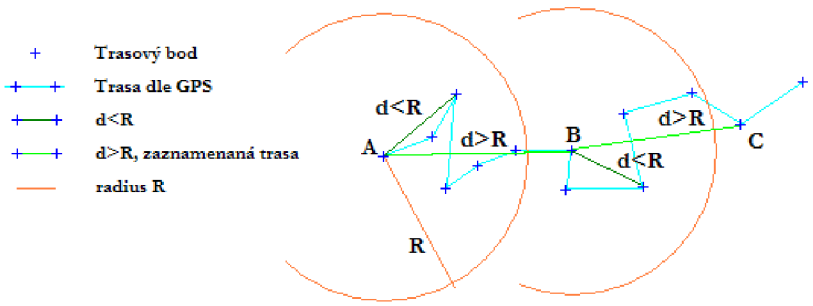
- První se zabývá příjmem GPS dat, filtrováním trasových bodů a vytvářením GPX souborů. K získání pozice využívá standardního Location API[9] a [12]
- Druhá část slouží k analýze zaznamenaných tras a jejich vizualizaci v podobě grafů a statistik (např. graf nadmořské výšky v závislosti na době trvání trasy či vzdálenosti, graf rychlosti, minima a maxima rychlosti, stoupání atd.). Dále umožňuje porovnání dvou GPX souborů. Společný úsek lze zobrazit na mapce nebo v grafu.

3.1 Filtrování trasových bodů

Při zaznamenávání všech dat poskytovaných pozičním serverem jsem zjistil, že spočítáním vzdálenosti této trasy získáme hodnoty řádově větší, než lze očekávat. Je to způsobeno nepřesnostmi GPS systému tím, že se neustále mění podmínky příjmu a tedy i poskytovaná pozice. Ponecháme-li přijímač déle na jednom místě se zapnutým záznamem, lze pozorovat změny aktuální pozice. Vypočítáme-li z těchto údajů vzdálenost „trasy“, získáme hodnotu ve stovkách metrů za necelých 5 minut záznamu. Toto je zcela nepřijatelné a je nutné tento problém řešit.

1) Tento problém můžeme řešit tak, že všechny údaje o poloze, které mají vzdálenost od výchozí polohy `A` nižší než určitý rádius `R`, můžeme považovat za shodné s výchozí polohou a nebudou tedy zaznamenávány. K uložení dalšího trasového bodu `B` dojde, až když jeho vzdálenost od bodu `A` bude větší než rádius `R`.

Důležitým faktorem pro správnou funkci tohoto algoritmu je optimální volba `R`. Pokud je hodnota příliš malá, může se stát, že výsledná trasa bude stále delší než reálná. Pokud naopak bude tato hodnota příliš vysoká, lze očekávat vyfiltrování i efektivních trasových bodů a zaznamenaná trasa bude v konečném důsledku kratší než trasa reálná. Zkrácení trasy se nejvíce projeví zejména v ostrých zatáčkách, které mohou být vyhlazeny či dokonce úplně vyfiltovány. V praxi je tedy třeba stanovit tuto hodnotu jako kompromis mezi přírůstkem způsobeným nestabilitou polohy a zkrácením, které je způsobeno vynecháváním příliš mnoha bodů. Jelikož přijímač testovacího mobilního telefonu neumí přijímat korekční informace WAAS/EGNOS a umožňuje příjem pouze SPS služby, lze očekávat typickou přesnost 15m dle tabulky 2.1, což se shoduje s dlouhodobým pozorováním údajů o přesnosti poskytovaných testovacím mobilním telefonem, kde se tento údaj pohyboval mezi 16–300 metry. Během experimentálního zaznamenávání tras jsem tedy vycházel z této hodnoty.



Obrázek 3.1: Filtrování trasových bodů podle `R`

Společně s kombinací dalších metod bylo možné snížit hodnotu `R` na 10 metrů. Ukázalo se však, že ideální nastavení této hodnoty není možné, neboť hodnota tohoto koeficientu závisí na mnoha faktorech – zejména na okolních podmínkách, dále pak na modelu daného mobilního telefonu a dokonce i verzi firmware.

2) Vyjdeme-li z předchozího algoritmu, lze si povšimnout, že body `B` a `C` byly zvoleny jen díky tomu, že byly první na řadě vně tolerančního kruhu. Můžeme tedy zavést prioritu dle hodnot přesnosti trasových bodů a do trasového soboru ukládat místo bodu B bod s nejlepší přesností mezi body A a B.

3) Opět vyjdeme z bodu 1). Místo hledání bodu s nejlepší přesností vypočítáme průměrnou souřadnici, kterou použijeme. Nevýhodou tohoto postupu je, že mimo souřadnice je třeba uložit také čas, rychlost aj. údaje, které by průměrováním ztrácely na relevantnosti.

V aplikaci je implementován postup popsaný v 1) doplněný o vynechávání trasových bodů, jejichž přesnost je horší než stanovená mez.

Výpadky GPS signálu

Jedním z dalších problémů, které bylo třeba vyřešit, jsou výpadky GPS signálu (snížení počtu aktivních družic pod tři) a tím způsobené úseky chybějících dat v zaznamenané trase. Ze strany aplikace nelze chybějící data nijak doplnit. Zamezit výskytu těchto výpadků není vždy možné, nejčastěji k nim dochází uvnitř a v blízkosti budov, v lese či dopravních prostředcích. Základem je volný výhled na oblohu. Jediná možnost zásahu ze strany aplikace je tedy možná pouze ve vhodné signalizaci těchto výpadků uživateli nebo zaznamenání akce. Krátké „jednorázové“ výpadky signálu lze přehlédnout, ale zejména dlouhé výpadky, trvající několik minut a vytvářející v zaznamenaných datech dlouhé mezery, je třeba řešit. Při výpadku signálu dojde k zobrazení údaje na display a v případě, že je povolen záznam trasy, uloží se do trasového souboru zpráva, že došlo k výpadku. Další funkcí je přerušení záznamu do trasového souboru při dlouhém výpadku. Po obnovení signálu se bude pokračovat v záznamu do nového trasového souboru. V nastavení aplikace lze specifikovat, po jaké době od výpadku signálu má dojít k vytvoření nového souboru.

Přerušování trasy při stání na místě

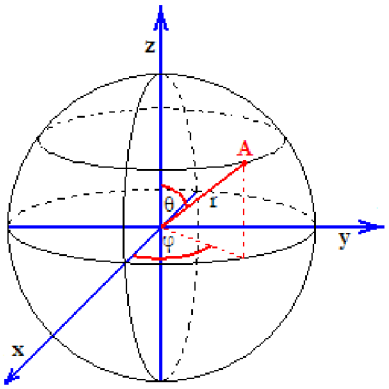
Dále se můžeme setkat s tím, že se mobilní telefon nepohybuje. Když tento stav trvá dlouho, lze předpokládat, že uživatel dorazil na cílové místo a není třeba nadále pokračovat v záznamu trasy. Pokud se telefon uvede opět do pohybu, budeme pokračovat záznamem do

nového souboru. Opět si stanovíme rádius poloh od místa zastavení – tento může být mnohem větší než u filtrování poloh při pohybu. Jako další si zavedeme časový interval, který nám řekne, jak dlouho se můžeme v této vzdálenosti pohybovat, aniž by došlo k přerušení trasy. Dojde-li k přerušení trasy, nejsou další polohy uvnitř tohoto kruhu zaznamenávány.

3.2 Výpočet vzdálenosti 2 pozic

Pro výpočet vzdálenosti dvou vzdálených pozic lze použít *Haversine formula*[18]. Ta Zemi nahrazuje pomocí *Great-circle* (koulí o daném průměru). Používá se *mean radius*, jehož hodnota je nastavena na 6371km. Pro velmi přesné výpočty, pro vzdálenosti, kde již není možné zanedbat to, že Země není ideální koule, se požívá *Vincenty formula* [19], která Zemi nahrazuje teoretickým elipsoidem. Používají se parametry referenčního elipsoidu WGS84 hlavní poloosa $a = 6378137m$, vedlejší poloosa $b = 6356752,31m$, převrácená hodnota zploštění $f = 1/298.2572235m$. Tyto algoritmy neuvažují nadmořskou výšku – počítají vzdálenost pozic v nadmořské výšce $0m$. Hodí se tedy pro výpočet ploché vzdálenosti. Pokud potřebujeme spočítat vzdálenost dvou velmi blízkých pozic (max. několik desítek metrů) má mnohem větší vliv na vzdálenost rozdílná nadmořská výška těchto bodů než zakřivení zemského povrchu. Tento výpočet reálné vzdálenosti, lze realizovat tak, že získané GPS údaje převedeme ze sférické souřadné soustavy do kartézské[5].

$$\begin{aligned} x &= r \sin\theta \cos\varphi \\ y &= r \sin\theta \sin\varphi \\ z &= r \cos\theta \end{aligned} \tag{3.1}$$



Obrázek 3.2: Převod sférické souřadné soustavy do kartézské – sférický model Země

Dosazením zeměpisné délky lon, šířky lat a nadmořské výšky alt získáme:

$$\begin{aligned} x &= (R + alt) \sin(\frac{\pi}{2} - lat) \cos lon \\ y &= (R + alt) \sin(\frac{\pi}{2} - lat) \sin lon \\ z &= (R + alt) \cos(\frac{\pi}{2} - lat) \end{aligned} \tag{3.2}$$

Za R dosadíme *mean radius* tj. $R = 6371009$ m. Takto převedeme obě pozice. Nyní můžeme spočítat jejich vzdálenost jako vzdálenost 2 bodů v 3D prostoru po přímce:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{3.3}$$

Tento postup je vhodný zejména kvůli své jednoduchosti a snadné implementaci. Existují i složitější a přesnější postupy převodu souřadnic využívající elipsoid (WGS84) a výpočet vzdálenosti po kružnici či elipse[1]. Tyto postupy jsou však výpočetně náročnější. Pro předpokládané vzdálenosti v řádu desítek metrů jsou rozdíly zanedbatelné.

3.3 Přehledy a statistiky

Údaje, které je možné získat analýzou trasového GPX souboru, rozdělíme do několika kategorií:

- vzdálenost
- čas
- rychlost
- nadmořská výška
- pozice
- přesnost
- obecné informace

Nyní se podíváme na jednotlivé údaje, které budeme analyzovat. Do kategorie vzdálenosti tedy patří:

- celková plochá vzdálenost: $\sum_{i=2}^{N} d_{flat}(p_i, p_{i-1})$
- celková reálná vzdálenost uvažující nadmořskou výšku: $\sum_{i=2}^{N} d_{real}(p_i, p_{i-1})$
- vzdálenost ve stoupavých úsecích: $\sum_{i=2}^{N} d_{real}(p_i, p_{i-1})$, kde $p.alt_i > p.alt_{i-1}, v_{act} > v_{min}$
- vzdálenost v klesajících úsecích: $\sum_{i=2}^{N} d_{real}(p_i, p_{i-1})$, kde $p.alt_i < p.alt_{i-1}, v_{act} > v_{min}$
- přímá vzdálenost ze startovní pozice do cílové pozice: $d_{real}(p_1, p_N)$

Jak je vidět, většina těchto údajů vyžaduje k jejich vypočítání projití všech trasových bodů. Těchto bodů může být i několik desítek tisíc v závislosti na délce trasy a hustotě záznamu. Tyto výpočty by tedy měly být prováděny hromadně, abychom předešli několikanásobnému procházení trasových bodů. Protože parsování GPX souborů je poměrně dlouhá operace a jsou při ní projity všechny trasové body, budeme realizovat tyto hromadné výpočty právě při načítání. Obdobná situace je i v dalších kategoriích.

Situace s rychlostí je o to složitější, že trasový soubor může obsahovat volitelný element `<speed>`, jak bylo rozvedeno v kapitole 2.4. Protože nelze zjistit předem bez projití trasových bodů, že se v zaznamenané trase vyskytuje tento element (pozn. element je ukládán

pouze je-li údaj o rychlosti dostupný), je třeba provádět výpočty zvlášť pro oba případy. Pokud není přítomen tento element, je třeba navíc vypočítat i aktuální rychlost. Tu můžeme spočítat velice jednoduše $v_{act} = d_{real}(p_i, p_{i-1})/(t_i - t_{i-1})$. Problém je, že vlivem nepřesnosti GPS dochází ke kumulaci chyb a ke kolísání aktuální hodnoty. Tento problém lze eliminovat aplikováním klouzavého průměru:

$$v_{act} = \frac{\sum_{i=n-a+1}^{n} d_{real}(p_i, p_{i-1})}{\sum_{i=n-a+1}^{n} (p.t_i - p.t_{i-1})} \tag{3.4}$$

Počet kroků a nám říká, na kolika předchozích hodnotách bude aktuální hodnota rychlosti závislá. Čím větší počet kroků, tím většího vyhlazení charakteristiky dosáhneme. Abychom nemuseli neustále dokola pro každý trasový bod sumarizovat hodnoty několika minulých bodů, odečteme od minulých sumárních hodnot p_{n-a} a přičteme aktuální p_n, doba výpočtu bude tedy vždy stejná pro libovolné a.

Uživatel si v aplikaci bude moci definovat, zda si přeje používat vestavěnou rychlost, pokud je dostupná. V případě používání vypočítané rychlosti může definovat počet kroků klouzavého průměru a.

Z aktuálních hodnot rychlosti pak dalšími výpočty určíme:

- maximální a minimální rychlost: $Max(v_{act}), Min(v_{act})$
- průměrná rychlost: $\sum_{i=2}^{N} d_{real}(p_i, p_{i-1}) / \sum_{i=2}^{N} (p.t_i - p.t_{i-1})$
- průměrná rychlost stoupání: $\sum_{i=2}^{N} d_{real}(p_i, p_{i-1}) / \sum_{i=2}^{N} (p.t_i - p.t_{i-1})$, kde $p.alt_i < p.alt_{i-1}, v_{act} > v_{min}$
- průměrná rychlost klesání: $\sum_{i=2}^{N} d_{real}(p_i, p_{i-1}) / \sum_{i=2}^{N} (p.t_i - p.t_{i-1})$, kde $p.alt_i > p.alt_{i-1}, v_{act} > v_{min}$

Situace ohledně času je mnohem jednodušší a většinu těchto údajů lze určit přímo:

- čas začátku trasy: $p.t_1$
- čas na konci trasy: $p.t_N$
- doba trvání: $p.t_N - p.t_1$
- doba stoupání: $\sum_{i=2}^{N} (p.t_i - p.t_{i-1})$, kde $p.alt_i < p.alt_{i-1}, v_{act} > v_{min}$
- doba klesání: $\sum_{i=2}^{N} (p.t_i - p.t_{i-1})$, kde $p.alt_i > p.alt_{i-1}, v_{act} > v_{min}$

Kde $p.alt$ je nadmořská výška trasového bodu, v_{act} je aktuální rychlost a v_{min} je uživatelsky definovaná minimální rychlost, která – pokud není dosažena – blokuje počítadla stoupání a klesání.

O nadmořské výšce nás může zajímat:

- minimální a maximální výška: $Min(p.alt), Max(p.alt)$
- průměrná výška: $\frac{\sum_{i=1}^{N} p.alt_i}{N}$
- celkové převýšení: $Max(p.alt) - Min(p.alt)$
- výška na počátku a na konci trasy: $p.alt_1, p.alt_N$
- počet nastoupaných metrů: $\sum_{i=1}^{N} p.alt_i$, kde $p.alt_i > p.alt_{i-1}$
- celkové klesání: $\sum_{i=1}^{N} p.alt_i$, kde $p.alt_i < p.alt_{i-1}$
- konečný rozdíl: $p.alt_1 - p.alt_N$

U pozice budeme zjišťovat krajní body, tj. nejsevernější, nejjižnější, nejzápadnější, a nejvýchodnější bod trasy. Tyto údaje budeme nadále využívat i při vykreslování mapového náhledu.

Zobrazení údajů o přesnosti je dostupné pouze tehdy, jestliže je ukládán element `<sat>` s počtem aktivních satelitů. Tento údaj je narozdíl od hodnot *DOP* přítomen vždy. Můžeme tedy zjistit minimální, maximální a průměrný počet aktivních satelitů a hodnot *DOP*.

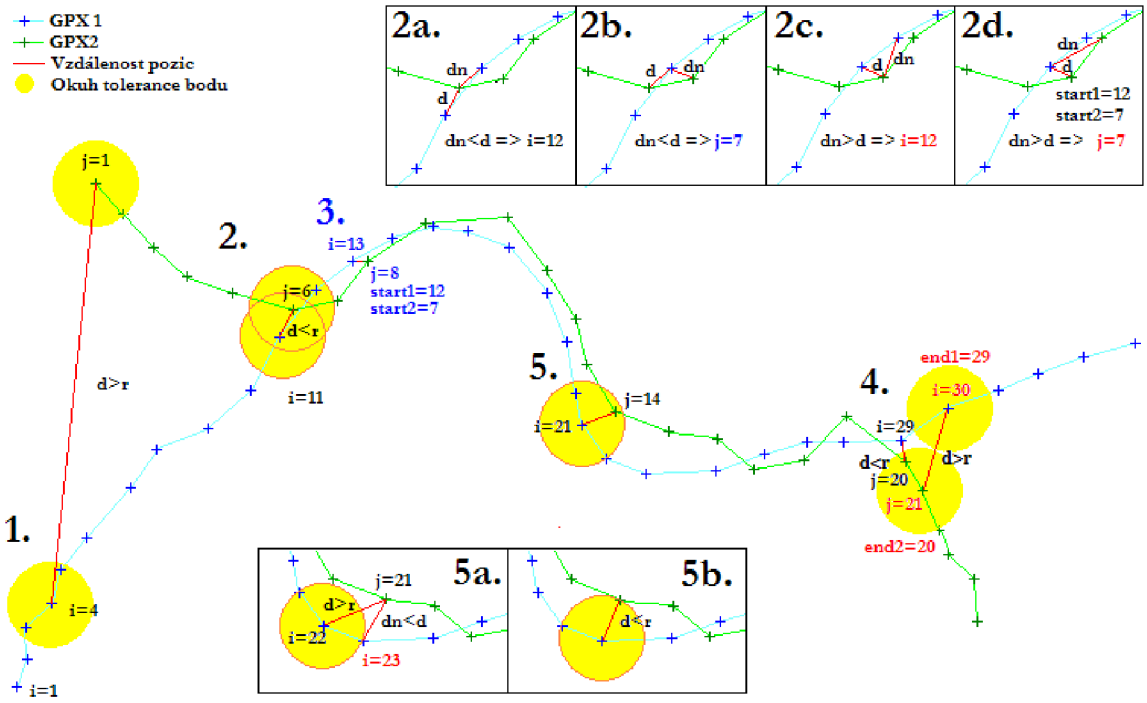
Z obecných informací nás bude zajímat:

- název trasy
- datum vytvoření souboru
- počet trasových bodů: N
- velikost souboru
- dostupnost rozšířených informací (rychlost/přesnost)

Nejzajímavějším údajem je počet trasových bodů, který se používá v moha výpočtech pro vykreslování grafu, porovnávání aj. Dostupnost rozšířených informací je zjištěna přítomností alespoň jednoho elementu `<speed>` či `<sat>`.

3.4 Porovnávání zaznamenaných tras

Vycházíme z toho, že máme načteny dva GPX soubory. Výsledkem algoritmu jsou indexy položek v GPX souborech, ukazující na začátek a konec společného úseku. Algoritmus pracuje tak, že postupně porovnává vzdálenosti pozic trasových bodů pomocí dvou zanořených cyklů (viz obr. 3.3, bod 1.). Pokud je vzdálenost pozic nižší než vzdálenost, která určuje okruh tolerance bodu, považujeme tyto pozice za začátek společného úseku (viz obr. 3.3, bod 2.). Ještě předtím však dojde k takzvanému dokročení – pokusíme se udělat v 1.trase další krok (viz obr. 3.3, bod 2a.). Dojde-li ke snížení vzdálenosti těchto bodů, považuje se tento nový index za začátek společného úseku (viz obr. 3.3, bod 2b.), není-li tomu tak, vrátíme změnu. Totéž se provede i ve druhé trase. Dojde-li alespoň v jedné z tras k úspěšnému kroku kupředu, dokročení opakujeme, jinak si zaznamenáme aktuální indexy obou tras (viz obr. 3.3, bod 2c., 2d.). Máme tedy určen začátek společného úseku.



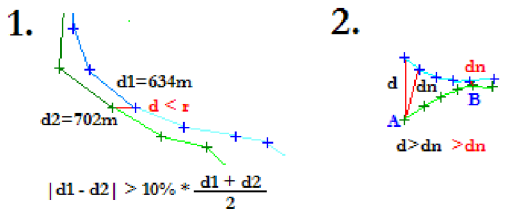
Obrázek 3.3: Porovnávací algoritmus tras

V obou trasách provedeme další krok a spočítáme vzájemnou vzdálenost těchto bodů (viz obr. 3.3, bod 3.). Algoritmus také počítá vzdálenost od počátečních bodů pro každý soubor zvlášť. Toto se využívá pro kontrolu, zda nedochází k příliš velké desynchronizaci způsobenou rozdílnou délkou tras. Pokud se délky tras liší o 10% (může být předefinováno), dojde k ukončení algoritmu (viz obr. 3.4, bod 1.). Toto omezení se uplatňuje až po dosažení minimální délky společného úseku. Protože trasy nemají obvykle zcela shodný počet a hustotu bodů, je nutné použít dokročení. Když jsme opět nalezli body s nejmenší vzájemnou vzdáleností, zkontrolujeme, zda je tato vzdálenost nižší než rádius. Je-li tomu tak, opakujeme vzájemný krok kupředu. V opačném případě opustíme tento cyklus a ověříme, zda nalezený společný úsek splňuje podmínku minimální délky. Pokud ji splňuje, poznamenáme si aktuální indexy jako konec společného úseku (viz obr. 3.3, bod 4.).

Algoritmus se umí vypořádat i s výjimečnou odchylkou, než je okruh tolerance bodu. Příklad je uveden na (viz obr. 3.3, bod 5.). Uplatní se zde dokročení, jehož výsledkem je nižší odchylka než maximální povolená (viz obr. 3.3, bod 5b.). Minimální vzdálenost společného úseku slouží k přeskočení příliš nezajímavých krátkých úseků nebo křížení tras. Také slouží pro omezení kontroly desynchronizace, protože při rozjezdu algoritmu dochází k mnohem větším procentuálním výkyvům délky tras, které by vedlo k jeho předčasnému ukončení.

Dokročení může způsobit zpožděné označení začátku společného úseku a to v případě pomalu se přibližujících tras, kde vzdálenost bodů je nižší než okruh tolerance bodu (viz obr. 3.4, bod 2.). Zkrácení může být i v řádech desítek metrů a není algoritmem řešeno – tato situace nastává jen ve velmi výjimečných případech a mnohdy ji lze zanedbat.

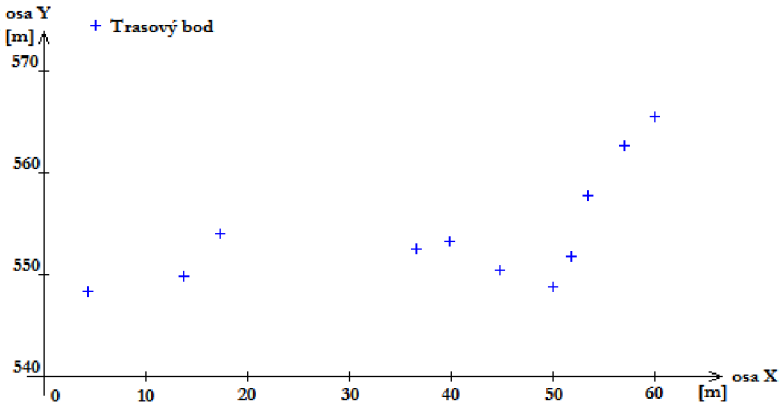
Když není nalezen žádný společný úsek, má tento popsaný porovnávací algoritmus v nejhorším případě kvadratickou složitost. V ideálním případě je při porovnání dvou zcela totožných tras složitost lineární.



Obrázek 3.4: Kontrola desynchronizace a problém dokročení.

3.5 Vykreslení grafu

Budeme uvažovat vykreslení grafu závislosti nadmořské výšky (osa Y) na projité vzdálenosti (osa X) (viz obr. 3.5). Pro jiné závislosti je situace obdobná. Základním problémem pro vykreslení je fakt, že zaznamenaná data mají odlišné vzájemné vzdálenosti a rozdílný počet trasových bodů. Algoritmus si musí poradit s tím, že trasových bodů k zobrazení je méně než počet sloupců (pixelů) – nazveme si ho *width*. Tehdy je třeba data v chybějících místech vhodně doplnit. Naopak je třeba umět data průměrovat – to v případě, že jednomu sloupci odpovídá více trasových bodů. V praxi se však mohou vyskytnout oba případy. Nelze tedy tato data přímo vykreslovat, ale je třeba provést přepočet pro správné zobrazení na display.



Obrázek 3.5: Data z GPX souboru pro zobrazení

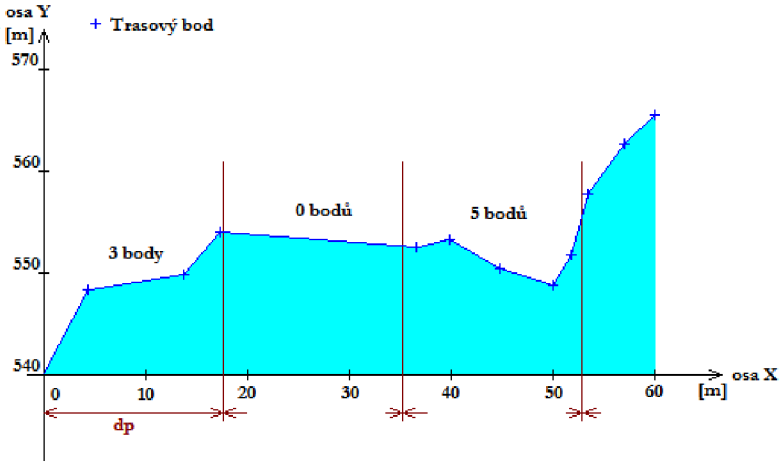
3.5.1 Přepočítávací algoritmus

Na vstupu očekáváme:

- načtený GPX soubor
- `dBegin` – zadaný začátek vykreslování v délkové míře na ose X
- `dEnd` – zadaný konec vykreslování v délkové míře na ose X
- `width` – počet sloupců k vykreslení

Výstupem pak bude naplněný vykreslovací buffer, který bude obsahovat požadovaný počet sloupců.

Odečteme-li *dBegin* od *dEnd*, získáme vzdálenost odpovídající zobrazenému úseku na obrazovce, tu následně podělíme *width* a získáme vzdálenost odpovídající jednomu sloupci. Nazvěme si ji šířka pixelu neboli *dp* (viz obr. 3.6). Předpokládejme, že bude vykreslování probíhat od začátku, tedy *dBegin* je 0. Na tomto obrázku je také naznačen počet trasových bodů v jednotlivých *dp* sekcích.



Obrázek 3.6: Šířka pixelu *dp*

Změnu hodnoty jednoho bodu na hodnotu druhého bodu budeme považovat za lineární a můžeme pospojovat jednotlivé trasové body přímkami. Výslednou hodnotu výšky sloupce získáme jako obsah plochy pod grafem podělený *dp*. Díky tomuto postupu nezáleží na vzájemné vzdálenosti jednotlivých bodů ani na počtu bodů k zobrazení.

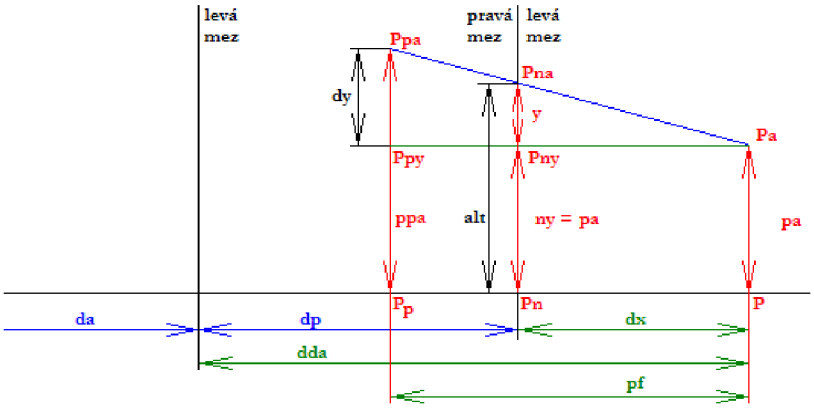
Plochu S ohraničenou dvěma body P_p a P můžeme vypočítat podle následujícího vzorce:

$$S_n = S_{n-1} + pf\frac{ppa + pa}{2} \tag{3.5}$$

Na počátku je $S_0 = 0$ a $n = 1$. P je aktuální trasový bod, P_p je předchozí bod. Vzdálenost těchto bodů nazveme pf a je součástí dat aktuálního bodu, kde *ppa* je předchozí výška a *pa* je výška aktuálního bodu.

Abychom mohli vypočítat obsah plochy, je třeba určit krajní body. Vzdálenost aktuálního sloupce od počátku osy X nazveme *da* a bude sloužit jako levá mez. Pravou mez získáme přičtením *dp* k *da*. Na počátku je *da* rovno 0 a leží v něm známý bod P_p. Budeme tedy pouze počítat hodnotu pravé meze, které se v následujícím kroku použije jako levá mez (viz obr. 3.7).

Výpočet plochy můžeme provádět, dokud subpixelová vzdálenost *dda* je nižší než *dp*, tj. nenacházíme se v následujícím sloupci. V takovém případě přejdeme na další trasový bod. Jakmile *dda* přesáhne *dp*, je třeba vypočítat nový pomocný bod P_n v místě $da + dp$. Hodnota *dda* je na počátku každého kroku inicializována jako vzdálenost mezi aktuálním a minulým bodem pf. Hodnotu pomocného bodu *alt* uvažujeme jako součet výšky obdélníka *ny* a trojúhelníka *y*. Nejdříve tedy určíme *ny*. Pro výpočet hodnoty *y* využijeme podobnosti trojúhelníků. Budeme uvažovat, že se jedná o klesající charakteristiku. Tedy $dda > da$ – pak je *ny* rovno *da*. Dále určíme odvěsny velkého pravoúhlého trojúhelníka. Těmi jsou vzájemná



Obrázek 3.7: Výpočet pravé meze $da + dp$

vzdálenost bodů pf a výška dy, která je v našem případě rovna $dda - da$. Odvěsnu malého trojúhelníka dx určíme jako $dda - dp$. Pokud je charakteristika stoupavá $dy < 0$, pak $dx = pf - dx$, $dy = |dy|$, $ny = dda$. Výslednou výšku nového bodu tedy získáme jako:

$$alt = ny + dy\frac{dx}{pf} \tag{3.6}$$

Obsah zbývající části od minulého bodu P_p k vypočítanému P_n získáme ze vztahu:

$$S = S_{n-1} + (pf - (dda - dp))\frac{ppa + alt}{2} \tag{3.7}$$

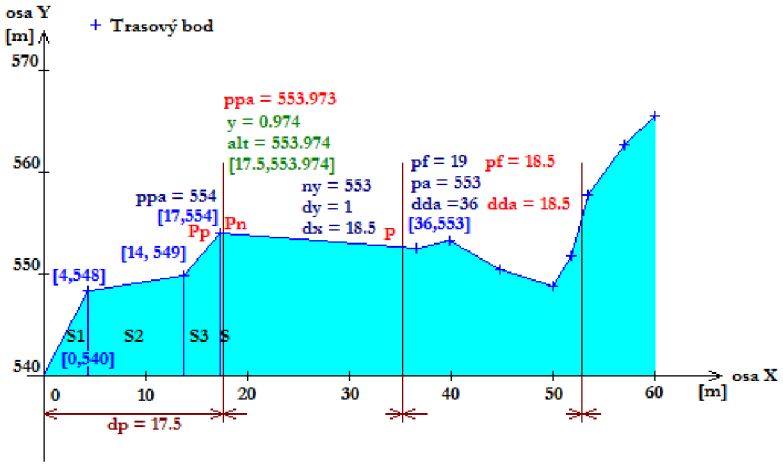
Hledanou průměrnou výšku sloupce vypočítáme jako $v = \frac{S}{dp}$. Abychom mohli pokračovat v následujícím kroku ve výpočtu dalšího sloupce, z vypočítaného bodu P_n uděláme minulý P_p a určíme vzdálenost pf k aktuálnímu P jako $dda - dp$. Protože jsme dokončili jeden krok, navýšíme vzdálenost da o velikost dp. Výpočet končí jakmile $da >= dEnd$.

V případě, že nebudeme vykreslovat od začátku souřadného systému (tj. $dBegin > 0$), je třeba se v GPX souboru nejprve přiblížit k vykreslovanému úseku. Protože je nutné, abychom postupovali po krocích dp, vyjdeme z přecházejícího algoritmu a odlehčíme ho o výpočet obsahu. Tento úsek, kde $da < dBegin$, je před zobrazenou částí a nebude tedy viditelný. Kdybychom v tomto přibližovacím algoritmu nepostupovali po krocích dp, ale pouze po vzdálenosti jednotlivých bodů pf, da by neodpovídalo $dBegin$ a docházelo by tedy k nežádoucímu posunu krajního bodu projevujícím se zejména při zobrazení nízkého počtu bodů na display.

3.5.2 Vykreslovací algoritmus

Po provedení přepočítání dat na šířku displeje následuje jejich zobrazení. Uvnitř vykreslovacího bufferu máme požadovaný počet sloupců. Nyní je třeba přemapovat hodnoty jednotlivých sloupců na velikost displeje. Hledanou výšku sloupce v pixelech tedy získáme:

$$v_n \doteq (buf(n) - min(buf)) * \frac{height}{max(buf) - min(buf)} \tag{3.8}$$



$$
\begin{array}{lcll}
S_0 & = & 0, dBegin = 0, & pf = 4, \quad dda = 4 \\
S_n & = & S_{n-1} + pf\frac{ppa+pa}{2} & \\
S_1 & = & S_0 + 4\frac{540+548}{2} = 2176 & pf = 10, \quad dda = 14 \\
S_2 & = & S_1 + 10\frac{548+549}{2} = 7661, & pf = 3, \quad dda = 17 \\
S_3 & = & S_2 + 3\frac{549+554}{2} = 9315.5, & pf = 19, \quad dda = 36 \\
S & = & S_{n-1} + (pf - (dda - dp))\frac{ppa+alt}{2} & \\
S & = & S_3 + (19 - (36 - 17,5))\frac{554+553.973}{2} = 9592.5, & pf = 18.5, \quad dda = 36 \\
v & = & \frac{S}{dp} = \frac{9592.5}{17.5} = 548.14 &
\end{array}
$$

Obrázek 3.8: Příklad jednoho kroku přepočtu

Kde buf je vykreslovací buffer, $height$ je výška displeje v pixelech, $min(buf)$ je minimální hodnota v buf, $max(buf)$ je maximální hodnota v buf, $buf(n)$ je hodnota sloupce, kde n je pořadové číslo aktuálního sloupce. Výsledkem je v_n v pixelech, přemapované tak, že $min(buf)$ odpovídá $v_n = 0$ a $max(buf)$ odpovídá $v_n = height$. Při zobrazování je třeba dbát aby rozdíl $max(buf) - min(buf)$ nebyl příliš malé desetinné číslo blížící se zaokrouhlovací chybě.

3.5.3 Zoomování

Změnu úrovně detailů provedeme jednoduchou transformací mezí vykreslování $dBegin$ a $dEnd$. Pro další výpočty si definujeme délku zobrazovaného úseku jako $dl = dEnd - dBegin$. Přiblížení pohledu jde realizovat tak, že dl rozdělíme na čtvrtiny a k $dBegin$ tuto čtvrtinu přičteme, naopak od $dEnd$ tuto čtvrtinu odečteme.

V případě oddalování pohledu je situace mnohem složitější. Zde dl rozdělíme na polovinu a od $dBegin$ ji odečteme, naopak k $dEnd$ ji přičteme. Může se však stát, že dojde k překročení jedné z krajích mezí. Minimální krajní mez je rovna 0, maximální krajní mez nazveme $dFEnd$ a v našem popisovaném případě se jedná o celkovou délku trasy. Nejprve prověříme například minimální mez, tj. zda $dBegin > 0$ a pokud není tato podmínka splněna, je třeba nastavit $dBegin$ na 0. Musíme ale provést i korekci $dEnd$, jinak by došlo k nežádoucí změně velikosti zobrazované oblasti. Proto $dEnd$ posuneme o tolik,

kolik byla hodnota $-dBegin$. Situace s mezí $dEnd$ je velice obdobná. Liší se podmínkou $dFEnd < dEnd$, $dBegin$ se pak posune o tolik, o kolik přesahuje $dEnd$ za $dFEnd$ a $dEnd$ se nastaví na $dFEnd$.

Poslední situací, která může nastat, je překročení obou mezí. Tehdy je třeba nastavit $dEnd$ na $dFEnd$ a $dBegin$ na 0.

3.5.4 Posun a ukazovátko

Posun zobrazení opět realizujeme jako transformaci mezí vykreslování $dBegin$ a $dEnd$. Velikost posunu získáme tak, že odečteme X složky aktuální a minulé pozice kurzoru. Jako krok posunu opět použijeme dp. V případě, že posun přesáhne některou z mezí, nedochází již k posunu mezí, ale pouze k posunu ukazovátka, které je jinak fixní. Ukazovátko slouží ke zjištění výšky vybraného sloupce v grafu. Jeho hodnota je tedy aktuálně vybraným sloupcem, který ukazuje na hodnotu sloupce v buf. Posun mezí není možný, pokud by $dBegin$ po posunutí byl nižší než 0 nebo $dEnd$ větší jak $dFEnd$. Tehdy posuneme jen $dBegin$ na 0 nebo $dEnd$ na $dFEnd$ a umožníme posun ukazovátka.

3.5.5 Porovnávání

Zobrazení dvou trasových souborů je řešeno tak, že jsou obě trasy vykresleny přes sebe s průhledností. Každá trasa má vlastní meze vykreslování, které jsou vzájemně sesazeny tak, aby byly ve stejných sloupcích. Při zoomování či posunu je tedy třeba měnit velikost těchto mezí tak, aby nedocházelo ke vzájemnému posunu tras. Pokud bychom aplikovali posun na každou trasu zvlášť a obě společně, došlo by při dosažení krajní meze na jedné trase k zablokování jejího posunu, ale pokračovalo by posouvání trasy druhé a došlo by k rozejití společného úseku. Proto je třeba zamezit dalšímu samostatnému posouvání obou těchto tras a to tak, že jakmile dojde k přesáhnutí některé z mezí i jen jednoho souboru, zablokujeme další posun. Obdobná situace je i v případě zoomování, kde by při oddálení došlo k přesáhnutí jedné či obou mezí. Pak by byly zobrazeny obě trasy ve zcela odlišném měřítku. Abychom mohli odlišit společné úseky trasy od zbytku trasy, vytvoříme pomocný buffer, ve kterém si poznačíme, zda je daný sloupec součástí společného úseku či nikoliv.

3.6 Zobrazení mapy

Aplikace dále umožňuje zobrazení zaznamenané trasy bez mapového podkladu. Aby mapa byla v kilometrovém měřítku, je potřeba přepočítání zeměpisné šířky a délky ve stupních na (kilo)metry a následná transformace těchto souřadnic pro zobrazení na displeji.

Abychom mohli také na mapě využívat zoom a posun pohledu, definujeme si krajní meze zobrazení mapy. Oproti grafu jsou zde čtyři krajní body tj. nejjižnější S, nejsevernější N, nejvýchodnější E a nejzápadnější W zobrazená pozice.

Nejprve určíme vzdálenost $lpix$ v metrech odpovídající jednomu pixelu. Použijeme následujících vztahů:

$$\begin{aligned} dx &= E - W \\ dy &= N - S \end{aligned} \tag{3.9}$$

$$\begin{aligned} ly &= f(N, W + \frac{dx}{2}, S, W + \frac{dx}{2}) \\ lx &= f(S + \frac{dy}{2}, W, S + \frac{dy}{2}, E) \end{aligned} \tag{3.10}$$

$$\begin{aligned} lpixX &= \frac{lx}{width} \\ lpixY &= \frac{ly}{height} \end{aligned} \tag{3.11}$$

$$lpix = Max(lpixX, lpixY) \tag{3.12}$$

N, E, S, W jsou krajní meze. Funkce f vypočítá vzdálenost v metrech dvou pozic. Dále pak určíme vzdálenost západ – východ jako dx ve stupních a lx v metrech. Vzdálenost sever – jih dy je ve stupních a ly v metrech. Jakmile máme vzdálenosti v metrech, můžeme spočítat vzdálenosti odpovídající 1pixelu – horizontální $lpixX$ a vertikální $lpixY$. Použijeme šířku $width$ a výšku $height$ displeje. Aby byla mapa v měřítku, musíme zajistit shodnou vzdálenost na pixel v horizontální i ve vertikální rovině. Výslednou $lpix$ určíme jako maximum z těchto hodnot. Pro přemapování stupňů na pixely budeme potřebovat přemapovávací koeficienty:

$$\begin{aligned} pixX &= lpix\frac{dx}{lx} \\ pixY &= lpix\frac{dy}{ly} \end{aligned} \tag{3.13}$$

Tím, že jsme sjednotili vzdálenost na pixel, dojde k rozejití mezí. Tedy 2 z nich již nebudou odpovídat krajům obrazovky, ukazovátko nebude zaměřeno na hledanou pozici a mapa bude při změně zoomu „odjíždět“ do strany. Proto je třeba provést kompenzaci a posunout vykreslování o určitý počet pixelů:

$$\begin{aligned} rx &\doteq \frac{dx}{pixX} \\ ry &\doteq \frac{dy}{pixY} \end{aligned} \tag{3.14}$$

$$\begin{aligned} offsetX &= -\frac{rx - width}{2} \\ offsetY &= ry - height2 \end{aligned} \tag{3.15}$$

Výsledné souřadnice bodu na obrazovce pro jednotlivé body p můžeme získat již poměrně jednoduchým výpočtem:

$$\begin{aligned} x &= offsetX + \frac{p.lon - W}{pixX} \\ y &= offsetY + height - \frac{p.lat - S}{pixY} \end{aligned} \tag{3.16}$$

Samotné znázornění trasy na obrazovku provedeme vykreslením jednotlivých úseček mezi těmito body. Protože na mapě lze zároveň zobrazit i trasy, které nemají žádný společný úsek, budeme vždy vykreslovat obě trasy. Na rozdíl od zobrazení grafu nebudeme předpokládat žádné zarážky, proto realizace posunu a zoomování je mnohem jednodušší.

Kapitola 4

Implementace aplikace

Vzhledem k rozsáhlosti implementační části si v této kapitole popíšeme pouze některé zajímavější části tvorby GUI aplikace *GPSTraces*. Pro správný chod aplikace bylo potřeba naimplementovat řadu pomocných tříd, které mají za úkol korektní spuštění aplikace a jsou nezbytné pro běh grafického rozhraní. Třída `CGPSTracesAppUi` implementuje základ uživatelského rozhraní. Umožňuje vytvoření a správu jednotlivých oken aplikace a řídí přepínání mezi nimi. Dále je na ní napojena třída `CGPSTracerEngine`, která vytváří rozhraní mezi okny a jednotlivými specializovanými třídami pro příjem GPS, ukládání a načítání GPX souborů. Dále má na starosti ukládání a načítání nastavení aplikace.

Okna aplikace se vždy skládají z instancí dvou tříd. Samotné okno aplikace je odvozeno od třídy `CAknView`. Tato třída řídí vytváření ovládacích tlačítek a menu. Jednotlivé ovládací prvky jsou pak umístěny v třídě odvozené od `CCoeControl`. Zajímavostí je, že její instance existuje pouze tehdy, je-li okno aktivní, proto je nutné všechny potřebné proměnné, které mají být přítomné i během neaktivního okna, ukládat v třídě první nebo v enginu aplikace.

Aplikace je tvořena ze třech základních oken:

- `CGPSTracesLBView` – zobrazení aktuálních údajů GPS a ovládání záznamu
- `CGPSTracesContainerView` – načítání GPX souborů, vizualizace zaznamenaných dat, porovnávání
- `CGPSTracesSettingsView` – konfigurace uživatelského nastavení

Nastavení uživatelských parametrů je realizováno pomocí *settinglistu* `CGPSTracesSettings`. S tím také souvisí pomocná třída `TGPSTracesSettingsSettings`, která na sebe dočasně váže hodnoty settinglistu.

Nastavení je ukládáno do binárního souboru `settings.dat`, který je umístěn v privátní složce aplikace. Načítání dat z tohoto souboru je realizováno v konstruktoru enginu a ukládání při ukončení aplikace. Do souboru se mimo hodnot nastavení ukládá také jeho verze a ukončovací sekvence. Díky tomu je možné rozpoznat odlišnou verzi souboru či poškozený soubor. V takovém případě soubor odstraní a zabrání tak pádu aplikace způsobenou neznámým formátem souboru.

4.1 GPS engine

V kapitole 2.1.1 jsme si popsali základní funkci location API. V aplikaci obstarává získávání polohy třída `CGPSTracerPositionRequestor`. Protože požadavky na získání GPS

pozice mohou trvat rozdílnou dobu v řádu jednotek až desítek sekund, je navržena jako asynchronní proces (tj. odvozena od třídy `CAktive`). Díky tomu nebude aplikace po dobu čekání na odpověď od pozičního serveru blokována. V metodě `ConstructL()` provedeme inicializaci session a subsession a nastavení parametrů aktualizací. Budeme vyžadovat periodické aktualizace v intervalu 1s, což je minimální možný interval, ve kterém může poziční server zasílat odpovědi klientovi. Požadujeme také částečné aktualizace, které obdržíme po vypršení požadavku klienta na poziční server. Tento limit nastavíme na 15 sekund a při standardních aktualizacích se nijak neprojeví, svojí roli ale sehraje při výpadku GPS signálu. O spuštění periodických aktualizací se postará metoda `Start()`, kde se provede nastavení asynchronního požadavku. Po dokončení žádosti server zavolá metodu `RunL()` našeho AO. Prověřením stavového slova AO `iStatus` zjistíme úspěšnost našeho požadavku:

- `KErrNone` – vše proběhlo v pořádku, získali jsme aktuální pozici, dále zpracujeme `PositionInfoUpdatedL`.
- `KPositionPartialUpdate` – došlo k výpadku GPS signálu, máme k dispozici aktuální čas.
- `KErrCancel` – požadavek na získání pozice byl zrušen.
- `KPositionQualityLoss, KErrTimedOut` – došlo ke ztrátě signálu, vypršení časového limitu požadavku. Tyto chyby nastanou pouze tehdy, pokud nemáme vyžádáno zasílání částečných aktualizací.
- `KErrAccessDenied, KErrUnknown` – kritické chyby.

Dojde-li k nějaké chybě, nebudeme o další aktualizaci žádat okamžitě, ale po vypršení 15 sekundového ochranného časového intervalu. K tomu slouží metoda `Wait()`, která spustí časovač, po jehož uplynutí dojde k obnovení aktualizací provedením `Start()`. `PositionInfoUpdatedL()` ověří, zda data vrácená pozičním serverem jsou validní, tj. nejsou typu `EPositionUpdateUnknown`.

GPS engine mimo vlastních získaných dat typu `TPositionInfo` či `TSatelliteInfo` předává také stavové slovo `iPositionState`. K tomuto účelu slouží metoda `PositionUpdatedL()`, která předzpracuje získaná data a nastaví příznaky `iPositionState` a následně přepošle pozorovateli `MPositionObserver`.

Mezi příznaky stavového slova, které jsou nastavovány v této metodě, patří:

- `EPositionOK` značí, že v získaných datech jsou platné údaje o aktuální pozici.
- `EPositionKO` indikuje stav výpadku GPS signálu, kdy nejsou dostupné platné informace o pozici.
- `ESatellite` pokud je nastaven, jedná se o rozšířený režim `TSatelliteInfo`, jinak pouze o základní režim `TPositionInfo`.
- `ELastKnownPos` jedná se o poslední známou pozici.
- `EModuleChanged` je generováno při změně aktivního GPS modulu, změně úrovně předávaných informací z `TPositionInfo` na `TSatelliteInfo` nebo opačně.

Stavové slovo obsahuje i další příznaky, které nejsou nastavovány v GPS enginu. Tyto jsou využívány generátorem GPX souborů:

- `ELogSpeed` – požaduje ukládání elementu speed
- `ELogSatInfo` – požaduje záznam údajů o přesnosti
- `ENewTrackFile` – požaduje vytvoření nového trasového souboru

Posledním příznakem je:

- `EIgnoreCalc` – slouží k blokování výpočtů pro zobrazení dat

Nyní si podrobněji popíšeme chování metody `PositionUpdatedL()`. V rámci základních informací není dostupný přesný GMT čas satelitu a polohy poskytnuté v tomto režimu jsou nastaveny na lokální čas mobilního telefonu, který ale neodpovídá času GMT. Podle definice formátu GPX souborů je požadováno ukládání v čase GMT. Pokud máme k dispozici pouze základní režim, nelze tuto podmínku zajistit. V rozšířeném režimu máme k dispozici přesný GMT čas poskytnutý satelitem. Pokud je tedy dostupný rozšířený režim, což lze zjistit otestováním `iPosInfoBase->PositionClassType() & EPositionSatelliteInfoClass`, pak můžeme časovou značku pozice nahradit přesným časem satelitu. Tento čas není závislý na lokálním čase telefonu, proto odpadají problémy spojené se změnou lokálního času během činnosti aplikace. Rozšířený režim budeme dále signalizovat nastavením příznaku `ESatellite`.

Po vykonání korekce času otestujeme platnost poskytnuté pozice. To můžeme provést tak, že ověříme hodnotu zeměpisné délky či šířky na nedefinovanou hodnotu `NaN`. Jestliže je hodnota definovaná, získali jsme platné informace o poloze. Příznak `ELastKnownPos` je nastaven při spuštění příjmu GPS signálu a po příchodu první platné aktuální pozice je vynulován. Pokud byla minulá pozice označena příznakem `EPositionKO` (výpadek GPS signálu), vynulujeme tyto příznaky a nastavíme platnou pozici `EPositionOK`. Je-li hodnota zeměpisné šířky nedefinovaná, došlo k výpadku GPS signálu. Nastavíme příznak `EPositionKO` a zrušíme případné nastavení příznaku `EPositionOK`. Aby mohla aplikace zobrazovat údaj o poloze i v době výpadku signálu, pamatujeme si poslední platnou pozici a tu nyní nahradíme místo neplatných dat.

Máme-li ověřenou aktuálnost dat, zbývá otestovat, zda nedošlo ke změně použitého GPS modulu. V případě prvního volání zjistíme, zda můžeme požadovat získávání rozšířených informací. To by ovšem vyžadovalo přepnutí datových struktur a získaná data by již nebyla aktuální. Proto musí pozorovatel `MPositionObserver` nejprve provést zpracování polohových dat. Samotné ověření provedeme podle postupu popsaného v kapitole 2.1.1. Došlo-li tedy ke změně, nastavíme příznak `EModuleChanged`, který se odešle s příštími získanými daty a po jejich odeslání bude opět vynulován. Nakonec přepneme datové struktury pro získání další pozice, pokud modul vyžaduje jejich změnu.

Třída `MPositionObserver` slouží ke komunikaci s příjemcem GPS dat. Metoda `PositionUpdatedL()` slouží předávání získaných GPS dat a stavového slova. Dále má toto třída metody `MessageL()` a `ErrorL()`, které slouží k signalizaci zpráv a kritických chyb.

4.2 Generování GPX souborů

Pro vytváření GPX souborů ve tvaru popsaném v kapitole 2.4 slouží třída `CGPSTracesGPXgen` odvozená od `CBase`. Pro započetí záznamu je třeba zavolat metodu `OpenL()`, která jako svůj jediný parametr očekává čas, který bude uložen v hlavičce GPX jako název trasy, čas vytvoření a je také z něho odvozen název souboru. Interně volá `WriteHeadL()`, která do nově vytvořeného souboru zapíše hlavičku GPX.

K přidání trasového bodu do souboru slouží `WritePositionL()`, která očekává data poskytovaná GPS enginem. Dle stavového slova `aStatus` se určí, které informace mají být zaznamenány. Neposkytuje-li GPS modul `TPositionSatelliteInfo`, zaznamenávají se pouze základní informace o poloze (`lon`, `lan`, `<ele>`, `<time>`). Nadmořská výška se ukládá s přesností na 1 desetinné místo, čas a datum se ukládá ve formátu ISO801 s rozlišením na sekundy. Ačkoli GPS modul poskytuje čas s přesností na milisekundy, zaznamenání desetinné části není nutné, protože data jsou poskytována v přesných periodických (sekundových) intervalech, proto se vzájemně neliší. V případě, že GPS engine získal strukturu typu `TPositionSatelliteInfo`, můžeme zapsat dodatečné informace o rychlosti či přesnosti. Pokud je ve stavovém slově nastaven bit `ELogSpeed`, je vyžadováno ukládání rychlosti. To že máme k dispozici *SatelliteInfo* neznamená, že lze určit rychlost – její údaj může být nastaven na `NaN` a v takovém případě se element `<speed>` u této položky neuloží. Pokud je ve stavovém slově povoleno ukládání informací o přesnosti nastavením příznaku `ELogSatInfo`, ukládá se počet aktivních družic `<sat>`, který je vždy dostupný. Dále se zaznamenává horizontální `<hdop>`, vertikální `<vdop>` a polohová `<pdop>` přesnost. Tyto údaje ale nemusí být vždy určitelné – také mohou nabývat hodnoty `NaN` a takový údaj nelze ukládat. Údaje o přesnosti času TDOP a celkového GDOP nejsou zaznamenávány, protože GPX ukládání těchto údajů dle specifikace[17] neumožňuje. K fyzickému uložení dat dojde vždy až po naplnění vyrovnávací paměti pro zápis.

Pokud není určeno jinak, soubory se standardně ukládají do adresáře `C:\Data\`. Pracovní složku lze změnit pomocí `SetWorkingDir()`.

Zápis do souboru ukončíme metodou `Close()`, která má na starosti zapsání patičky, tj. uzavření otevřených elementů. Následně proběhne `CommitL()`, tj. uložení dat z vyrovnávací paměti a díky metodě `CloseNoCommit()` k bezpečnému uzavření souboru. Třída obsahuje také počitadlo trasových bodů. Pokud soubor nemá alespoň minimální počet trasových bodů, dojde k jeho smazání. Množství lze nastavit `SetTrkptMinLimit()` a lze jej zjistit `GetTrkptMinLimit()`.

Další metodou je `Reset()` sloužící k uzavření a otevření nového souboru. Rozlišuje případ, kdy došlo k zápisu nějakých položek do souboru. Pak je soubor uzavřen pomocí `Close()`. Aby se předešlo zbytečnému mazání souborů a tím se omezil počet zápisů do flash paměti, dojde pouze k vyprázdnění vyrovnávací paměti a vytvoření nové hlavičky souboru.

`WriteCommentL()` slouží k zápisu komentáře do souboru. Používá se k indikaci výpadků signálu, druhu přerušení záznamu atd.

4.3 Načítání GPX souborů

Protože GPX soubory jsou postaveny na XML formátu, využijeme pro jejich zpracování třídu `CParser`, která je založena na SAX parseru. Načítání souborů realizujeme pomocí třídy `RFile` po 4KB blocích do vyrovnávací paměti, která je následně zpracována parserem. Protože načítání souborů je dlouhá operace, budeme ji provádět asynchronně za pomocí AO. Aby uživatel byl informován o aktuálním stavu načítání, zobrazíme *process dialog*, ve kterém budeme signalizovat procentuální stav načtených dat. Třída `CGPSTracesGPXparser` má na starosti mimo načítání a parsování také výpočty statistik. Funguje také jako datové uložiště těchto informací.

Ke sledování aktuálního stavu při parsování slouží stavové slovo `TGpxElement`, kde se nastavením jednotlivých bitů udává aktuální zanoření elementů. Díky tomu lze odlišit význam elementů se shodnými názvy a kontrolovat správný výskyt elementů.

4.4 Pomocné funkce GPS

Třída `CGPSTracerLocationEngine` obsahuje metody k určování vzdálenosti GPS souřadnic a zjištění směru pohybu. `GetDistance()` využívá vzdálenost standardního location API[9]. `GetDistanceWithAltitude()` při výpočtu uvažuje i nadmořskou výšku a je postavena na algoritmu popsaném v kapitole 3.2. Při implementaci byl brán ohled na rychlostní optimalizaci výpočtů. `GetDirectionL()` slouží k určení směru z výchozí pozice do nové. Výsledkem je textová prezentace směru uložená v deskriptoru. Poslední metodou poskytovanou touto třídou je `GetDegreesString()`. Jejím úkolem je naformátování hodnoty ve stupních do zvoleného textového formátu dle parametru `type`. `KTDLong` určuje dlouhý formát ve tvaru `dd°mm'ss.sss"`. `KTDShort` má tvar `ddd°mm.mmmm'` a `TDDeg` slouží jen pro zobrazení samostatných stupňů.

4.5 Obsluha tlačítek hlasitosti

K zajímavostem u telefonů s OS Symbian rozhodně patří, že se všechny klávesy neobsluhují vždy shodným způsobem. Telefony se Symbian 5th mají obvykle velmi málo kláves (výjimkou je Nokia N97, která má úplnou alfanumerickou klávesnici). Obsluha numerických kláves a většiny ovládacích kláves se provádí pomocí metody `HandleKeyEventL()`, která však ovládání tlačítek pro změnu hlasitosti nezachytí. Tyto tlačítka patří do skupiny takzvaných multimediálních kláves, které jsou přístupné pomocí Bluetooth Remote Control API. Zachycení těchto kláves pak proběhne v `MrccatoCommand()`. V aplikaci jsou tato tlačítka využita k změně zoomu.

Kapitola 5

Závěr

Aplikace *GPSTraces* je navržena tak, aby mohla být spuštěna na pozadí a uživatel v průběhu záznamu nemusel do ní nijak zasahovat. Tomu napomáhá možnost automatického startu GPS a záznamu trasy po spuštění aplikace. Díky volitelnému automatickému přerušování trasy při zastavení nemusí uživatel zasahovat do záznamu.

Metody použité v aplikaci nám umožňují vytváření trasových souborů, které obsahují pouze nezbytný počet trasových bodů, přesto dobře popisují zaznamenávanou trasu. To je zajištěno filtrováním trasových bodů ihned po jejich získání z GPS modulu – dle různých kritérií, jako je přesnost či vzájemná poloha. Zaznamenaná data lze zobrazit ve formě grafu různých závislostí a můžeme díky němu snadno najít libovolnou část trasy a pohodlně zjistit hodnotu veličiny k hledané poloze. Důraz je kladen na vzájemné porovnávaní tras a zobrazení společných úseků v grafu. K získání přehledu o dané trase také slouží možnost zobrazení tras pomocí jednoduchého mapového náhledu. Dále můžeme zobrazit spoustu užitečných souhrnných informací o trase. Aplikace *GPSTraces* je navržena jako vícejazyčná. Výběr jazyka je proveden automaticky dle nastavení telefonu při instalaci aplikace.

Základní testování aplikace jsem provedl v emulátoru dodaném společně se 5th Edition SDK, který umožňuje i simulaci GPS modulu. Protože GPS data poskytované simulátorem se nemění, bylo třeba aplikaci otestovat v terénu. Ke sběru dat jsem měl k dispozici mobilní telefony Nokia 5800 a zapůjčený Nokia N97 v různých podmínkách příjmu. Při testování na druhém telefonu byla odhalena chyba generování a zpracování GPX souborů, kde vlivem rozdílného národního nastavení byly v číselných hodnotách použity desetinné čárky na místo očekávaných desetinných teček. Tento problém se projevoval pádem aplikace a v aktualizované verzi je již správné chování zajištěno.

Přestože jsem se snažil odhalit a vypořádat se s co největším množstvím problémů, aplikace obsahuje několik známých chyb, které jsou způsobeny neočekávanými událostmi při provozu telefonu. Pokud se při spuštěném záznamu trasy vybije telefon, může dojít k poškození vytvářeného souboru, nebo dokonce k jeho úplnému zmizení. Je to způsobeno tím, že operační systém násilně ukončí všechny aplikace. Proto není tento soubor uzavřen. Další chyba vznikne, pokud porovnávání zrušíme během zobrazeného wait dialogu a následně načteme jiný GPX soubor. Tehdy může – ve velmi výjimečných případech – dojít k pádu vlákna zajišťujícího porovnávání a již nebude možné bez restartu aplikace znovu použít porovnávací funkci. Záznam trasy v tomto případě není ohrožen.

Při další práci na aplikaci by bylo možno zdokonalit zobrazení výsledků porovnávání. Aktuální verze umí zvýraznit pouze jeden společný úsek. Pro nalezení dalšího je třeba vyvolat porovnávací funkci. Aby bylo možné zobrazit všechny společné úseky zároveň, museli bychom udělat úpravy ve vykreslovacím engine. Také by bylo zajímavé přidat online po-

rovnávání aktuální trasy se zaznamenanou. Zde by ovšem bylo nutné zajistit co nejmenší výpočetní náročnost tak, aby byly minimalizovány energetické nároky aplikace.

Programování v jazyce Symbian C++ bylo pro mě zejména na počátku velmi obtížné, z důvodu velkého množství odlišností od standardu. Práce s vývojovým prostředím Carbide C++ byla často zpestřena některými jeho nedostatky – jako chybějícími možnostmi editace grafického návrhu či chybami v generovaném kódu. Většinu těchto problémů lze s větším či menším úsilím obejít, ať již aplikováním příslušného patche či dodržováním správných postupů při využívání UI designeru. Velkým problémem jsou kritické chyby typu `Panic`, které způsobí pád aplikace. Jejich příčiny se odhalují velmi složitě i za použití debuggeru, protože často nelze odhalit přesné místo vzniku chyby. To je navíc umocněno přílišnou obecností některých chyb. Nepříznivým faktorem je také poměrně dlouhé načítání emulátoru.

Literatura

[1] *Elipsoid* [online]. Wikipedia, the free encyclopedia, 2008-10-28 [cit. 2010-04-17].
URL http://cs.wikipedia.org/wiki/Elipsoid

[2] *GPS-System* [online]. kowoma.de, 2009-04-19 [cit. 2010-04-18].
URL http://www.kowoma.de/en/gps/

[3] *GPS system information* [online]. UNITED STATES NAVAL OBSERVATORY (USNO), 2009-08-20 [cit. 2010-04-18].
URL ftp://tycho.usno.navy.mil/pub/gps/gpssy.txt

[4] *Block II satellite information* [online]. UNITED STATES NAVAL OBSERVATORY (USNO), 2009-11-20 [cit. 2010-04-18].
URL ftp://tycho.usno.navy.mil/pub/gps/gpsb2.txt

[5] *Sférická soustava souřadnic* [online]. Wikipedia, the free encyclopedia, 2009-12-27 [cit. 2010-04-17].
URL
http://cs.wikipedia.org/wiki/Sf%C3%A9rick%C3%A1_soustava_sou%C5%99adnic

[6] *Assisted GPS* [online]. Wikipedia, the free encyclopedia, 2010-04-15 [cit. 2010-04-18].
URL http://en.wikipedia.org/wiki/AGPS

[7] *Global Positioning System* [online]. Wikipedia, the free encyclopedia, 2010-04-17 [cit. 2010-04-18].
URL http://en.wikipedia.org/wiki/GPS

[8] Babin, S.: *Developing Software for Symbian OS: an introduction to creating smartphone application in C++*. John Wiley & Sons, Chichester, 2006, ISBN 0-470018-45-3.

[9] Forum Nokia: *S60 5th Edition C++ Developer's Library v2.1 » Location API* [online]. [cit. 2010-04-01].
URL http://library.forum.nokia.com/

[10] Forum Nokia: *S60 5th Edition C++ Developer's Library v2.1 » New features in S60 5th Edition* [online]. [cit. 2010-04-01].
URL http://library.forum.nokia.com/index.jsp?topic=/S60_5th_Edition_Cpp_Developers_Library/GUID-CA617A81-53B3-4B19-A69E-32DF920A26C5.html

[11] Rapant, P.: *Družicové polohové systémy* [online]. VŠB-TU Ostrava, 2002, ISBN 80-248-0124-8.
URL http://gis.vsb.cz/dokumenty/dns-gps/at_download/file

[12] Symbian: *Essential S60: Creating Location-Aware Applications* [online]. Symbian Software Limited, 1st edition, 2008-11-30 [cit. 2010-04-01].
URL http://sw.nokia.com/id/9ef38bb4-4407-4f1a-9f8f-5319acd998ac/S60_Essentials_LBS_v1_0_en.pdf

[13] Symbian Wiki: *Descriptors (Fundamentals of Symbian C++)* [online]. Symbian, 2010-03-11 [cit. 2010-04-05].
URL http://developer.symbian.org/wiki/index.php/Descriptors

[14] Symbian Wiki: *Active Objects (Fundamentals of Symbian C++)* [online]. Symbian, 2010-03-11 [cit. 2010-05-11].
URL http://developer.symbian.org/wiki/index.php/Active_Objects_%28Fundamentals_of_Symbian_C%2B%2B%29

[15] Symbian Wiki: *Threads, Processes, and IPC (Fundamentals of Symbian C++)* [online]. Symbian, 2010-03-11 [cit. 2010-05-11].
URL http://developer.symbian.org/wiki/index.php/Threads,_Processes,_and_IPC_%28Fundamentals_of_Symbian_C%2B%2B%29

[16] Tomuta, L.: *Do a little more with the UI Designer* [online]. Forum Nokia Blog, 2009-01-31 [cit. 2010-04-11].
URL http://blogs.forum.nokia.com/blog/lucian-tomutas-forum-nokia-blog/2009/01/31/do-a-little-more-with-the-ui-designer

[17] TopoGrafix: *GPX: the GPS Exchange Format* [online]. Version 1.1. [cit. 2010-02-28].
URL http://www.topografix.com/gpx.asp

[18] Veness, C.: *Calculate distance, bearing and more between Latitude/Longitude points* [online]. Movable Type Scripts, [cit. 2010-04-17].
URL http://www.movable-type.co.uk/scripts/latlong.html

[19] Veness, C.: *Vincenty formula for distance between two Latitude/Longitude points* [online]. Movable Type Scripts, [cit. 2010-04-17].
URL http://www.movable-type.co.uk/scripts/latlong-vincenty.html

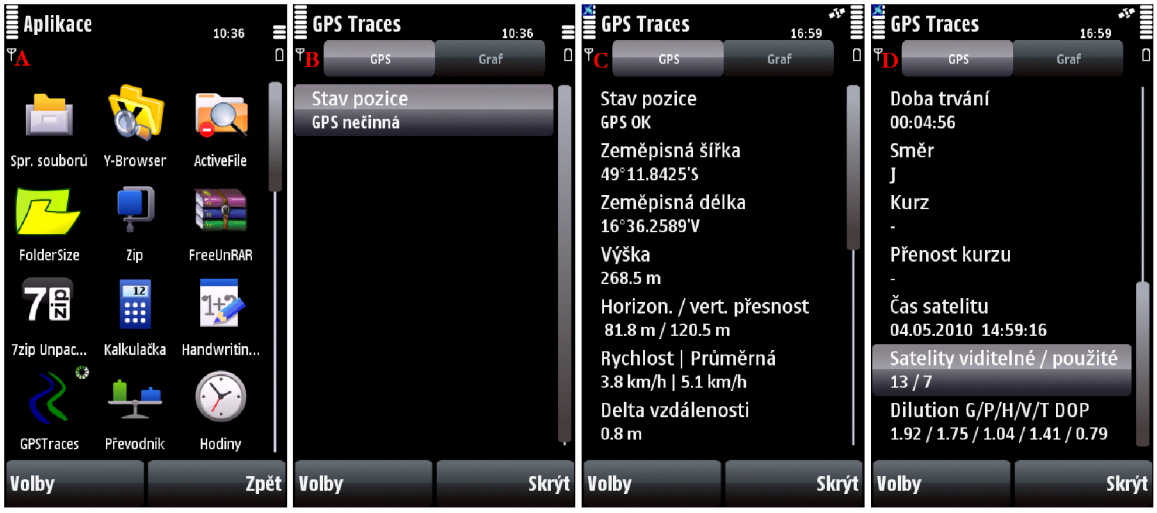
Příloha A

Obsah DVD

- `README` – soubor podobný této příloze
- `workspace\GPSTraces` – obsahuje projekt pro vývojové prostředí Carbide C++
  - `data` – resource soubory a definice textových řetězců aplikace
  - `gfx` – ikony a grafické prvky aplikace
  - `src` – zdrojové soubory aplikace
  - `group` – projektové soubory
  - `sis` – soubory pro instalaci aplikace v mobilním telefonu
    - `GPSTraces.sisx` (self-signed)
    - `GPSTraces.sis` (nepodepsaný)
- `doxygen` – generovaná programátorská dokumentace
- `gpx` – složka s ukázkovými GPX soubory
- `poster` – plakát
- `screens` – snímky obrazovky aplikace
- `install` – instalační soubory
  - `README` – popis instalace
- `text` – text práce ve formě *pdf*
  - `tex` – zdrojové kódy pro LaTeX
- `video` – krátké ukázkové video aplikace

Příloha B

Návod k aplikaci



Obrázek B.1: Ukázka aplikace: Ikona aplikace, Okno GPS

Po nainstalování aplikace *GPSTraces* do mobilního telefonu ji lze nalézt v menu ve složce Aplikace (obr. B.1.A). Po spuštění *GPSTraces* se zobrazí obrazovka umožňující zapnutí příjmu GPS (obr. B.1.B). To lze provést pomocí menu nebo stiskem položky aktuálního stavu GPS. Po aktivování příjmu GPS se zobrazí další položky (obr. B.1.C,D):

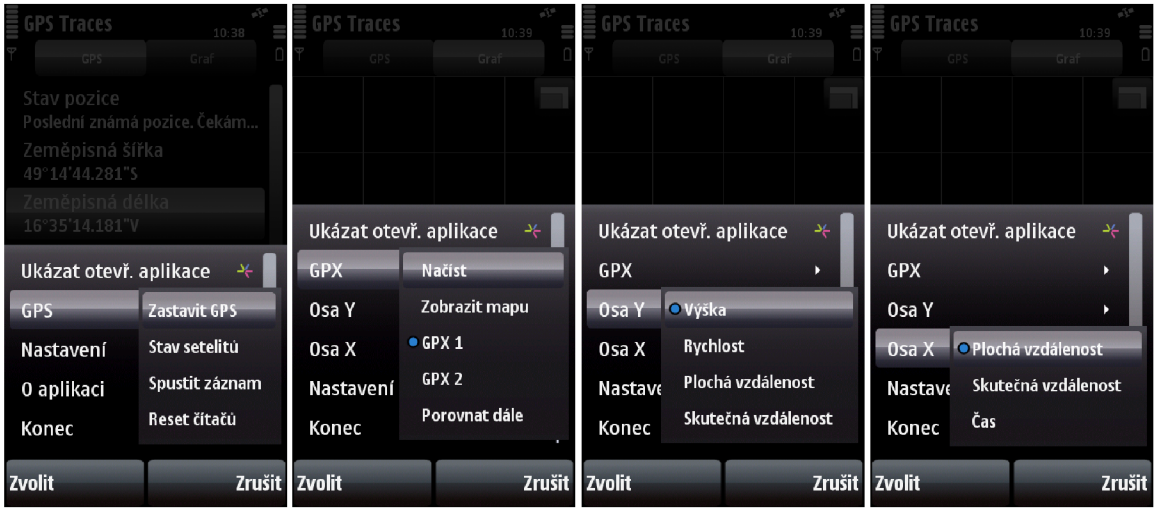
- Zeměpisná šířka a délka – Ukazují aktuální polohu. Stiskem položky lze přepínat zobrazovaný formát.
- Nadmořská výška
- Přesnost pozice v metrech
- Rychlost – Mimo aktuální rychosti zobrazuje také průměrnou rychlost od začátku záznamu či trasy. V závorce může být zobrazena rychlost poskytnutá GPS modulem. Stiskem lze přepnout jednotky (km/h, m/s).
- Rozdíl vzdálenosti od poslední pozice
- Celková vzdálenost – Vzdálenost od počátku záznamu. Stisk mění jednotky (km, m).

- Doba trvání záznamu – Lze přepínat krátký/dlouhý formát data.
- Směr – Směr vypočítaný z aktuální a minulé pozice. Rozlišuje 8 základních směrů.

Jestliže jsou dostupné rozšířené informace, také se zobrazí:

- Kurz – Směr ve stupních poskytovaný GPS modulem (není vždy dostupný)
- Přesnost kurzu
- Aktuální čas satelitu – Přesný GMT čas. Lze přepínat krátký/dlouhý formát.
- Počet satelitů – Satelity viditelné / skutečně použité k výpočtu pozice. Při stisku se zobrazí dialog se stavy družic.
- Geometrická přesnost pozice

Tlačítko *Volby* slouží k zobrazení menu. *Skrýt* minimalizuje aplikaci na pozadí.



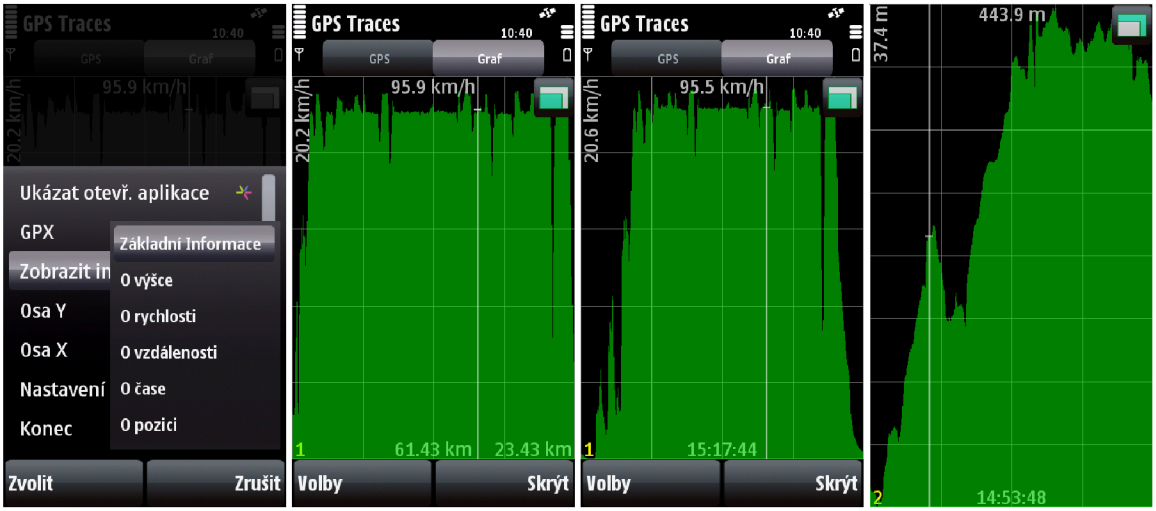
Obrázek B.2: Ukázka aplikace: Menu

Menu aplikace je rozdílné pro každou z obrazovek. V *GPS* režimu lze zapnout či vypnout příjem GPS a záznam. *Reset čítačů* vynuluje celkovou vzdálenost, dobu trvání a průměrnou rychlost. *Nastavení* slouží k zobrazení konfigurační obrazovky.

V režimu *Graf* můžeme načíst GPX soubor. Tlačítka *GPX 1* a *GPX 2* přepínají aktivitu GPX. Potřebujeme-li načíst oba GPX soubory, je nutné před otevřením druhého přepnout aktivitu. *Zobrazení mapy* zrušíme opětovným vybráním některé z možností *Osy X*. Položka *Porovnat dále* slouží k porovnání načtených tras. Další společný úsek se vyhledá opětovným použitím této funkce. *Osa X* a *Osa Y* slouží k výběru závislostí.

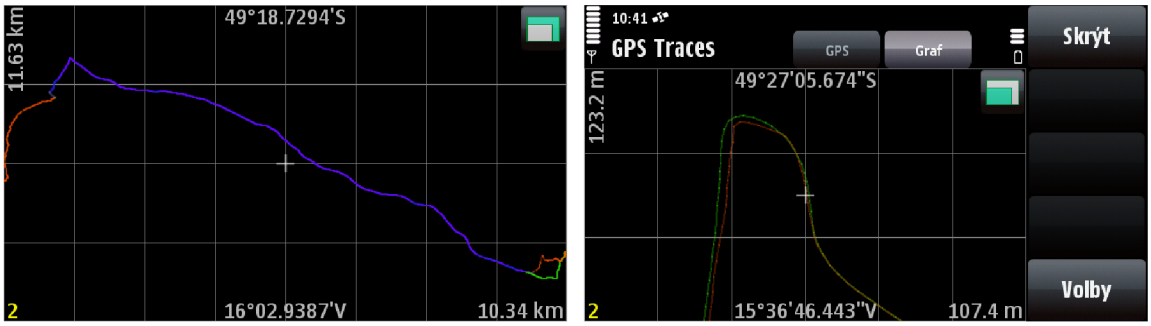
Po načtení GPX souboru se zpřístupní položka *Zobrazit informace*.

Žluté číslo v levém dolním rohu signalizuje aktivní GPX soubor. V pravém dolním rohu se nachází měřítko mřížky ve měru osy X a v levém horním rohu ve směru osy Y. Hodnota uprostřed nahoře udává velikost sloupce vybraného kurzorem. Vzdálenost kurzoru na ose X udává údaj uprostřed dole. Polohu kurzoru lze měnit dotykem obrazovky. Tlačítko v pravém horním rohu slouží k přepnutí zobrazení na celou obrazovku. K zoomování je třeba využít tlačítka na boku telefonu.



Obrázek B.3: Ukázka aplikace: Grafy

Na mapě jsou vždy zobrazeny obě načtené trasy. Zelenou barvou je vyznačena trasa 1, oranžovou barvou trasa 2. Po aplikování porovnání je identifikovaný společný úsek zvýrazněn odstíny modré. Při přiblížení mapy jsou viditelné jednotlivé trasové body jako „světlé“. Lze tedy snadno identifikovat „tmavší“ místa s výpadkem GPS signálu. Kurzor ve tvaru křížku určuje pozici na mapě.

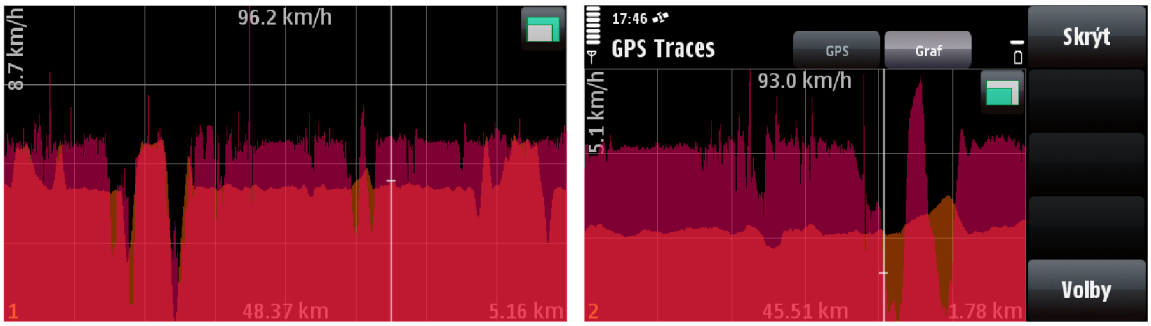


Obrázek B.4: Ukázka aplikace: Mapa – porovnání zaznamenaných tras Brno – Jihlava

Po aplikaci porovnávací fuunkce se v grafu objeví obě trasy se sesazeným začátkem a koncem tras. Pro snadné zjištění rozdílu veličiny ve vybraném bodě stačí přepnout aktivní GPX soubory. Společný úsek je vyznačen odstíny červené. trasa 1 zeleně, trasa 2 modře.

V nastavení je množno změnit mnoho parametrů:

- *GPX adresář* – místo pro ukládání GPX souborů
- *Minimální počet bodů v GPX souboru* – soubory s nižším počtem než zde uvedeným nebudou vytvářeny
- *Automatické spouštení* – Žádné/příjem GPS/záznam trasy
- *Povolení ukládání informací o přesnosti/rychlosti*

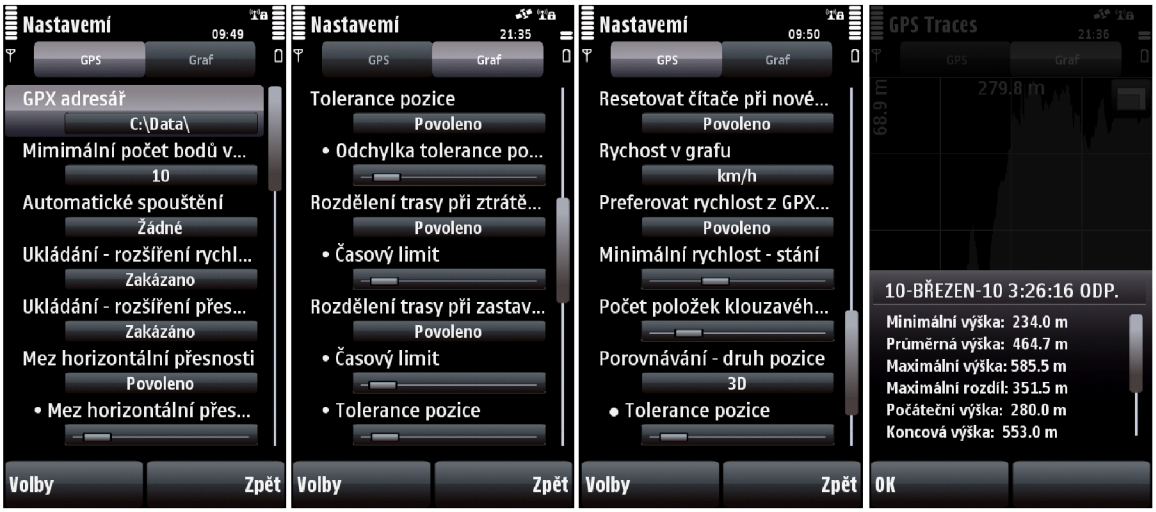


Obrázek B.5: Ukázka aplikace: Graf – porovnání zaznamenaných tras Brno – Jihlava

- *Resetování čítačů při vytvoření nové trasy*

Filtry trasových bodů při záznamu trasy:

- *Mez horizontální přesnoti* – všechny body s přesností v metrech horší než je definováno, budou zanedbány
- *Tolerance pozice*
- *Rozdělení trasy při ztrátě signálu*
- *Rozdělení trasy při zastavení na místě*



Obrázek B.6: Ukázka aplikace: Nastavení

Nastavení grafu:

- *Rychlost v grafu* – jednotky (km/h, m/s)
- *Preferovat rychlost z GPX* – přepíná mezi používáním elementu `<speed>` a vypočítané rychlosti

- *Minimální rychlost* – pokud není dosaženo této rychlosti, nejsou započítávány změny stoupání a klesání
- *Počet položek klouzavého průměru* – určuje z kolika minulých trasových bodů se má hodnota rychlosti získat, pokud je vypočítávána
- *Porovnávání - druh pozice* – určuje, zda porovnávací algoritmus má používat plochou vzdálenost (2D) nebo reálnou vzdálenost s nadmořskou výškou (3D)
- *Porovnávání - Tolerance pozice*
- *Porovnávání - Minimální vzdálenost* – minimální společná vzdálenost v metrech, kratší společné úseky jsou přeskakovány
- *Porovnávání - Rozdíl vzdálenosti* – maximální možná odchylka délky společného úseku tras v procentech