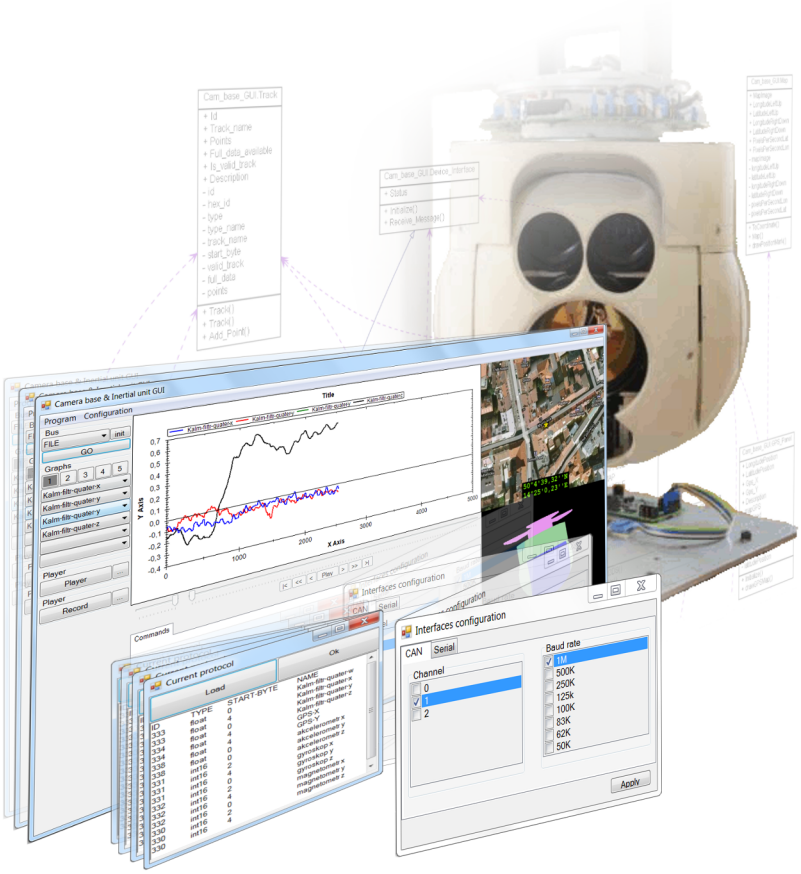
Rád bych na tomto místě poděkoval vedoucímu mé bakalářské práce panu Ing. Martinu Řezáčovi za poskytnuté rady a čas strávený konzultováním problematiky.


Camera base & Inertial unit GUI
Program
Configuration
Bus
int
FILE
GO
Graphs
Kalm-filt-quater-x
Kalm-filt-quater-y
Kalm-filt-quater-z
Player
Record
Title
Y Axis
X Axis
Commands
Current protocol
Load
Ok
Interfaces configuration
CAN
Serial
Channel
Baud rate
1M
500K
250K
125k
100K
83K
62K
50K
Apply

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

BAKALÁŘSKÁ PRÁCE

Grafické rozhraní pro inerciálně stabilizovanou leteckou kamerovou základnu

Autor: Josef Hák

Vedoucí práce: Ing. Martin Řezáč

Praha, 2010

Název práce: Grafické rozhraní pro inerciálně stabilizovanou leteckou kamerovou základnu

Autor: Josef Hák

Katedra (ústav): Katedra řídicí techniky

Vedoucí bakalářské práce: Ing. Martin Řezáč

e-mail vedoucího: rezacma3@fel.cvut.cz

Abstrakt Tato práce pojednává o návrhu a programování grafického uživatelského rozhraní určeného ke komunikaci se stabilizovanou kamerovou základnou pro bezpilotní letoun, která je vyvíjena na Katedře řídicí techniky. Program slouží hlavně ke zpracování a analýze přijímaných dat a k jejich zobrazování formou grafů. Takto přijímá také údaje z GPS přijímače a zobrazuje odpovídající polohu na mapě. Způsob přenosu dat mezi programem a základnou přes sběrnici CAN definuje konfigurovatelný komunikační protokol. Díky tomu je možné program použít i pro komunikaci s jiným zařízením. Program je dále rozšířen o podporu sběrnice RS232 a dokáže ukládat a přehrávat přijímaná data. Zdrojový kód je dokumentován, což má usnadnit jeho další vylepšování a úpravy.

Title: Graphical user interface for inertially stabilized airborne camera platform

Author: Josef Hák

Department: Department of Control Engineering

Supervisor: Ing. Martin Řezáč

Supervisor's e-mail address: rezacma3@fel.cvut.cz

Abstract This paper discusses the design and programming graphical user interface designed to communicate with a stabilized camera platform for unmanned aircraft, which is being developed at the Department of Control Engineering. The program is used primarily to process and analyze received data and displaying it as graph. Thus the program also receives the data from the GPS receiver and displays the appropriate location on the map. Method for transferring data between the program and the platform via the CAN bus is defined with configurable communication protocol. This allows a program to be used to communicate with other devices. The program is further extended to support the RS232 bus and can store and play the taken data. The source code is documented, which should facilitate its further improvement and adjustment.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Josef Hák**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Grafické rozhraní pro inerciálně stabilizovanou leteckou kamerovou základnu**

Pokyny pro vypracování:

Cílem práce je rozšířit již existující grafické uživatelské rozhraní používané k ovládání stabilizované kamerové základny vyvíjené na katedře řídicí techniky.
1. Úpravte stávající grafické rozhraní tak, aby zobrazovalo data jak ze stabilizované základny tak i z přidružené inerciální jednotky. Hlavní okno aplikace bude rozděleno do několika podoken či záložek (Tabs).
2. Návrhněte nové funkce pro ukládání zobrazených dat do souborů, jejich zpětné načtení, zobrazení a přehrávání v grafu pomocí již připraveného OpenGL modelu.
3. Upravte grafické rozhranní tak, aby umožnilo definovat formát komunikačního protokolu na aplikační úrovni (pomocí konfiguračního souboru nebo přímo v GUI).
4. Rozšiřte GUI o možnost sběru dat nejen z CAN sběrnice, ale i RS232 (příp USB či TCP/IP).
5. Implementujte zobrazení dat polohy z GPS senzoru. Data budou zobrazována v XY grafu na jehož pozadí bude obrázek mapy.
6. Důkladně dokumentujte zdrojový kód např. s využitím volně dostupného prostředí DoxyGen.

Seznam odborné literatury:

[1] M. Řezáč, "Návrh řízení pro systém stabilizace optické osy kamerového systému pro bezpilotní letoun", (diplomová práce), ČVUT, 2008.
[2] J. Žoha, "Elektronika pro systém stabilizace optické osy kamerového systému", (diplomová práce), ČVUT, 2008.

Vedoucí: Ing. Martin Řezáč

Platnost zadání: do konce zimního semestru 2010/2011

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 15. 12. 2009

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 27. května 2010 Josef Hák

OBSAH

KAPITOLA
PRVNÍ

ÚVOD

Tato práce je součástí projektu, který se zabývá vývojem stabilizované kamerové základny (dále pouze základna) pro bezpilotní vojenský letoun. Program, jehož návrh a implementace jsou zde popisovány, má sloužit k analýze dat ze senzorů umístěných na základně a k jejímu řízení. Vychází ze starší verze programu, kterou má nahradit. Má mít více funkcí, má být stabilnější a uživatelsky přívětivější než jeho předchůdce.

1.1 Účel grafického rozhraní

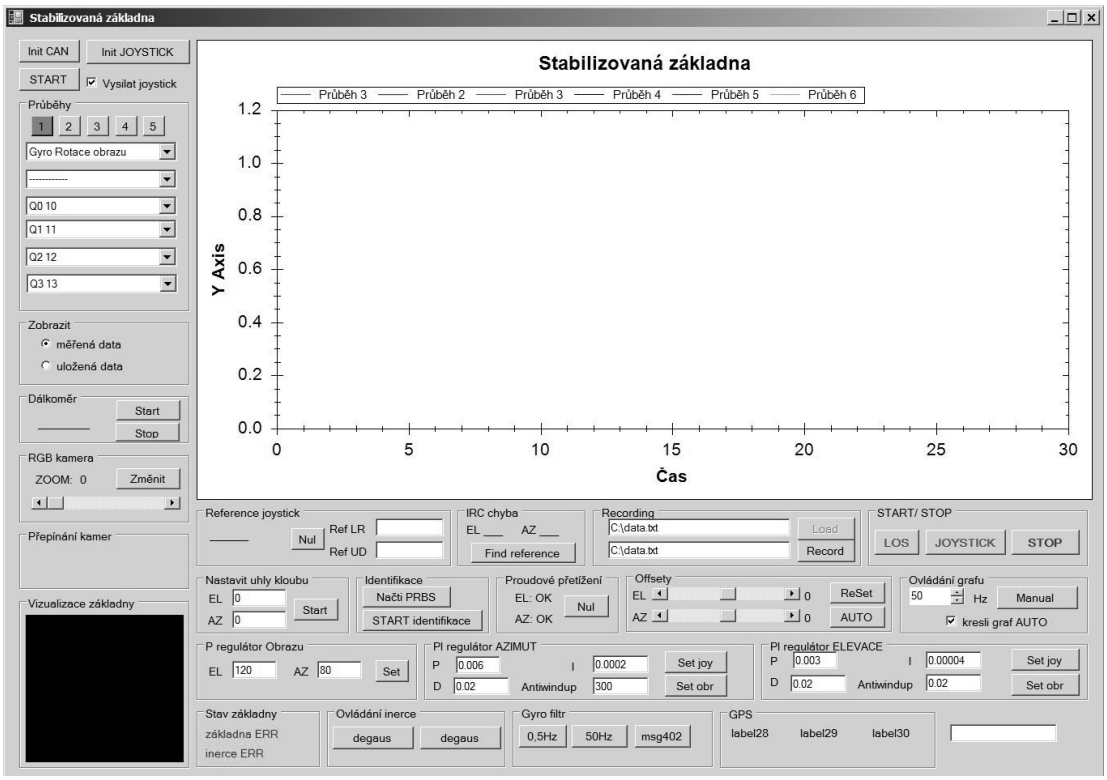
Program, o jehož návrhu a implementaci tato práce pojednává, představuje grafické uživatelské rozhraní (GUI), pomocí kterého lze základnu real-time ovládat a zobrazovat průběhy veličin ze senzorů umístěných na ní. Těchto průběhů může být poměrně velký počet. Program umožňuje přepínání mezi jednotlivými z nich a také zobrazování více průběhů současně. Podobným způsobem může být toto grafické rozhraní použito i pro práci s jiným externím zařízením, nejen se zmíněnou základnou. Starý program toto neumožňoval, protože komunikační protokol byl definován přímo ve zdrojovém kódu. Nový program disponuje konfigurovatelným komunikačním protokolem definovaným v externím souboru. Tato vlastnost činí z programu užitečný nástroj schopný usnadnit práci i na jiných projektech.

1.2 Konkrétní cíle práce

V následujících několika bodech bych rád shrnul hlavní cíle mé práce.

- Prvním úkolem bylo upravit rozvržení grafických komponent do přehlednější podoby. Původní aplikace (obrázek 1.1) měla spoustu tlačítek, textových polí i jiných ovládacích prvků, jejichž počet s vývojem programu rostl a na umístění a rozvržení těchto komponent se nekladl důraz. Bylo tedy třeba navrhnout a realizovat přívětivější uživatelské rozhraní využitím např. oken se záložkami, dialogových oken atd.

- Dalším požadavkem bylo, aby program uměl ukládat a znovu načítat zobrazovaná data. Zde bylo třeba rozmyslet zejména způsob ukládání a přehrávání uložených dat, správně navrhnout funkce a třídy této části programu a zajistit, aby k výstupním částem programu



Obrázek 1.1: Hlavní okno „starého programu“

(např. ke komponentě grafu) vždy přistupoval jen jeden tok dat, tj. buď jenom z přehrávače nebo jen ze sběrnice.

- Program má být schopen současně zobrazovat data z různých senzorů, tento úkol vyžadoval zavedení komunikačního protokolu, ve kterém bude možné definovat různé druhy zpráv. Také bylo třeba umožnit konfigurovatelnost tohoto protokolu, aby byl program případně univerzálně použitelný pro různá zařízení.
- Původní program podporoval komunikaci přes sběrnici CAN, nový program by měl být schopen číst data také ze sběrnice RS232 a do budoucna být rozšiřitelný o podporu dalších typů sběrnic (USB, TCP/IP).
- Jedním ze senzorů stabilizované základny je přijímač GPS. Původní program číselně zobrazoval souřadnice polohy, nový program by měl navíc zobrazovat tuto polohu na mapě načtené z externího souboru.
- Aby bylo možné program v budoucnu vylepšovat a rozšiřovat, měl by být zdrojový kód dobře dokumentován.

Zvolený programovací jazyk je objektově orientovaný C#. V tomto jazyce jsem programoval už v předchozím studiu, dalším důvodem jeho výběru je fakt, že původní program je v něm napsán také a některé jeho třídy (např. pro vizualizaci základny) lze převzít .

KAPITOLA
DRUHÁ

NÁVRH ŘEŠENÍ

Ze všeho nejdříve jsem sestavil objektový návrh programu. Protože kompletní plán se všemi vazbami nebo dokonce funkcemi by byl komplikovaný a bylo by složité se v něm orientovat, zaměřil jsem se hlavně na návrh stěžejních tříd programu. Detaily týkající se jednotlivých částí popíšu následně v podkapitolách.

2.1 Objektový návrh programu

Nejprve tedy návrh programu jako celku. Vše bude lépe vysvětleno pomocí schématu (viz obrázek 2.1).

Ze schématu je patrné, že jádrem programu je třída *Form_Main* volaná třídou *Program*. Této třídě jsou podřízeny všechny ostatní. S výjimkou bloku *Device_Interface*, který je rozhraním, jsou všechny bloky ve schématu třídami. Tento fakt tedy nebudu dále zdůrazňovat.

Vlevo nahoře jsou vidět objekty určené pro definici komunikačního protokolu:

- *Track* je třída, která definuje jeden typ dat[1], se kterými program pracuje . Program do ní tento typ dat ukládá a přes tuto třídu s nimi dále pracuje.
- *Protocol* zahrnuje v sobě všechny definované typy dat - instance třídy *Track*.
- *Config* pokrývá konfigurovatelné parametry programu - jména konfiguračních souborů a parametry sběrnic.

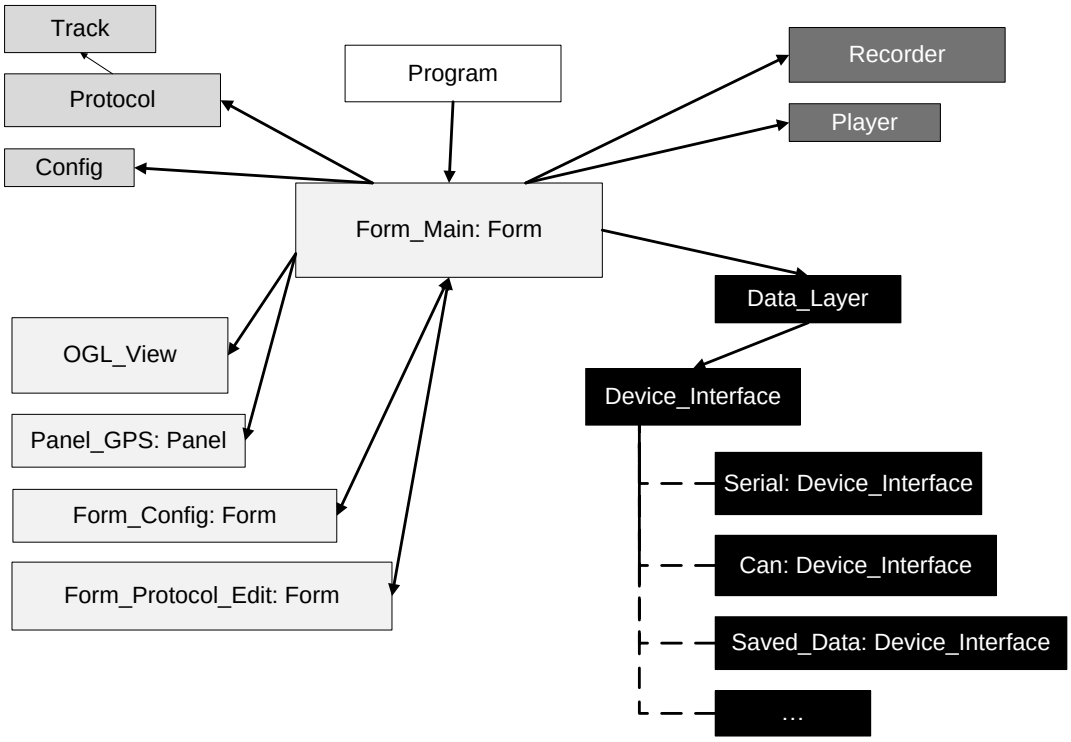
Třídy vpravo nahoře obstarávají ukládání zobrazovaných dat do souboru a jejich zpětné načítání:

- *Recorder* ukládá zobrazená data.
- *Player* načítá a „přehrává“ data z externího souboru.

Vlevo dole jsou na obrázku důležité grafické objekty:

- *OGL-View* je grafický panel, který pomocí knihoven OpenGL zobrazuje virtuální model základny, jenž mění prostorovou orientaci podle svých vstupních dat.

[1] Spojením **typ dat** se budou vždy myslet nějaká specifická data, se kterými program pracuje (např. zeměpisná délka, zrychlení v ose x), nemá tedy nic společného se slovním spojením **datový typ** (např. integer, double, ...).



Obrázek 2.1: Objektový návrh celého programu

- *Panel-GPS* zobrazuje mapu s aktuální zeměpisnou pozicí.
- *Form-Config* je vyskakovací dialogové okno pro nastavení parametrů sběrnic.
- *Form-Protocol-Edit* je vyskakovací dialogové okno pro zobrazení komunikačního protokolu a volbu jeho konfiguračního souboru.

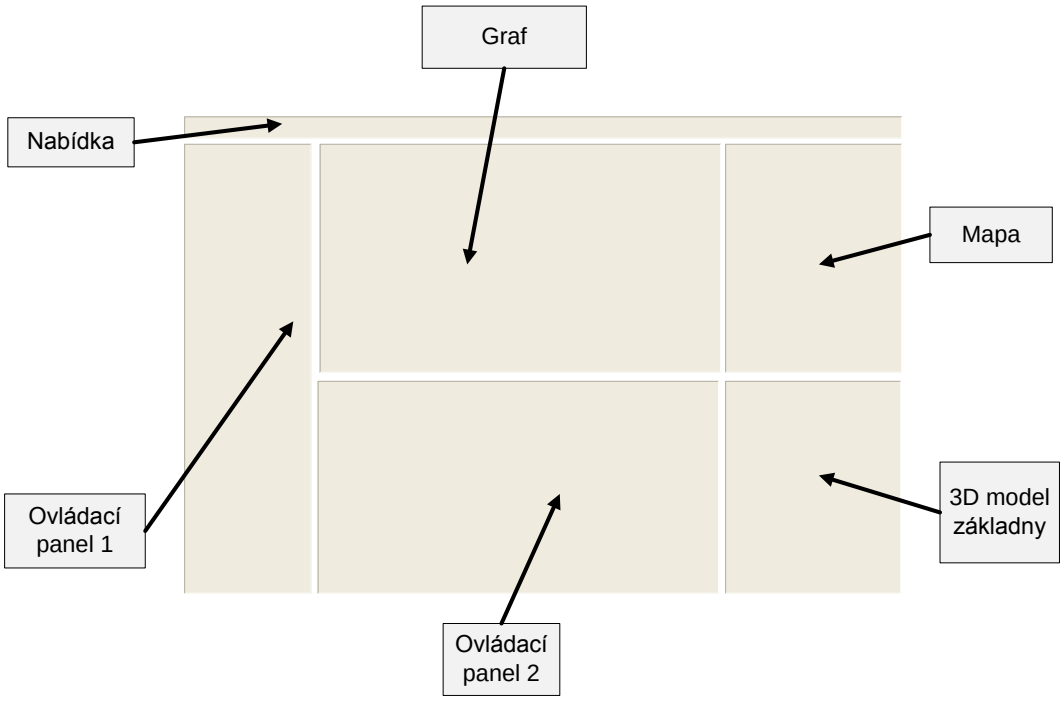
Třídy vpravo dole zajišťují komunikaci s periferií. Provádějí čtení dat ze sběrnice a případný zápis na ni.

- *Device-Interface* je rozhraní definující funkce, které musí mít každá třída pracující v rámci programu s nějakou sběrnicí či zařízením. Obecně funkce pro posílání a příjem dat.
- *Can* reprezentuje sběrnici CAN a implementuje rozhraní *Device_Interface*.
- *Serial* reprezentuje sběrnici RS232 a implementuje rozhraní *Device_Interface*.
- *Data-Layer* obsahuje objekt implementující rozhraní *Device_Interface* a funkce, které s tímto objektem pracují.

To je tedy souhrn hlavních tříd programu a nástin jejich vzájemných vztahů. V následujících podkapitolách se zaměřím podrobněji na jednotlivé části a jejich detailnější návrh.

2.2 Rozložení komponent GUI

Původní program (viz obrázek 1.1) má dvě komponenty, které něco zobrazují. Jsou jimi panel grafu a panel pro vizualizaci základny. V novém programu přibývá ještě třetí taková komponenta, kterou je panel pro zobrazování polohy na mapě. V hlavním okně musíme také počítat s prostorem pro ostatní menší komponenty. Požadavkům vyhovuje rozložení na obrázku 2.2.



Obrázek 2.2: Návrh rozložení komponent v hlavním okně aplikace

Graf je nejpodstatnější částí programu, proto jsem ho umístil na dominantní místo uprostřed horní části okna. Vpravo od grafu se nachází panel pro mapu. Okno pro 3D vizualizaci základny jsem dal do pravého dolního rohu.

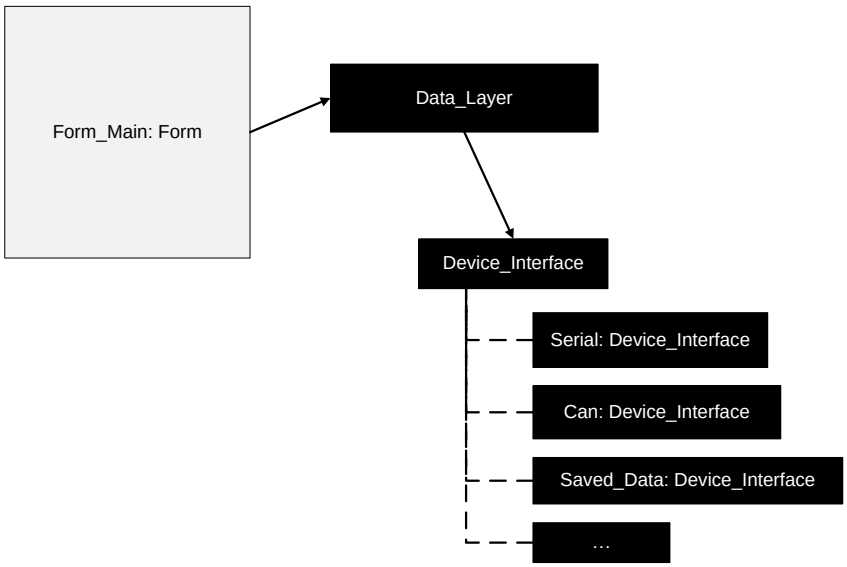
Tímto rozmístěním třech hlavních komponent zbyl prostor ve tvaru písmene L v levé části okna. Toto místo je využito pro ovládací prvky programu. Pro lepší uspořádání těchto malých komponent jsem zmíněný prostor ještě rozdělil na dvě části. Jak je vidět na obrázku 2.2 v levé části okna je Ovládací panel 1 a pod grafem Ovládací panel 2.

2.3 Ukládání a načítání naměřených dat

Než budu psát o návrhu funkcí pro ukládání a načítání, popíšu podrobněji formu, kterou program se základnou komunikuje.

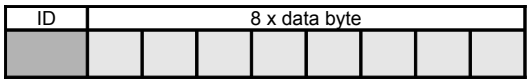
2.3.1 Způsob komunikace programu se základnou

Protože program umožňuje komunikaci s externím zařízením přes různé typy sběrnic, zavedl jsem třídu *Data-Layer*, prostřednictvím které GUI komunikuje s externím zařízením jednotně - nezávisle na typu použité sběrnice. Toho je dosaženo tím, že každý typ sběrnice je reprezentován odpovídající třídou, která implementuje rozhraní *Device-Interface*. Problematika je znázorněna na obrázku 2.3.



Obrázek 2.3: Přístup programu na sběrnici přes *Data-Layer* a *Device-Interface*

Třída *Data-Layer* potom volá jen funkce definované tímto rozhraním. Na této datové vrstvě probíhá komunikace po jednotlivých zprávách (viz obrázek 2.4). Každá zpráva nese své identifikační číslo (ID) a osm datových bytů.



Obrázek 2.4: Formát zprávy posílané přes *Data-Layer*

V praxi věc funguje tak, že základna posílá údaje pomocí těchto zpráv a úkolem našeho programu je zprávy postupně přijímat, rozkódovat, ukládat a zobrazovat. Podobně je možné také zprávy základně posílat. K získání potřebných údajů ze zpráv je nutné znát komunikační protokol. Ten říká, co konkrétní datové byty zprávy s určitým ID znamenají. O tomto protokolu více v podkapitole 3.3.

2.3.2 Ukládání přijatých dat

Nyní popíšu, jak jsem navrhl způsob ukládání přijatých dat do souboru. Vycházel jsem z konceptu, který byl popsán výše (podkapitola 2.3.1), tedy z toho, že je třeba uložit posloupnost zpráv s určitým ID a osmi datovými byty. Funkce z knihoven pro práci se sběrnicí CAN umožňují získat mimo těchto údajů také časový údaj o tom, kdy byla zpráva přijata. To je důležité pro následné vynesení získaných průběhů do grafu, a proto musíme ukládat i tyto časové údaje.

Rozhodl jsem se ukládat tato data do souboru v textové podobě, což umožňuje soubor prohlížet v textovém editoru a nabízí možnost kompatibility s jinými programy. Každá zpráva se zapisuje na jeden řádek. Nejprve ID, poté osm datových bytů oddělených čárkami a nakonec čas přijetí. Tento způsob ukládání je také vhodný pro zpětné načítání, které bude vysvětleno v dalším bodě. O ukládání dat do souboru se stará třída *Recorder*. Příklad části výstupního souboru je na obrázku 2.5.

```
id:334 msg:98,247,196,189,155,91,22,190, t:28
id:335 msg:197,35,213,60,66,145,101,188, t:29
id:336 msg:247,71,212,59,56,202,42,62, t:30
id:337 msg:123,20,182,190,80,235,183,190, t:31
id:1 msg:247,71,212,59,56,202,42,62, t:32
id:338 msg:31,79,72,66,158,169,102,65, t:33
id:339 msg:51,179,69,67,56,202,42,62, t:34
id:330 msg:247,255,183,0,1,1,102,65, t:35
id:331 msg:138,0,45,0,64,254,102,65, t:36
id:332 msg:75,0,32,0,63,255,102,65, t:37
id:333 msg:107,201,123,191,149,218,100,189, t:38
id:334 msg:15,95,203,189,92,150,15,190, t:39
id:335 msg:254,17,213,60,131,236,101,188, t:40
id:336 msg:12,253,212,59,16,160,143,61, t:41
id:337 msg:39,232,128,190,125,53,130,190, t:42
```

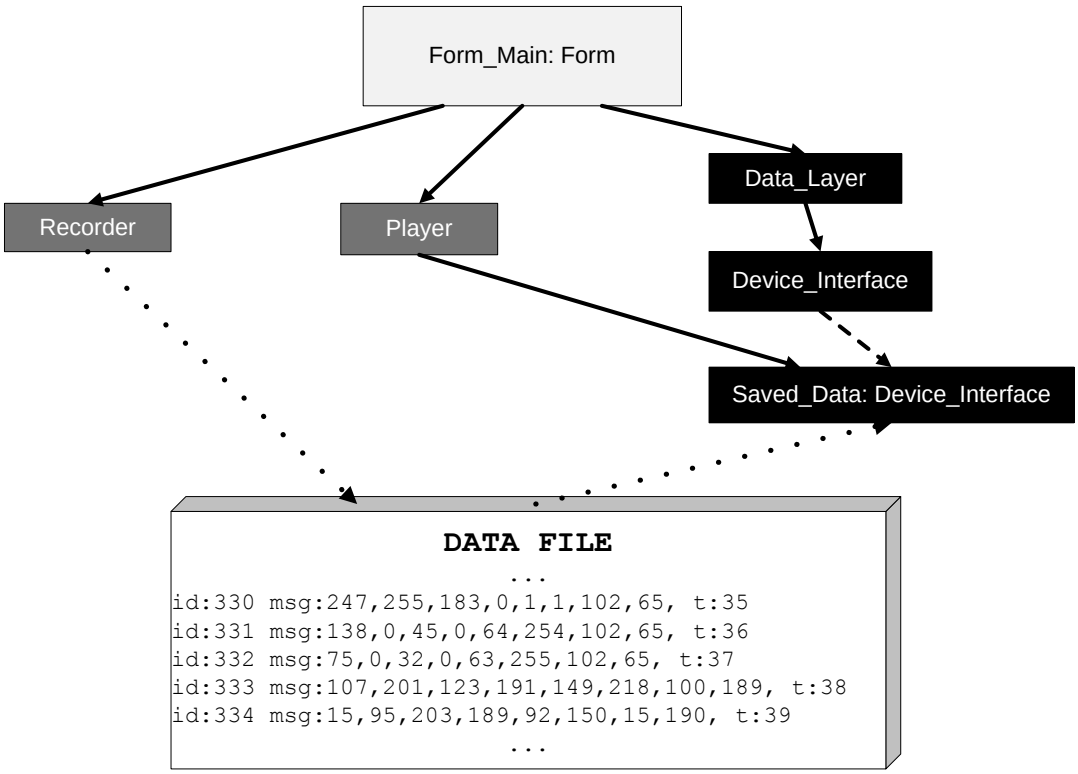
Obrázek 2.5: Příklad části výstupního souboru s uloženými daty

2.3.3 Načítání dat ze souboru

Navržený způsob ukládání umožňuje zpětné načítání souboru podobným způsobem, jakým se čtou data ze sběrnice. Stačí mít funkci, která ze vstupního souboru přečte vždy jeden řádek, jenž odpovídá jedné zprávě, a vrátí ID, pole datových bytů a čas zprávy. Toho jsem docílil tak, že se data načtená ze souboru zapouzdří do třídy *Saved_Data*, která implementuje rozhraní *Device_Interface* (viz podkapitola 2.5). S takto zpracovanými daty ze souboru už program dále pracuje stejně jako s třídou *Can*.

Pro rozšíření možností práce s daty načtenými ze souboru jsem navrhl třídu *Player*, která umožňuje naměřené průběhy přehrávat a krokovat v nich. Třída tedy bude mít přístup k celému objemu načtených dat, ale zobrazovat bude jen jejich určitou podmnožinu, tj. vybrané průběhy od počáteční časové hodnoty do časové hodnoty nastavené v této třídě. *Player* také poskytuje potřebná data pro zobrazení polohy na mapě a vykreslení 3D Open GL modelu v aktuální ori-

entaci. Pro snadnější pochopení uvádím schéma návrhu tříd pro ukládání a načítání dat (je na obrázku 2.6).



Obrázek 2.6: Návrh ukládání a načítání naměřených dat

2.4 Konfigurovatelnost komunikačního protokolu

Dostávám se k návrhu nejdůležitější části programu. Tím je realizace komunikačního protokolu a zajištění jeho konfigurovatelnosti. V podkapitole 2.1 byla už o této problematice zmínka, v souvislosti s třídami *Track* a *Protocol*. Zde tyto třídy a jejich vztah popíšu podrobněji.

Každý definovaný průběh[2] reprezentuje v programu jedna instance třídy *Track*. Třída obsahuje specifické proměnné, které konkrétní průběh určují. Jednou z těchto proměnných je ID zprávy, z jejíž dat získáváme každou další hodnotu průběhu. Jak víme, každá zpráva nese až osm datových bytů. Tyto byty mohou nést hodnoty více průběhů v závislosti na zvoleném datovém typu. Například datový typ int16 má 16 bitů, tedy 2 byty. Jedna zpráva může tedy v tomto případě nést hodnoty čtyř veličin, jestliže zvolený datový typ každé z nich je int16. Proto je další proměnnou třídy *Track* index prvního datového bytu, který náleží ve zprávě danému průběhu. Dále jsou proměnnými třídy *Track* také název instance a již zmíněný datový typ.

Třída *Protocol* zahrnuje všechny instance třídy *Track*, tedy všechny definované průběhy. Abych zajistil konfigurovatelnost komunikačního protokolu, jednotlivé instance třídy *Track* se vytvářejí

[2] *Průběh* je vhodné slovo v případě, že se jedná například o data ze senzoru, někdy je výstižnějším výrazem *veličina* či *příkaz*. Aby nedocházelo ke zmatkům přidržím se jednotně označení *průběh*.

podle externího textového konfiguračního souboru. V tomto souboru každý řádek definuje jednu instanci třídy *Track*. Řádek definující instanci *Track* musí respektovat konvenci zápisu, jinak je neplatný. Obsahuje následující parametry:

1. ID - identifikační číslo zprávy, která nese data dané instance třídy *Track*. Jedná se vždy o celé číslo a může být zapsáno i hexadecimálně (např. 0xA1).
2. NAME - jméno instance. Může obsahovat libovolné znaky krom dvojtečky.
3. TYPE - datový typ, v němž jsou data přenášena (int16, int32, uint16, uint32, float, double). Udává také počet datových bytů.
4. POSITION - index prvního datového bytu, který ve zprávě náleží dané instanci.

Jednotlivé parametry se oddělují dvojtečkou. Příklad konfiguračního souboru definujícího komunikační protokol je na obrázku 2.7.

```
ID:NAME:TYPE:POSITION

332:gyroskop x:int16:0
332:gyroskop y:int16:2
332:gyroskop z:int16:4

338:GPS-X:float:4
338:GPS-Y:float:0

331:akcelerometr x:int16:0
331:akcelerometr y:int16:2
331:akcelerometr z:int16:4
```

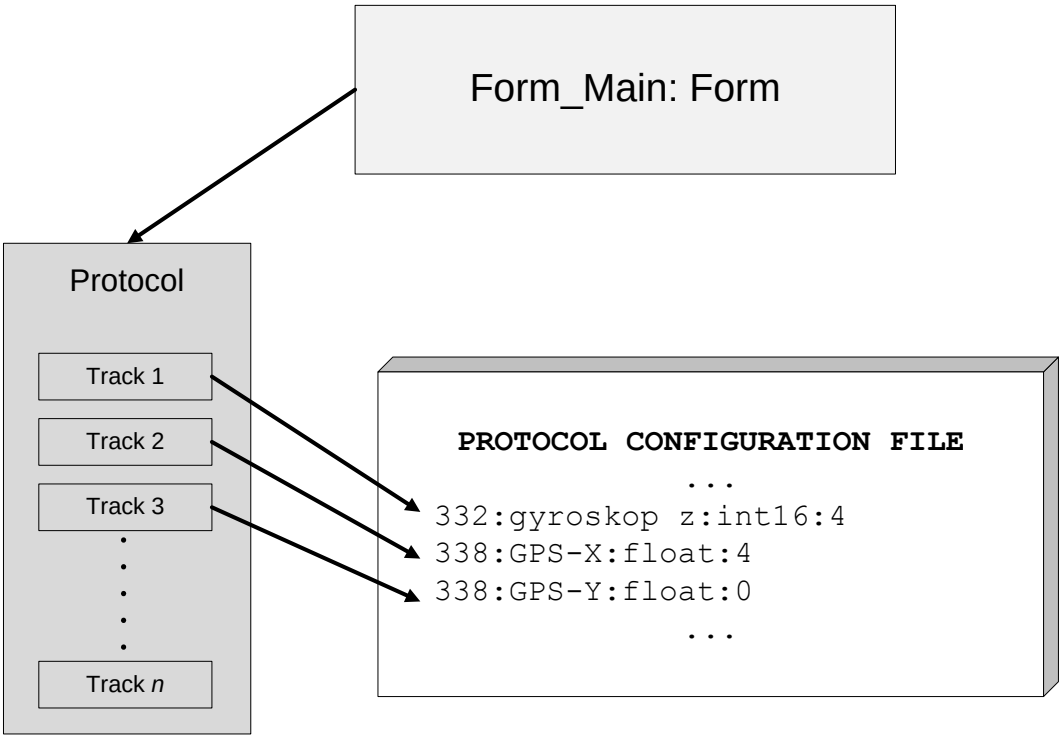
Obrázek 2.7: Konfigurační soubor komunikačního protokolu

Jak už bylo zmíněno, každá instance třídy *Track* schraňuje svoje data - svůj průběh, takže je následně možné tento průběh např. vykreslit do grafu. Pro názornost a snadnější pochopení návrhu uvádím diagram na obrázku 2.8.

2.5 Kompatibilita se sběrnicí RS232

V této podkapitole se zaměřím na návrh části programu zajišťující podporu sběrnice RS232.

Původní program je kompatibilní se sběrnicí CAN, jež je zde reprezentována třídou *Can*. Při návrhu jsem využil objektově orientovaného charakteru jazyka C# a zavedl jsem rozhraní *Device_Interface* definující funkce, které musí splňovat každá třída reprezentující nějakou sběrnici či externí zařízení, s nímž má program komunikovat. Obecně jsou to funkce obousměrné komunikace mezi zařízením a aplikací prostřednictvím zpráv, jejichž formu jsem popsal výše



Obrázek 2.8: Návrh tříd pro definici komunikačního protokolu

v podkapitole 2.3.1. Za vzor jsem tedy vzal způsob, jakým program komunikuje se sběrnicí CAN.

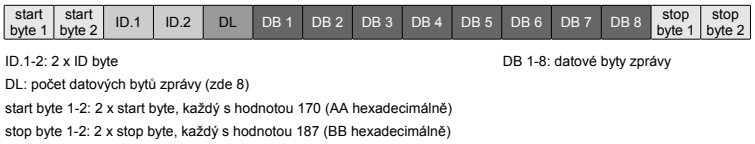
Při přenosu dat po sériové lince je nutné definovat, jak budou od sebe jednotlivé zprávy odděleny. Zvolil jsem oddělení zpráv pomocí dvojic start a stop bytů jak je ukázáno na obrázku 2.9.

Zpráva začíná dvěma start byty, z nichž každý má hodnotu 170 a končí dvěma stop byty s hodnotou 187. Relativně malá pravděpodobnost výskytu takových dvojic bytů jinde ve zprávě zajišťuje malou ztrátovost dat. Třetí a čtvrtý byte zprávy nesou její ID, byte následující udává počet bytů datových.

Dále jsem navrhl třídu *Serial*, která implementuje rozhraní *Device_Interface*. Třída pracuje s výše popsanými zprávami a poskytuje kompatibilní data vyšší vrstvě (třídě *Data_Layer*).

2.6 Zobrazování polohy z GPS senzoru

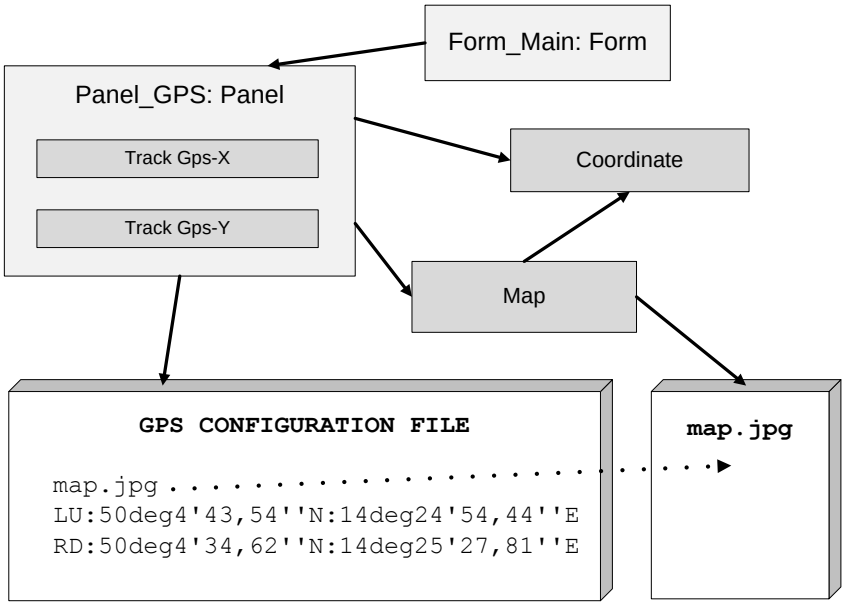
V této podkapitole se zaměřím na návrh části programu, která obstarává zobrazování zeměpisné polohy základny podle dat z GPS přijímače. Pro tento účel jsem navrhl třídy *Panel_GPS*, *Map*, strukturu *Coordinate*.



Obrázek 2.9: Schéma zprávy posílané po sériové lince

2.6.1 Objektový návrh

Třídu *Panel_GPS* jsem už letmo představil v podkapitole 2.1, o třídě *Map* a struktuře *Coordinate* mluvím až nyní, protože s jinými třídami programu než je *Panel_GPS* nesouvisejí. Vztahy mezi zmíněnými objekty vysvětlím následně pomocí obrázku 2.10.



Obrázek 2.10: Objektový návrh třídy *Panel_GPS*

Třída *Panel_GPS* používá k vykreslování údaje dvou průběhů. Těmi jsou průběh s údaji zeměpisné šířky (*Track Gps-Y*) a průběh s údaji zeměpisné délky (*Track Gps-X*). Program na zavolání vždy vykreslí polohu podle posledních hodnot těchto průběhů.

Aby bylo možné zobrazovat zeměpisnou polohu na mapě, je nutné mít vhodnou mapu a znát např. zeměpisné souřadnice pravého dolního a levého horního rohu mapy. U map malého měřítka lze zanedbat zakřivení země, čehož jsem využil při návrhu. Dále jsem při návrhu vycházel z toho, že poledníky budou na mapě rovnoběžné s jejími svislými okraji a podobně budou rovnoběžky rovnoběžné s vodorovnými okraji mapy. Ze známých souřadnic zmíněných rohů pak

lze vypočítat, kolik pixelů na obrázku mapy odpovídá zeměpisně jedné sekundě. Tento poměr se v programu počítá zvlášť pro zeměpisnou šířku a zvlášť pro délku, jejichž hodnoty se mohou lišit.

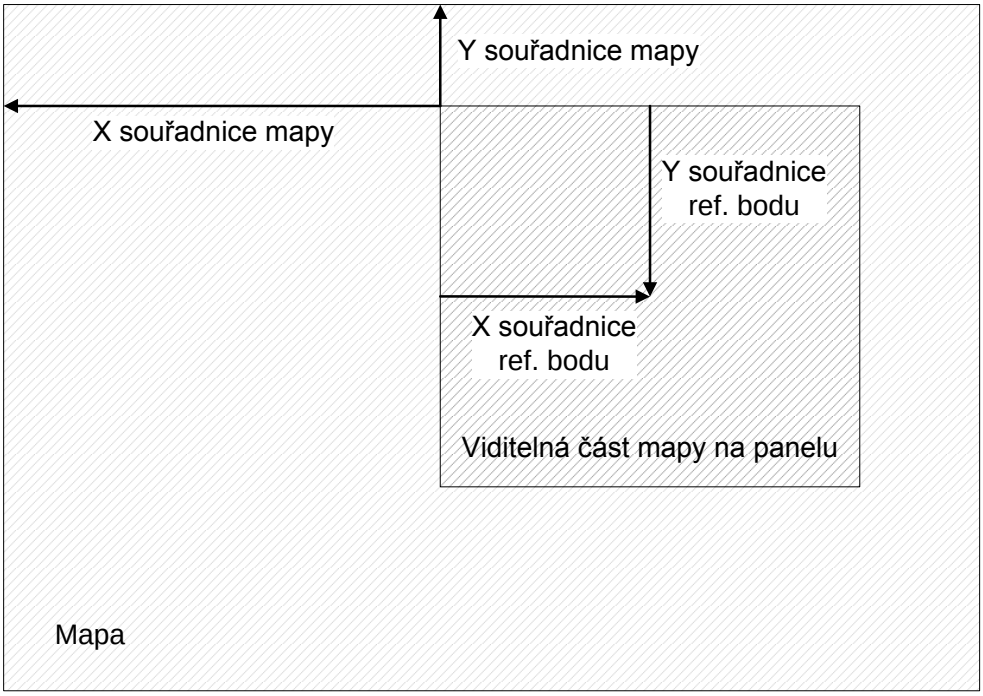
Třída *Map* tedy sdružuje proměnné popsané v předchozím odstavci a obsahuje také obrázek mapy.

Struktura *Coordinate* umožňuje zadávat zeměpisné souřadnice ve tvaru „stupně, minuty, sekundy“ a převádět tento tvar na stupně a naopak. Zeměpisná souřadnice má charakter datového typu, proto jsem zvolil použití struktury a ne třídy.

Na obrázku 2.10 je také znázorněno, že jméno souboru s mapou a zem. souřadnice zmíněných dvou rohů této mapy jsou třídou *Panel_GPS* načítány z externího textového souboru. Forma zápisu údajů do tohoto konfiguračního souboru je z obrázku také zřejmá.

2.6.2 Zobrazování mapy

Zeměpisnou polohu ukazuje program stejným způsobem jako jiné aplikace podobného typu, tedy referenční značka je vždy uprostřed panelu a v pozadí se pohybuje mapa. Program počítá souřadnice (v pixelech) levého horního rohu mapy a podle toho obrázek mapy vykresluje na komponentu *Panel-GPS*. Tento princip zobrazování je znázorněn na obrázku 2.11.



Obrázek 2.11: Zobrazování polohy na mapě

2.7 Dokumentace zdrojového kódu

Vytvořit solidní dokumentaci zdrojového kódu ručně by bylo časově náročné a nepraktické. Rozhodl jsem se pro tento účel využít volně dostupný nástroj Doxygen [1], který je schopný do-

kumentaci generovat z XML komentářů zdrojového kódu. Také jsem využil plugin Graphviz [2], pomocí něhož Doxygen přidá do dokumentace diagramy tříd. Ukázka výsledné dokumentace je na obrázku.

Main Page | Classes

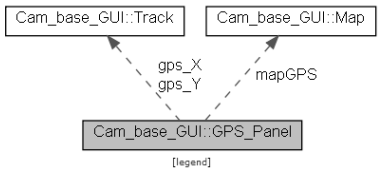
Class List | Class Hierarchy | Class Members

Cam_base_GUI::GPS_Panel

Cam_base_GUI::GPS_Panel Class Reference

Panel for GPS map with current position mark. More...

Collaboration diagram for Cam_base_GUI::GPS_Panel:



List of all members.

Public Member Functions

void	**Initialize** (string config_file, BufferedGraphics bg, **Track** gps_X, **Track** gps_Y) Load gps map and its settings.
void	**drawGPSMap** (BufferedGraphics bg, Label lab) Draws gps map with position mark in current position and current coordinates on label.

Properties

double	**LongitudePosition** `[get, set]` Get or set current gps position - longitude.
double	**LatitudePosition** `[get, set]` Get or set current gps position - latitude.
Track	**Gps_X** `[get, set]`
Track	**Gps_Y** `[get, set]`
string	**Description** `[get]` Returns GPS settings description.

Obrázek 2.12: Ukázka dokumentace generované nástrojem Doxygen

KAPITOLA
TŘETÍ

IMPLEMENTACE A TESTOVÁNÍ

V této kapitole se zaměřím na popis implementační a testovací části mé práce. Rozhodl jsem se tyto dvě věci popisovat najednou, protože jsem během programování aplikaci zároveň i spouštěl a testoval nově naprogramované funkce. Nebudu popisovat jednotlivé funkce, ale spíš způsob implementace, aby bylo zřejmé jak program funguje. Popíši vnitřní strukturu tříd, jejichž návrh jsem rozebíral v předchozí kapitole, jejich hlavní metody, a představím podrobněji některé důležité použité komponenty.

3.1 Rozložení komponent GUI

V této podkapitole se zaměřím na implementaci komponent grafického rozhraní. Tento problém se týkal především hlavního okna *Form-Main* (obrázek 3.1), okna komunikačního protokolu (obrázek 3.3) a okna pro nastavování parametrů sběrnic (obrázek 3.3).

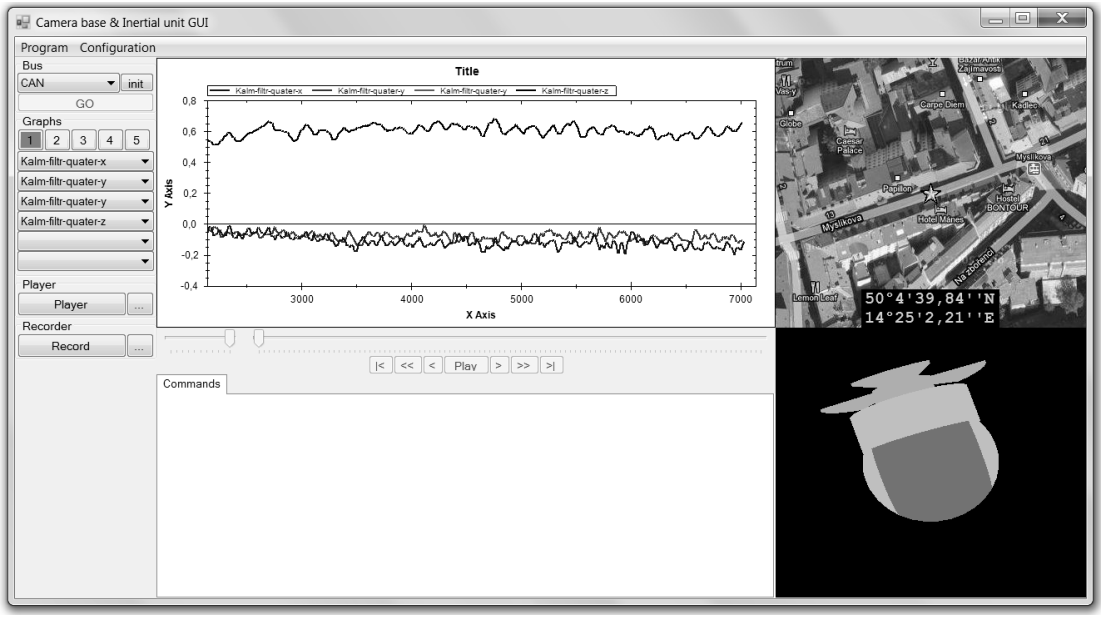
3.1.1 Hlavní okno *Form-Main*

Toto okno je nejdůležitější částí celého programu. Návrh jeho rozdělení do dílčích sekcí jsem popsal v podkapitole 2.2 a názorně je vidět na obrázku 2.2.

O umístění a velikost jednotlivých komponent v okně se stará metoda *Component-Size-Settings*, která je volána při startu programu a při překreslování okna, např. při změně jeho velikosti. Metoda mimo jiné zajišťuje proporcionální změnu velikosti jednotlivých komponent okna, takže aplikace vypadá pěkně i když není ve full-screen módu.

O vykreslování grafů se stará komponenta *ZedGraphControl*, která je součástí knihovny *ZedGraph* stažitelné z webu [3]. Třídu okna zobrazujícího virtuální model pomocí grafických knihoven Open GL jsem převzal beze změn z původního programu.

Zaměřím se nyní blíže na komponenty v levém ovládacím panelu. Komponenty jsem podle účelu rozčlenil do odpovídajících regionů použitím komponent typu *GroupBox*. V horní části je skupina komponent označených jako *Bus*, pomocí nichž je možné přepínat a inicializovat sběrnice. Přepínání je realizováno pomocí komponent typu *ComboBox*, vedle je tlačítko pro inicializaci sběrnice a pod těmito prvky je tlačítko, kterým je možné pozastavit čtení dat, čímž se zastaví vykreslování grafů, polohy na mapě i virtuálního modelu.



Obrázek 3.1: Hlavní okno programu

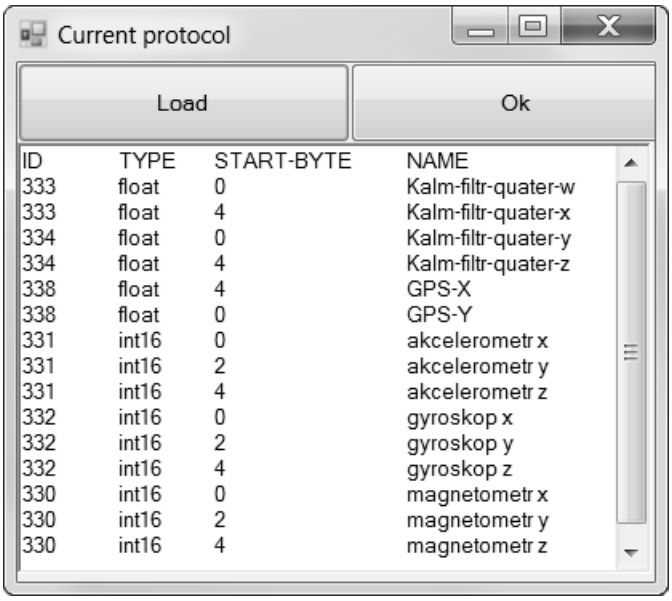
Další skupina ovládacích prvků je označena jako *Graphs*, protože se pomocí nich vybírá, které průběhy má program na komponentě grafu zobrazovat. Šest komponent typu *ComboBox* umožňuje zobrazení až šesti zvolených průběhů najednou a horní tlačítka s číslicemi umožňují přepínat mezi pěti předvolenými sadami grafů. Nastavení je po skončení programu uloženo a při dalším spuštění programu jsou předvolené sady opět k dispozici.

Skupina komponent označená jako *Player* se skládá pouze ze dvou tlačítek. Velké slouží jako spínač a vypínač přehrávače záznamů dat. Přepíná mezi módem přehrávání uložených dat ze souboru a módem, kdy program komunikuje se zařízením prostřednictvím rozhraní *Device-Interface*. Menší tlačítko vedle umožňuje vybrat vstupní soubor. K přehrávači ale patří také podélný panel pod komponentou grafu. Zde jsou dva posuvníky (komponenta *TrackBar*). První - menší umožňuje zvolit rychlost přehrávání. Druhý - dlouhý indikuje aktuální časovou pozici v celém přehrávaném záznamu a je možné jeho posouváním rolovat v časové ose. Tlačítka pod posuvníky mají standardní funkce běžné u jiných přehrávačů (pozastavení, přeskakování, zpětné přehrávání, posun na konec/začátek).

3.1.2 Okno komunikačního protokolu

Komunikační protokol (viz podkapitola 2.4 a 3.3) lze prohlížet a načíst ze souboru pomocí dialogového okna *Form-Protocol-Edit*, jež je vidět na obrázku 3.2.

Okno se z programu zavolá z horní lišty: Configuration→Protocol. V textovém poli se zobrazuje aktuálně používaný protokol. Lze zde vyčíst parametry jednotlivých průběhů (viz podkapitola 3.3). V horní části okna jsou dvě tlačítka. Tlačítkem označeným jako *Load* lze zadat cestu k novému konfiguračnímu souboru pro nastavení komunikačního protokolu. Druhé tlačítko s nápisem *Ok* okno zavírá.



Obrázek 3.2: Dialogové okno *Form_Protocol_Edit*

3.1.3 Okno konfigurace sběrnic

Při inicializaci sběrnice CAN či RS232 je nutné nastavit některé parametry (číslo kanálu, baudovou rychlost atd.), pro tento účel je v aplikaci okno *Form_Config*. Jeho náhled vidíme na obrázku 3.3.

Do okna je vloženo další okno se záložkami, to umožňuje přidat další listy určené pro nastavování jiných věcí v programu. Pro příklad je zde nastavení parametrů při komunikaci se sběrnicí CAN. Nastavování je realizováno zaškrtáváním požadovaných položek ze seznamu možných.

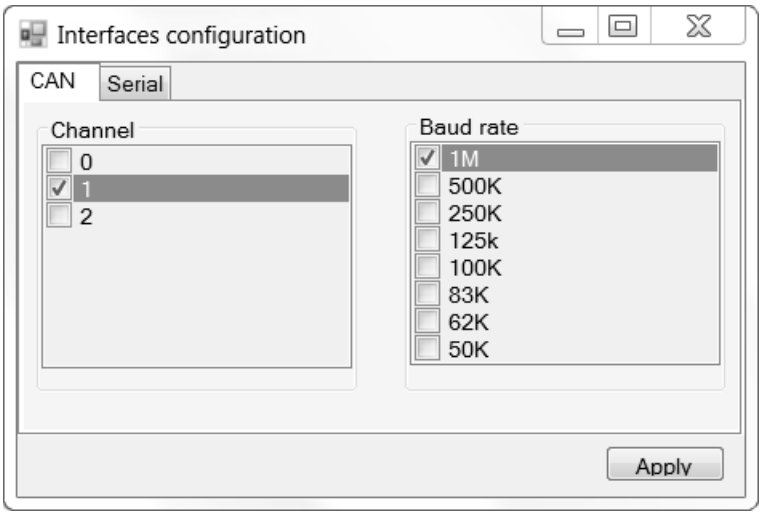
V pravém dolním okně je tlačítko *Apply*, kterým se potvrzují provedené změny. Okno se poté zavře. Toto tlačítko je společné pro všechny listy záložkového okna, všechny změny se tedy potvrzují najednou.

3.2 Ukládání a načítání naměřených dat

Návrh řešení problému ukládání a zpětné načítání měřených průběhů jsem popisoval v podkapitole 2.3, v následujících řádcích rozeberu, jak jsem návrh realizoval.

3.2.1 Implementace třídy Recorder

O zápis dat do výstupního souboru se podle návrhu stará třída *Recorder*. Na obrázku 2.5 je vidět, že data se ukládají po řádcích v textové podobě. Hlavní metodou této třídy je metoda *Record-Message*, která pomocí třídy *StreamWriter* zapíše do výstupního souboru jeden řádek odpovídající jedné zprávě. Vstupními parametry metody jsou ID zprávy, osm datových bytů a čas. Všechny hodnoty jsou převedeny na řetězec znaků (*String*) a v předepsaném tvaru zapsány do souboru.



Obrázek 3.3: Okno pro konfiguraci sběrnic

Protože metoda *Record-Message* může být případě sepnutého rekorderu volána velice často, zavedl jsem ve třídě *Recorder* buffer, realizovaný proměnnou typu *String*, do kterého jsou data zapisována primárně. Jde vlastně o sčítání řetězců. Teprve, když je buffer plný, přepíše se jeho obsah do souboru, což je náročnější operace a je lépe ji provádět méně často. Zapínání a vypínání rekorderu je realizováno jednoduše nastavováním jedné jeho proměnné typu *bool*.

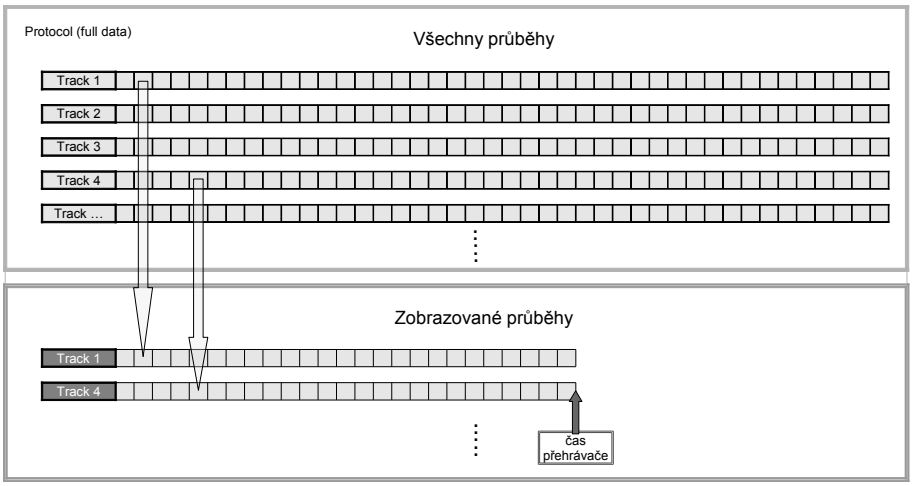
3.2.2 Implementace třídy Player

Třída *Player* zahrnuje všechny funkce, které umožňují zaznamenané průběhy přehrávat a pracovat s nimi. Hlavním problémem implementace této třídy bylo docílit toho, aby bylo možné přistupovat k celému objemu dat, tedy ke všem zaznamenaným průběhům v celém jejich rozsahu a zároveň zobrazovat jen jejich určitou část charakterizovanou časovým ukazatelem přehrávače.

Tento problém jsem vyřešil tak, že při inicializaci playeru se do paměti načte celý objem zaznamenaných dat. Tato data jsou uložena prostřednictvím instance třídy *Protocol*. Zobrazovaná data jsou reprezentována ve třídě *Player* zvlášť pomocí instancí třídy *Track*, ve kterých se udržují jen vykreslované části průběhů (viz schéma na obrázku 3.4). Podobně jsou zde reprezentovány také GPS průběhy a průběhy určující prostorovou orientaci základny, aby bylo možné vždy vykreslit polohu na mapě a virtuální Open GL model. K takovému vykreslování ale dojde pouze tehdy, když jsou dané průběhy ve třídě *Protocol* nadefinovány (viz podkapitola 3.3).

Za nejdůležitější část třídy *Player* bych označil metodu *Play*, která, pomocí dvou dalších metod (*Next-Point*, nebo *Previous-Point*), po svém zavolání aktualizuje data v zobrazovaných průbězích. Přidává do nich nebo z nich ubírá hodnoty podle toho, jestli je hodnota požadovaného referenčního času přehrávače menší než hodnota stávající, či naopak.

Kvůli funkci přehrávání je v programu zaveden zvlášť časovač (*Timer*), který v pravidelném intervalu volá zmíněnou metodu *Play*. Velikost časového intervalu tohoto časovače lze měnit v ovládání přehrávače (menším ze dvou posuvníků) a tím nastavovat tempo přehrávání.



Obrázek 3.4: Schéma uložení průběhu ve třídě *Player*

3.3 Konfigurovatelnost komunikačního protokolu

V této podkapitole vysvětlím, jak jsem naimplementoval část programu, která se stará o správu a konfiguraci komunikačního protokolu a také schraňuje všechna data naměřených průběhů. Zaměřím se hlavně na praktické provedení tříd *Track* a *Protocol*.

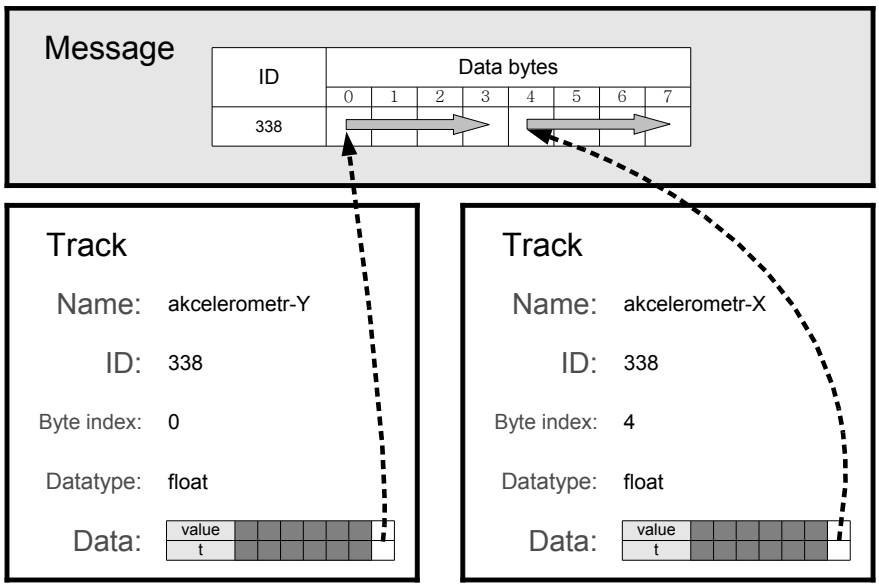
3.3.1 Realizace třídy *Track*

Jak už bylo zmíněno v podkapitole 2.3.1, hlavním úkolem každé instance třídy *Track* je reprezentovat jeden konkrétní průběh. Každý takový průběh charakterizují následující parametry:

- Název průběhu (např. akcelerometr-X)
- ID zprávy (např. 338)
- Index počátečního datového bytu ve zprávě (0 - 7), který náleží danému průběhu (např. 4)
- Použitý datový typ při přenosu hodnot průběhu zprávou (např. float)

Vnitřní struktura třídy *Track* je znázorněna na obrázku 3.5. Na obrázku je také vidět, jak jsou data z příchozích zpráv tříděna do příslušných průběhů. Ku příkladu zde nese zpráva s ID 338 data jak pro průběh *akcelerometr-X* tak pro *akcelerometr-Y*.

Parametrem konstruktoru třídy je řetězec znaků odpovídající jednomu řádku z konfiguračního souboru pro komunikační protokol. Z tohoto řetězce jsou za použití regulárního výrazu rozpoznány hodnoty příslušných parametrů instance. Jednotlivé hodnoty průběhu jsou postupně ukládány jako body do speciálního pole, které lze snadno zobrazovat komponentou grafu. U



Obrázek 3.5: Vnitřní struktura třídy *Track*

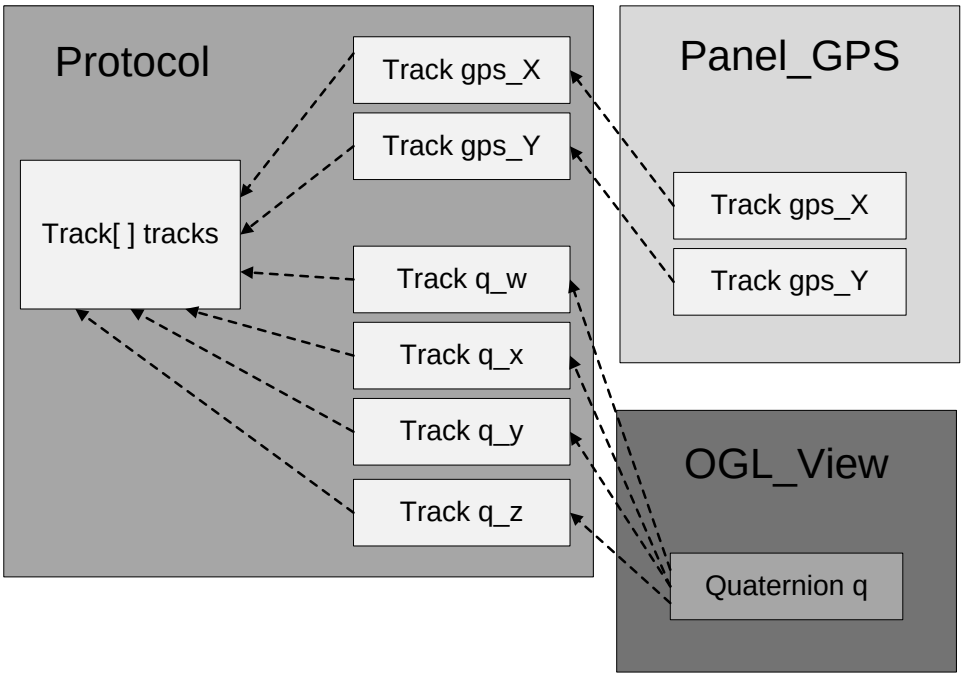
těchto bodů tvoří souřadnici x čas a souřadnici y samotná hodnota průběhu v daném časovém okamžiku.

Instanci třídy *Track* lze vytvořit ve dvou verzích. V první dochází k ukládání hodnot průběhu do pole konečné délky. Když je pole zcela naplněno, nové hodnoty jsou nadále přidávány, ale dochází k mazání hodnot starých. V poli se tak udržuje konstantní množství aktuálních dat, jejichž průběh může být vykreslována a zároveň se tak zamezuje přetečení kapacity paměti. U druhé verze tomu tak není. Tam má pole pro ukládání dat nekonečnou kapacitu. První verze se v programu používá pro reprezentaci dat ze sběrnice, druhá k načtení dat ze souboru a přehrávání pomocí třídy *Player*. Zda má dojít k vytvoření instance třídy *Track* s polem konečné, či nekonečné kapacity, rozhoduje hodnota dalšího parametru konstruktoru této třídy.

3.3.2 Realizace třídy *Protocol*

Hlavním účelem třídy *Protocol* je schraňovat všechny definované průběhy a zpřístupňovat tyto průběhy ostatním částem programu.

Hlavní částí této třídy je metoda *Load*, na jejíž zavolání se načte konfigurační soubor komunikačního protokolu a vytvoří se podle něho všechny instance třídy *Track*. Třída také při této inicializaci zaznamenává základní informace o těchto zmíněných instancích a ukládá je do své proměnné *description*. Vypsáním této proměnné lze zjistit jaké průběhy jsou v programu na-



Obrázek 3.6: Schéma vnitřní struktury třídy *Protocol*

definovány a jejich parametry. Této vlastnosti je využíváno například při práci s oknem *Form-Protocol-Edit*, kde se výpis provádí do textového okna (viz obrázek 3.2). Třída *Protocol* také poskytuje průběhy potřebné pro vykreslování polohy na mapě a virtuálního modelu základny. K tomuto účelu jsou v této řídě definovány referenční proměnné, které ukazují na příslušné instance třídy *Track* v seznamu (viz schéma na obrázku 3.6).

Protože je prostřednictvím třídy *Protocol* přistupováno ke všem definovaným průběhům, obsahují referenční proměnnou na její instanci třídy *Player*, *Data-Layer*, samozřejmě *Main-Form* a také *Form-Protocol-Edit*. Pomocí poslední je možné komunikační protokol načítat a prohlížet.

3.4 Kompatibilita se sběrnicí RS232

Jak už jsem popsal v návrhu (viz podkapitola 2.5) kompatibilitu programu se sběrnicí RS232 jsem realizoval použitím rozhraní. Pojďme se na věc podívat po implementační stránce.

Rozhraní (interface) umožňuje v programu definovat určitou charakteristickou množinu metod, kterou musí obsahovat každá třída, jež toto rozhraní implementuje. Pomocí rozhraní lze v objektově orientovaném programování pracovat jednotným způsobem s různými třídami, jestliže dané rozhraní implementují. Je to jeden z případů použití polymorfismu. Právě této vlastnosti jsem využil při řešení problému s kompatibilitou programu s jinými sběrnicemi.

Zavedl jsem tedy rozhraní *Device-Interface*, jež zahrnuje funkce společné pro všechny typy sběrnic, které by mohly s programem spolupracovat. Jsou to funkce pro odesílání zpráv, příjímání zpráv a pro inicializaci sběrnice.

Nyní konkrétně k realizaci třídy *Serial*, která reprezentuje sériovou linku (RS232). Třída tedy

implementuje rozhraní *Device-Interface*. Využil jsem již existující standardní třídy *SerialPort* a zahrnul ji jako proměnnou do třídy *Serial*. Prostřednictvím funkcí třídy *SerialPort* probíhá komunikace po sériové lince. Při inicializaci je nutné nastavit příslušné parametry pro přenos po sériové lince (např. baudovou rychlost). Tyto parametry jsou, podobně jako tomu je v případě sběrnice CAN, reprezentovány ve třídě *Config* a je možné je měnit prostřednictvím dialogového okna *Form-Config* (viz obrázek 3.3).

Důležitým prvkem třídy je metoda pro čtení dat ze sběrnice. Příchozí data jsou ukládána do bufferu a poté v pravidelném časovém intervalu čtena. Tento princip je shodný se čtením dat ze sběrnice CAN. Data lze třídit do odpovídajících průběhů jen pokud jsou přijímána v předepsané formě (viz obrázek 2.9). Ve zmíněné metodě se v cyklu provádí čtení bytů a formují se do požadovaných výstupních parametrů. Metoda potom vrací parametry zprávy (ID, datové byty, čas), které umí program dále zpracovávat jednotně a nezávisle na použitém typu sběrnice. Podobně probíhá inverzní operace, tedy zápis dat na sběrnici.

3.5 Zobrazování polohy z GPS senzoru

V této podkapitole se zaměřím hlavně na popis provedení třídy *Panel-GPS*. Implementace odpovídá návrhu na obrázku 2.10.

Třída *Panel-GPS* je potomek standardní třídy *Panel*. Z tohoto faktu vyplývá, že se jedná o grafickou komponentu, na niž lze vykreslovat. V programu je tato komponenta určena k vykreslování obrázku mapy a referenční značky ukazující polohu na této mapě. Aby se zamezilo „blikavému“ efektu při častém překreslování použil jsem jako vykreslovací prostředek třídu *BufferedGraphics*. Obrázek mapy a příslušné parametry (viz obrázek 2.10) jsou reprezentovány třídou *Map* a načítají se ze zvláštního konfiguračního souboru. Při inicializaci panelu dochází k načtení obrázku a do třídy *Image*. V této formě už lze obrázek přímo vykreslit. Zbývá tedy vypočítat pozici obrázku na grafickém panelu. Tuto pozici lze snadno vypočítat ze známých hodnot příslušných parametrů:

$$X = (H - H_{LU}) \cdot \left(\frac{p}{s}\right)_H - \frac{W_w}{2} \tag{3.1}$$

$$Y = (V_{LU} - V) \cdot \left(\frac{p}{s}\right)_V - \frac{W_h}{2} \tag{3.2}$$

- X, Y - souřadnice pro vykreslení obrázku mapy na komponentu *Panel-GPS* v pixelech (obrázek 2.11)
- H - referenční zeměpisná délka v sekundách
- H_{LU} - zeměpisná délka levého horního rohu mapy v sekundách
- $\left(\frac{p}{s}\right)_H$ - koeficient určující kolik pixelů připadá na jednu sekundu zeměpisné délky
- W_w- šířka komponenty *Panel-GPS* v pixelech
- V- referenční zeměpisná šířka v sekundách
- V_{LU} - zeměpisná šířka levého horního rohu mapy v sekundách

- $\left(\frac{p}{s}\right)_V$ - koeficient určující kolik pixelů připadá na jednu sekundu zeměpisné šířky
- W_h - výška komponenty *Panel-GPS* v pixelech

Takto je také výpočet pozice mapy prováděn v programu. Pro úplnost dodávám, že východní zeměpisná délka a severní šířka jsou reprezentovány kladnými hodnotami, západní délka a jižní šířka zápornými. Referenční zeměpisná poloha je také ve spodní části panelu vypisována v konvenční formě (stupně, minuty, sekundy, světová strana). Pro převod do této formy zápisu a opačně je v programu zavedena struktura *Coordinate*. Jak vypadá výsledné zobrazování polohy komponentou je vidět na obrázku 3.7.

Obrázek 3.7: Ukázka zobrazení pozice na mapě komponentou *Panel-GPS*

KAPITOLA
ČTVRTÁ

ZÁVĚR

Tato kapitola shrnuje dosažené výsledky a přínos vytvořeného programu. Zmiňuji zde také některé návrhy toho, jak program v budoucnu vylepšit.

4.1 Výsledky práce

Protože zadání práce se skládalo z několika bodů, zhodnotím výsledek práce odpovídající každému z nich zvlášť několika větami.

- Program obsahuje přehledné grafické uživatelské rozhraní. Hlavní okno je rozděleno do podoken, jak je požadováno v zadání. Program obsahuje také standardní horní lištu, odkud lze volat další dialogová okna.
- Díky implementaci třídy *Recorder* dokáže program přijímaná data ukládat do externího souboru v uživatelem čitelné podobě. Třída *Player* naproti tomu data v takové podobě ze souboru načítá a umožňuje zobrazovat na komponentě grafu, případně ukazuje zaznamenanou polohu na mapě a zobrazuje pohybující se virtuální model základny. Zmíním však, že použitý způsob, kdy dochází při přehrávání dat k načtení celého souboru do paměti, není vhodný v případě hodně velkého vstupního souboru.
- Ve výše popsané formě lze v externím konfiguračním souboru programu definovat průběhy, se kterými má program pracovat. Po načtení tohoto souboru lze nadefinovaný komunikační protokol v programu prohlížet prostřednictvím dialogového okna.
- Program je rozšířen o podporu přenosu dat po sériové lince. Při přepnutí do režimu, kdy se komunikuje s externím zařízením přes sběrnici RS232, program přestane být ovladatelný a je nutné ho vypnout. Problém vyřeší změna implementace metod třídy *Serial*.
- Aplikace zobrazuje polohu na mapě díky návrhu a realizaci komponenty *Panel-GPS*. Potřebná mapa je načítána z externího souboru.
- Celý projekt je dokumentován formou html stránek. Jsou zde popsány jednotlivé třídy a metody. Vztahy mezi objekty dokreslují přiložené grafy. Dokumentace je součástí obsahu přiloženého CD.

4.2 Návrhy na vylepšení do budoucna

Program skýtá mnoho možností na rozšíření. Zmíním se zde o dvou konkrétních věcech.

Konfigurace komunikačního protokolu by šla rozšířit o možnost definice použitých jednotek, případně převodního koeficientu, což by se následně projevilo při zobrazování odpovídajícího průběhu v grafu.

Původní program měl mnoho ovládacích prvků pro řízení základny realizovaných mnoha grafickými komponentami. Tyto komponenty byly v programu pevně a vždy. V případě nového programu s konfigurovatelným komunikačním protokolem by bylo vhodné tento koncept změnit do nějaké nastavitelné formy, protože pro různé definice komunikačního protokolu budou nutné různé ovládací prvky. Takovou nastavitelností by se zajistila přítomnost pouze aktuálně použitelných grafických komponent.

4.3 Přínos této bakalářské práce

Hlavní výhodou tohoto programu je možnost jeho univerzálního využití. Díky konfigurovatelnému komunikačnímu protokolu ho lze použít např. pro čtení dat z inerciální jednotky. Zvolený způsob implementace dále dovoluje jednoduše rozšířit program o libovolný typ komunikačního rozhraní. V těchto bodech vidím hlavní přínos této práce.

LITERATURA

[1] http://www.doxygen.nl/download.html, [cit. 2010-05-15]

[2] http://www.graphviz.org/, [cit. 2010-05-15]

[3] http://zedgraph.org/wiki/index.php?title=Main_Page, [cit. 2010-05-22]

Obsah přiloženého CD

Přiložené CD obsahuje vlastní program, dokumentaci ve formě html stránek a tuto bakalářskou práci ve formátu pdf.