VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

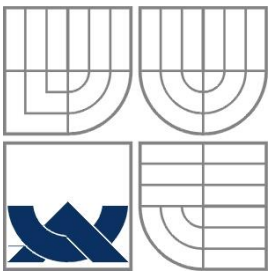
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TELEMETRIE PRO FORMULI DRAGON IV

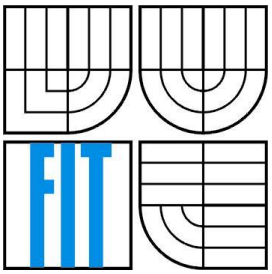
DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE Bc. Jan BEZDÍČEK
AUTHOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TELEMETRIE PRO FORMULI DRAGON IV

TELEMETRY FOR DRAGON IV FORMULA

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Jan BEZDÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Václav ŠIMEK

BRNO 2014

Abstrakt

Cílem této diplomové práce bylo navrhnout a zkonstruovat kompletní telemetrické zařízení určené pro formuli Dragon IV soutěže Formula Student. Tato práce se nejprve zabývá teorií problematiky měření fyzikálních veličin v automobilu, jejich vzájemnou komunikací a komunikací se sběrnicí CAN. Je zde popsán postup návrhu hardwaru včetně výběru a použití jednotlivých inerciálních senzorů a GPS modulu. Jsou zde obsaženy i podklady pro tvorbu dvouvrstvé desky plošného spoje, která rozšiřuje mikropočítač BeagleBone Black o inerciální senzory a modul GPS. Součástí této práce je také vytvoření softwaru v jazyce C# a C++ a to jak firmwaru pro hardware, tak aplikace pro řídící počítač. Tato uživatelská aplikace slouží pro bezdrátový příjem stavu senzorů získaných telemetrickým zařízením a jejich zobrazování. Umožňuje také bezdrátově zařízení konfigurovat. Výsledný produkt je kompletní telemetrický systém již vhodný pro koncového zákazníka.

Abstract

The aim of this master's thesis was to design and construct complete telemetry system for the student formula Dragon IV constructed for international Formula Student competition. At first, the work deals with the measurement of the physical quantities, telemetry system and automotive sensors of the formula, their mutual communication and communication with the CAN bus. It also describes the procedure of hardware design including choosing right inertial sensors and a GPS module and their using in telemetry system. The work contains materials for production of two-layer printed circuit board extending the microcomputer BeagleBone Black on the inertial sensors and the GPS module. The bigger part of the telemetry system is the firmware for hardware and software for the computer user. Both written in programming language C++ and C# are included in this work as well. This user application serves for wireless receiving data from the hardware and their showing and logging. In addition this user application can be used for wireless hardware configuration. The final product is the complete telemetry system and it is suitable for selling to end customer.

Klíčová slova

Telemetrie, Formula Student, GPS, NMEA 0183, CAN sběrnice, akcelerometr, gyroskop, inerciální senzory, MEMS, senzory v automobilu, Wi-Fi, vestavěné zařízení

Keywords

Telemetry, Formula Student, GPS, NMEA 0183, CAN Bus, Accelerometer, Gyroscope, Inertial Sensors, MEMS, Automotive sensors, Wi-Fi, Embedded System

Citace

Bezdíček Jan: Telemetrie pro formuli Dragon IV, diplomová práce, Brno, FIT VUT v Brně, 2014

Telemetrie pro formuli Dragon IV

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Václava Šimka a konzultací s panem Ing. Alešem Marvanem.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jan Bezdíček
7. května 2014

Poděkování

Tímto děkuji za vedení a odbornou podporu při tvorbě této diplomové práce vedoucímu panu Ing. Václavu Šimkovi, jakožto i konzultantovi panu Ing. Aleši Marvanovi a studentské skupině TU Brno Racing.

Obsah

1 Úvod

V této diplomové práci se zabývám návrhem kompletního telemetrického systému, tedy systému umožňujícím měřit na dálku. Jedná se o zařízení, jehož cílem bude získávání přes sběrnici CAN stavů všech senzorů studentské formule Dragon IV. Navíc zařízení obsahuje i akcelerometr, gyroskop, kompas a GPS modul. Všechna tato získaná data bude zařízení bezdrátově odesílat do řídícího počítače, ve kterém budou naměřená data zobrazována. Součástí diplomové práce je jak fyzické vyrobení telemetrického zařízení, tak napsání firmwaru pro toto zařízení a také aplikace pro řídící počítač, která umožní naměřená data zobrazovat a také bezdrátově konfigurovat hardware. Celou diplomovou práci můžeme rozdělit do tří logických celků.

První část diplomové práce vychází ze semestrálního projektu. V něm jsem se zabýval získáním teoretických znalostí nutných pro návrh telemetrie pro měřením dat v závodním jednomístném automobilu (monopostu). V následující kapitole to bylo přiblížení mezinárodní soutěže Formula SAE/Formula Student, pro kterou je zmiňovaný monopost vyvíjen. V kapitole 3 uvádím význam slova telemetrie, její historií, využití a možný budoucí vývoj. 4. kapitola se zabývá problematikou měření fyzikálních veličin v automobilu a rozebrání jednotlivých senzorů zde používaných. Senzory jsou rozděleny podle měřené fyzikální veličiny do jednotlivých podkapitol. Poslední podkapitola této kapitoly se věnuje speciálním senzorům, mezi které patří inerciální senzory a geolokační přístroje. V 5. kapitole je popsána komunikační sběrnice CAN používaná v automobilech pro komunikaci mezi jednotlivými jednotkami i jako rozhraní pro přístup k datům získaných z jednotlivých senzorů.

V druhé části této diplomové práce začínající kapitolou 6 jsem se zabýval návrhem hardwaru. Tento návrh je rozdělen podle logických celků do jednotlivých podkapitol. První z podkapitol se zabývá jádrem celého zařízení mikropočítačem BeagleBone Black. Další podkapitoly pak popisují připojení CAN sběrnice, bezdrátové rozhraní, GPS modul a objasňují princip nastavení tohoto hardwaru. Poslední podkapitola se zabývá návrhem rozšiřujícího modulu s inerciálními senzory včetně návrhu schématu zapojení a desky plošného spoje.

Třetí a zároveň poslední část této práce se zabývá návrhem softwaru. Kapitola 7, kterou tato část začíná, se zabývá návrhem softwaru pro hardwarovou část. Jsou zde popsané jednotlivé knihovny sloužící pro komunikaci se sběrnicemi CAN, I^2C a UART i knihovny a zdrojový kód sloužící pro řízení hardwaru. Kapitola končí shrnutím všech použitých tříd pro řízení hardwaru a zobrazením metrik kódu. Kapitola 8 se zabývá návrhem aplikace pro řídící počítač. Jedná se o relativně složitou aplikaci bezdrátově komunikující s hardwarem. Tato aplikace obsahuje mnoho tříd, které budou popsány v podkapitolách této kapitoly. Budu zde rozebírat pouze hlavní funkční prvky aplikace, protože detailní popis všech metod by přesáhl rozsah této diplomové práce. Nicméně veškeré zdrojové kódy jsou dobře komentovány a společně s podrobnou technickou dokumentací umístěny v příloze této práce na CD nosiči a čtenáři tak kdykoliv přístupné.

V poslední kapitole shrnuji dosažené výsledky a poukazuji na další možný vývoj a rozšíření tohoto telemetrického systému i pro jiné aplikace.

2 Soutěž Formula Student

Tato kapitola čerpá z [1] a [2]. Soutěž Formula Student je celosvětová konstrukční soutěž technických univerzit, ve které mají studenti za úkol postavit jednomístný závodní speciál, s kterým následně studenti závodí v statických i dynamických disciplínách a demonstrují tak svoje inženýrské znalosti a schopnosti. Její historie se traduje až do roku 1981, kdy soutěž vznikla v USA pod názvem Formula SAE. O 17 let později, v roce 1998, se soutěž dostala i do Evropy a to pod názvem Formula Student pro monoposty se spalovacími motory a později Formula Student Electric pro monoposty poháněné elektromotorem. V současné době je Formula SAE nejuznávanější vzdělávací automobilová soutěž. Jejím cílem je inspirovat a rozvíjet mladé studenty. V roce 2013 se touto soutěží zabývalo více než 500 týmů z celého světa, z toho 6 týmů z České republiky a do budoucna se očekává, že se toto číslo bude stále zvyšovat.

2.1 Pravidla

Soutěž Formula Student je ryze studentský projekt ve kterém kromě konstrukční stránky je nutno umět také pracovat v týmu a řešit věci jako jsou marketing, ekonomika a další oblasti nutné pro chod týmu a zdárnému zkonstruování závodního monopostu. Proto tým obsahuje studenty nejrůznějších fakult od studentů strojního inženýrství, přes studenty informačních a elektrotechnických směrů až k studentům ekonomického zaměření nutných pro správu financí a zajišťování sponzorů. Monopost zapsaný týmem do soutěže musí být vymyšlen, navrhnut, vyroben a spravovaný pouze studenty bakalářského nebo magisterského studijního programu bez přímého podílení se profesionálních závodníků, automobilových inženýrů a dalších profesionálů. Studenti by si měli vyrobit co největší část komponent, nicméně vyrobení komponent externí firmou je také povoleno.

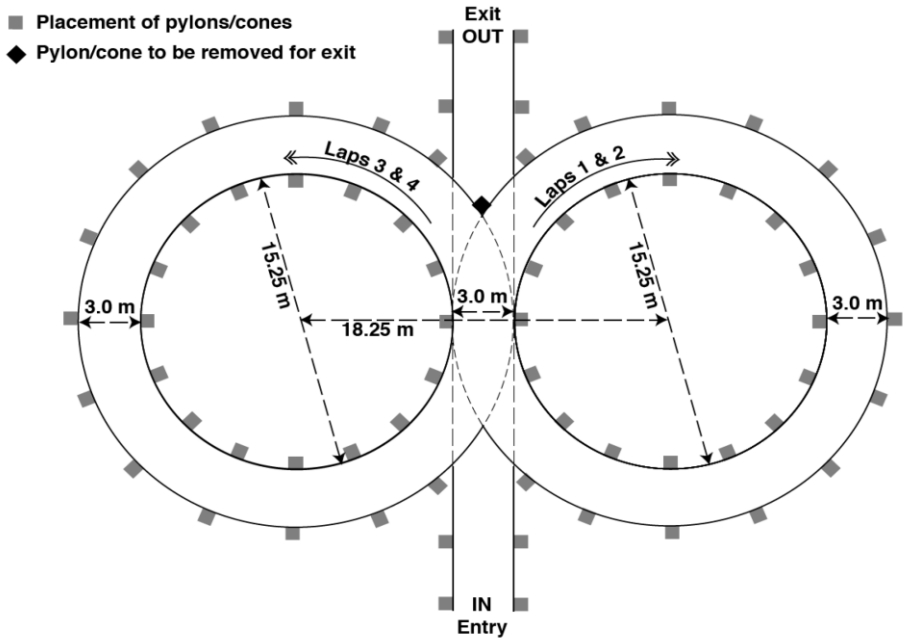
V soutěži Formula Student jde především o to, aby si studenti zkusili práci na reálném projektu a přenesly teoretické znalosti do praxe, proto je celá soutěž omezena přísnými pravidly, aby nedošlo k ohrožení ničí bezpečnosti. Pro každý rok se vydává nová verze pravidel [2], které jsou striktní a stejné pro všechny zúčastněné týmy. Navíc pořadatel každého závodu může obecná pravidla rozšířit o další svoje pravidla, proto je nutné sledovat i je. Příkladem zde může být například Formula Student Germany [3]. Před přistoupením k samotnému závodu jsou týmy nuceny dokázat znalost těchto pravidel správným a včasným splněním vstupního testu. Bez úspěšného splnění testu není týmu dovoleno se závodu zúčastnit.

Každá část monopostu má svoje relativně detailně popsaná pravidla, čímž je docíleno vyrovnaného souboje mezi jednotlivými týmy. Z velkého množství jednotlivých pravidel bych zmínil například pravidlo, které přesně definuje minimální velikost prostoru pro řidiče a jeho ochranné prvky jako je trubkový rám nad hlavou a nohama řidiče, 6bodový bezpečnostní pás, nehořlavý oblek a další. Významným omezením je také použití čtyřdobého spalovacího motoru o maximálním objemu 610 ccm a 20mm restriktoru v sání. Mezi konstrukční pravidla patří například přesně daný minimální průměr a tloušťka ocelových trubek při jejich použití jako rámu, parametry předního deformačního členu, či oddělení kokpitu řidiče od motoru plechovou přepážkou. Další pravidla se týkají výhledu řidiče, ovládacích prvků, odpružení, kol, brzd, řízení, světel, elektroniky, designu. U každé části je přesně definováno, co je povoleno a co naopak zakázáno a před každým závodem jsou všechny části monopostu důkladně zkontrolovány komisaři, zda vyhovují pravidlům. To vede k tomu, že výsledné

monoposty jednotlivých týmů si jsou podobné a o to víc se ve výsledném pořadí projeví kvalita a důmyslnost návrhu jednotlivých částí.

2.2 Hodnocení

Závod vyhrává tým, který získá největší počet bodů, přičemž týmy nejsou hodnoceny pouze podle nejrychlejších časů v závodech, ale také jsou bodovány v statických disciplínách. Mezi ně patří hodnocení designu, finanční náročnost vývoje a prezentace monopostu popisující jeho přednosti a postup výroby. Po statických disciplínách následují dynamické disciplíny, ve kterých se měří časy projetí monopostu různými typy tratí. Jednou z nich je měření akcelerace formule na trati dlouhé 75 metrů. Dalšími jsou pak Skid-Pad viz obr. č. 2.1 pro měření zatáčecích schopností, Autocross pro měření ovladatelnosti, kdy automobil musí projet kužely přesně vytyčenou dráhu obsahující pasáže se slalomem, rovinky i krátké a dlouhé zatáčky v co nejlepším čase. Posledním typem trati je 22 km dlouhá trať Endurance, kde se testuje spolehlivost a výdrž monopostu. Při tomto závodě se již jede i s ostatními závodníky, přičemž jejich předjíždění je povoleno pouze na k tomu vyhrazených místech. Při tomto závodu se měří jak nejrychlejší čas, tak i množství spotřebovaného paliva, z čehož se pak hodnotí efektivnost automobilu.



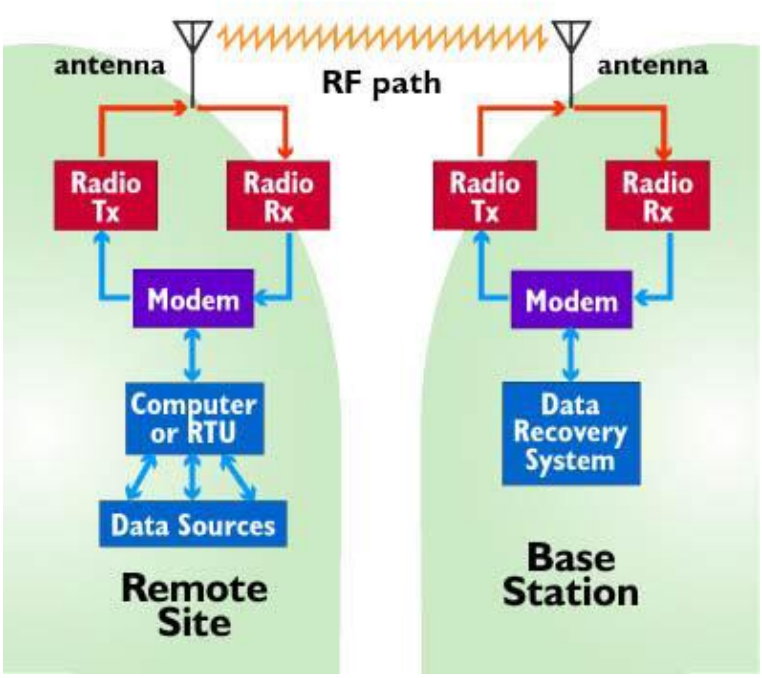
Obr. č. 2.1 Trať Skid-Pad [2]

2.3 Tým TU Brno Racing

Tato podkapitola čerpá z [4]. Tým TU Brno Racing se této prestižní soutěže zúčastňuje letos již 4. rokem pod záštitou Vysokého učení technického v Brně jakožto technické univerzity. Vyvíjený monopost nese označení Dragon s číslem odpovídajícím zúčastněnému roku. V tomto roce tým pracuje na vývoji již 4. monopostu s označením Dragon IV, ve kterém se plánuje reálné využití telemetrie zkonstruované v rámci této diplomové. Tým se skládá z 20 až 40 studentů a jeho cílem je především navrhnout a zkonstruovat jednomístné závodní vozidlo pro účast v mezinárodních soutěžích Formula Student.

3 Telemetrie

Slovo telemetrie vzniklo spojením dvou řeckých slov, tele=vzdálený a metron=měřidlo a je definována, jako snímání a měření informací z nějakého vzdáleného místa [5]. Základní koncept telemetrie je znám už po staletí, kdy byly používané nejrůznější metody i média přenosu. V současnosti je slovo telemetrie spojováno s bezdrátovým přenosem dat pomocí radiových nebo infračervených vln, ale stále existují i telemetrické systémy využívající kabelového spojení jako je například telefonní či počítačová síť. Bezdrátová telemetrická zařízení nám poskytují hned několik výhod oproti klasickým drátovým systémům. Nejvýznamnějším přínosem je především to, že můžeme měřit data v místech, kde by bylo velmi nepraktické, nebo zcela nemožné použít kabelového spojení. Tyto telemetrické zařízení jsou využívány stále častěji a v nejrůznějších oborech. Jejich využití najde uplatnění v oborech, kde je nutné sledování vodních průtoků, migrace živočichů až po vysoce sofistikované obory jako je navádění raket ve vojenství, přenos signálu z vesmírných družic v kosmonautice, nebo právě měření jízdních dat v závodním automobilu. Základní schéma telemetrického systému je zobrazeno na obr. č. 3.1.

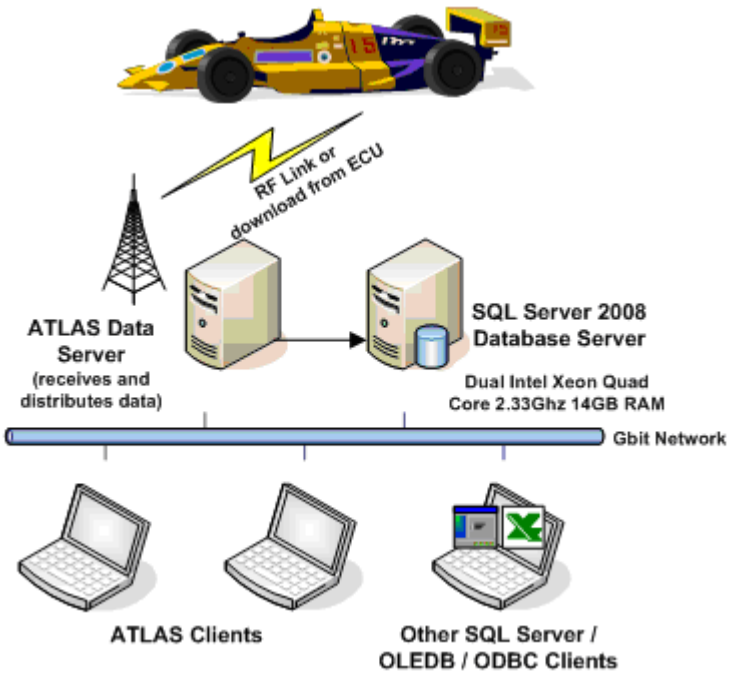


Obr. č. 3.1 Struktura typického telemetrického systému [5]

Datovými zdroji (Data sources) jsou ve vzdálené síti označovány senzory, poskytující nám data. Výstup těchto senzorů je převeden do elektronické podoby a ve vhodném formátu pomocí modemu odesílán k domovské stanici. V domovské stanici jsou přijatá data analyzována a dále zpracovávána příp. vhodně zobrazována. Komunikace mezi těmito stanicemi může být jednosměrná, nebo obousměrná, kdy domovská stanice může například nastavovat vzdálenou stanici, nebo s ní jinak komunikovat.

3.1 Telemetrie v závodech F1

Ve světě závodů je i pouhá sekunda velmi cenná, a proto týmy stále hledají způsoby, jak dosáhnout rychlejšího času. Toho můžou dosáhnout lepším nastavením automobilu, k čemuž o něm potřebují znát přesné údaje v každém okamžiku. A tady nastupuje na řadu telemetrie, která nám umožňuje snímat stav jednotlivých senzorů po celou dobu závodu, aniž bychom museli být ve fyzickém kontaktu s vozem. Telemetrická jednotka ve vozidle získává data ze všech senzorů vozidla a tyto data obsahující informace o motoru, odpružení, převodovce, palivové nádrži, teplotách, silách působících na vozidlo i chování pilota odesílá technikům do depa pro další zpracování. Obecné schéma je zobrazeno na obr. č. 3.2.



Obr. č. 3.2 Obecné schéma telemetrie v závodech F1 [6]

Tento odstavec čerpá z [6]. Telemetrie se v závodech F1 začala používat už v 80 letech, kdy týmy posílaly data v dávkách při průjezdu kolem depa. Postupem technologií a také nutností znát data z vozu ihned a ne pouze jednou za okruh technologie na začátku 90 let povýšila na real-time zasílání dat, kdy jsou data zasílána ihned po jejich získání z čidel vozu. To ale přineslo další komplikace v podobě nutnosti řešit ztrátu signálu během závodu, což se může dít například při průjezdu vozu tunelem, jak je to běžné například na trati v Monaku. Tento problém byl vyřešen až po roce 2000 průběžným ukládáním získaných dat ve vozu a při ztrátě a následném obnovení signálu jsou ještě nepřenesená data v dávce přeneseny, a tak nedojde k jejich ztrátě. V současné době je tento systém ještě podpořen rozmístěním více antén po celém závodním okruhu, což razantně snížilo množství nepokrytých míst.

V roce 2002 byla, sice pouze na pár let, povolena obousměrná telemetrie, která umožňovala nejen stahovat data z vozidla, ale také dálkově měnit jeho nastavení [7]. V současné době je ale obousměrná telemetrie v závodech F1 zakázána, opět se ale uvažuje o její povolení.

Závodní monoposty běžně obsahují 150 až 300 nejrůznějších senzorů, které budou probrány v dalších kapitolách, umožňující získat co nejpřesnější data o vozidle. Tyto data jsou z vozidla bezdrátově přenášeny do depa, což dělá asi 1,5 miliardy vzorků a zhruba 5 GB dat, za závod. Toto obrovské množství vzorků proto musí být před odesláním z vozidla zkomprimováno a samozřejmě zašifrováno. Po přijmutí dat z vozidla jsou data dekódována a zobrazena v čitelné podobě. V závodech F1 se k tomuto převodu a zobrazení využívá systém ATLAS (Advanced Telemetry Linked Acquisition Systém) zavedený MES (McLaren Electronics Systems), který se stal standardem pro všechny týmy [8]. Výstup z tohoto systému je zobrazen v příloze č. 1.

Snímání dat z vozu v reálném čase nám umožní upravit některá nastavení vozu při vjezdu do depa i informovat řidiče prostřednictvím vysílačky o důležitých naměřených datech při jízdě. Ale stejně tak, ne-li více, je důležité data uchovávat pro jejich pozdější analýzu.

3.2 Telemetrie v studentské formuli Dragon IV

V studentské formuli Dragon IV byla snaha vytvořit co nejkvalitnější telemetrické zařízení. Inspiraci jsem získával právě v telemetrii používané v závodech F1. Telemetrické zařízení vyvíjené v rámci této diplomové práce snímá data ze všech senzorů v monopostu vysílaných po CAN sběrnici a prostřednictví Wi-Fi sítě je společně s daty získanými z inerciálních senzorů a údajem o aktuální poloze dál distribuuje do depa, příp. k dalším i mobilním zařízením. Součástí diplomové práce byl i vývoj softwaru pro následné zobrazování naměřených dat, viz další kapitoly.

4 Měření fyzikálních veličin v automobilu

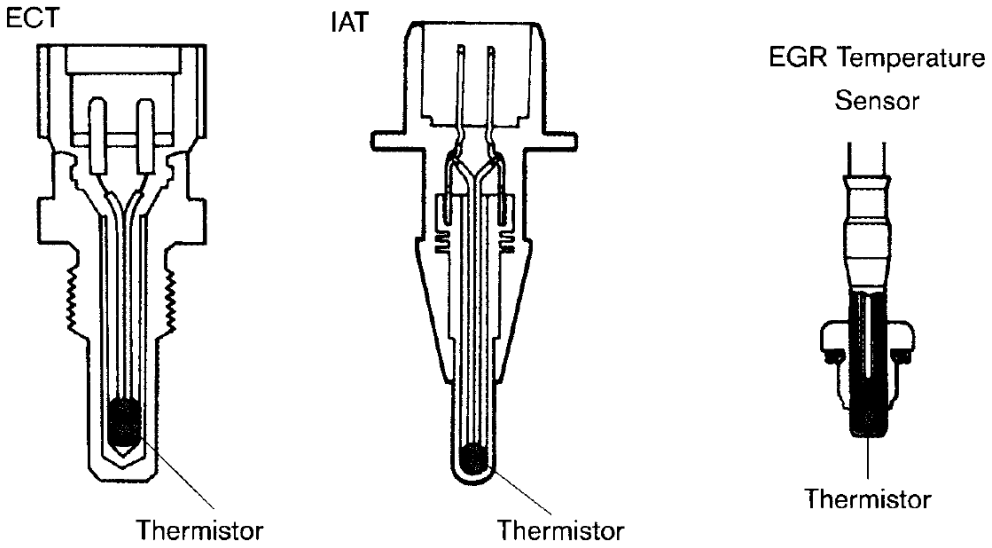
Tato kapitola řeší problematiku měření fyzikálních veličin v závodních automobilech a používaných senzorech pro jejich měření. Pro získání co nejpřesnějšího stavu automobilu potřebujeme použít velké množství senzorů, v současné době jsou to desítky až stovky senzorů a toto číslo neustále stoupá. Výstupy všech těchto senzorů jsou připojeny k řídící jednotce vozu, která data z těchto senzorů dále zpracovává a ovládá aktuátory a další zařízení vozidla. Pro externí přístup k datům z jednotlivých senzorů, ať už z důvodu diagnostiky závady automobilu, či připojení dalšího systému (např. telemetrie), nám řídící jednotka může nabízet vhodné rozhraní, nejčastěji CAN sběrnici. Největší množství senzorů slouží pro řízení motorů, další pak pro sledování chování celého vozu a speciální kategorií jsou inerciální senzory sloužící pro měření zrychlení a GPS senzory pro informaci o poloze. Toto velké množství senzorů se dá rozdělit do podkategorií podle měřené veličiny. Jsou to senzory pro měření teploty, tlaku, množství, pozice a zrychlení. Součástí dalších podkapitol je podrobná analýza jednotlivých typů senzorů.

Odstavec čerpá z [9]. V dnešní době se začínají využívat tzv. MEMS senzory (Micro-Electro-Mechanical Systems). Jedná se o velmi malé senzory, které obsahují elektroniku, ale hlavně miniaturní mechanické prvky. Díky přítomnosti jak mechanického subsystému nutného pro transformaci fyzikální veličiny na elektrickou, tak i elektronického subsystému pro následné zpracování elektrického signálu, se v souvislosti s těmito senzory hovoří i o tzv. inteligentních snímačích.

4.1 Měření teploty

Závodní automobil musí pracovat ve velmi náročných podmínkách. Teploty okolí se mohou pohybovat od teplot pod bodem mrazu až k tropickým teplotám. Když přihlédneme k velmi vysokým teplotám vznikajících při procesu hoření ve válci motoru, není překvapením, že závodní automobily obsahují velké množství senzorů teploty. Měří se zde teplota snad všeho. Nejdůležitější je pro nás hlídat teplotu chladící kapaliny, oleje a hlavy válců. Tato média dosahují největšího rozptylu teplot a přímo ovlivňují výkon motoru. Studený olej je pro nás stejně nebezpečný jako přehřátý. Teplotní čidla dále najdou uplatnění v měření teploty nasávaného vzduchu, výfukových plynů, brzdových a hydraulických kapalin, oleje v převodovce a v neposlední řadě i teplot pneumatik.

Pro měření teploty se používají především termistory, jejichž vnitřní odpor je závislý na jejich teplotě. Používají se dva základní typy termistorů, NTC a PTC. Rozdíl mezi těmito dvěma typy je v jejich reakci na zvyšující se teplotu. V případě NTC (Negative Temperature Coefficient) senzoru ze vzrůstající teplotou klesá jeho vnitřní elektrický odpor. U PTC (Positive Temperature Coefficient) senzoru je to přesně naopak. NTC typ senzoru se v automobilech používá častěji než typ PTC [10]. Obr. č. 4.1 zobrazuje typický tvar teplotních senzorů pro měření teploty chladicí kapaliny – ECT (Engine Coolant Temperature), nasávaného vzduchu – IAT (Intake Air Temperature) a výfukových plynů – EGR (Exhaust Gas Recirculation).



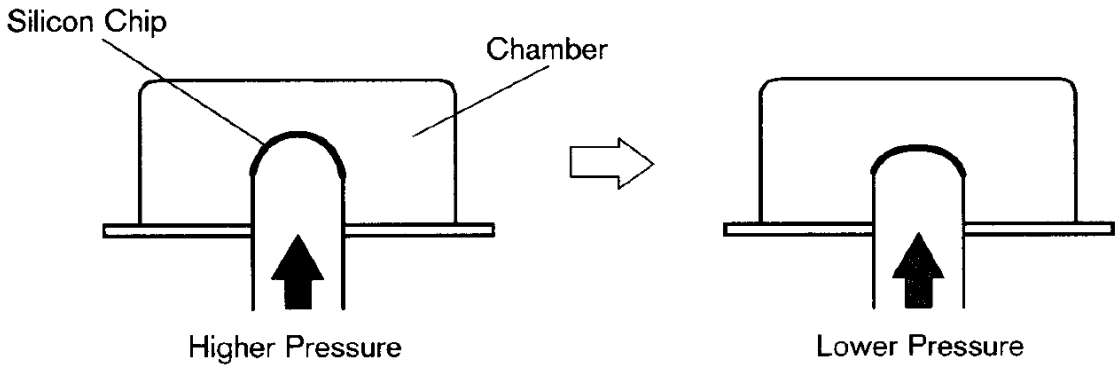
Obr. č. 4.1 Typický tvar teplotních senzorů používaných v automobilech [10]

Kromě termistorů se pro měření teploty používají také pyrometry, které nám umožní měřit teplotu bezkontaktně. Jejich využití je pro měření teploty pneumatik, což je také velmi důležité pro dosažení co nejlepších výsledků a pro měření teploty brzdového kotouče a brzdového obložení, což nás naopak může včas upozornit na přehřátou brzdovou soustavu a možné snížení brzdného účinku.

4.2 Měření tlaku

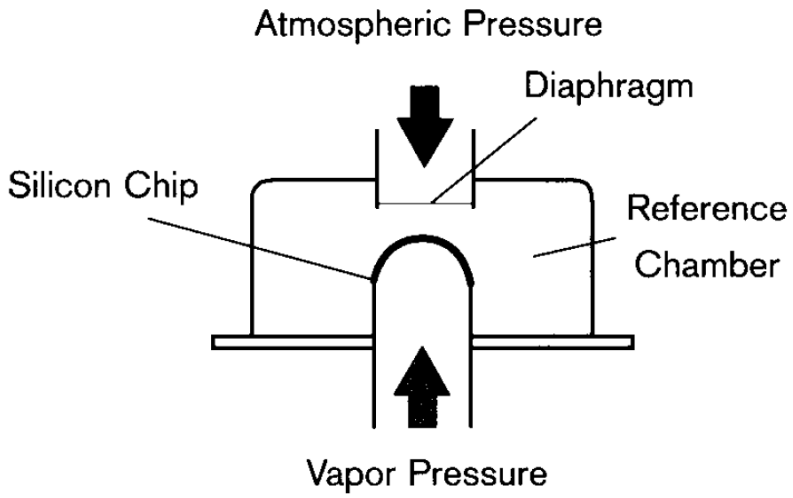
Automobilové motory pracují s různými tlaky, proto je k jejich řízení potřeba tyto tlaky měřit, vyhodnocovat a následně podle nich řídit chod motoru. V automobilovém průmyslu se nejčastěji používají piezorezistivní snímače tlaku s křemíkovou membránou, kterými můžeme měřit tlaky kapalin i plynů [11]. Pro řízení motoru potřebujeme měřit především množství motorem nasátého vzduchu. To můžeme provést buď přímo měřením množství proteklého vzduchu, jak bude rozebráno v další podkapitole zabývající se měřením množství, nebo měřením podtlaku v sacím potrubí. Měření tlaku plynu se dále používá například pro měření tlaku vzduchu v odpružení, přetlaku či podtlaku v palivové nádrži, tlaku ve spalovacím prostoru pro rozpoznání vynechávání zapalování a klepání, tlaku vzduchu při použití přeplňování s turbodmychadlem, barometrického tlaku, nebo tlaku vzduchu v pneumatikách, což je dokonce v Evropské unii od roku 2012 povinné i pro všechny civilní automobily [12]. Měření tlaku kapalin využijeme například u snímání tlaku oleje, tlaku kapaliny v brzdovém okruhu, modulačního tlaku u automatických převodovek, nebo tlaku paliva ve vstřikovacím zařízení.

Odstavec čerpá z [9]. Každý senzor se liší podle svého použití, ale princip funkce zůstává u všech senzorů zachován. Opět, stejně jako u teplotních senzorů, se jedná o senzory, které mění svůj vnitřní odpor, tentokrát v závislosti na tlaku. Uvnitř senzoru se nachází nejčastěji křemíková membrána, na kterou z jedné strany působí přesně stanovený tlak, v případě podtlakových čidel vakuum, a na druhou stranu membrány je přiváděný měřený tlak. Membrána pak v závislosti na tlaku mění svůj tvar, jak je zobrazeno na obr. č. 4.2.



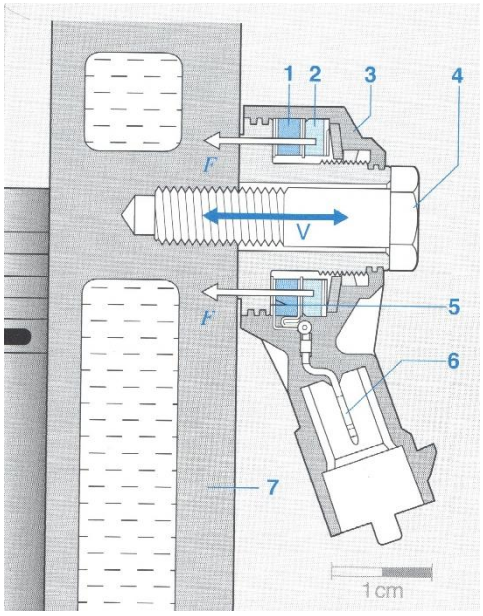
Obr. č. 4.2 Princip činnosti tlakového čidla [13]

V některých případech je nutné senzor neustále kalibrovat podle jiného tlaku. To je typické pro senzor snímání přetlaku v palivové nádrži. Při zahřívání paliva dochází k uvolňování plynů, které je nutné odpouštět, aby nedošlo k výbuchu. K rozpoznání, zda je v nádrži přetlak vzhledem k atmosférickému tlaku, je nutné použít senzor, který pracuje na principu zobrazeném na obr. č. 4.3. Tento typ senzorů je doplněn o další membránu, na kterou je přiváděn atmosférický tlak. Tato membrána mění referenční tlak uvnitř senzoru a tím jej neustále kalibruje.



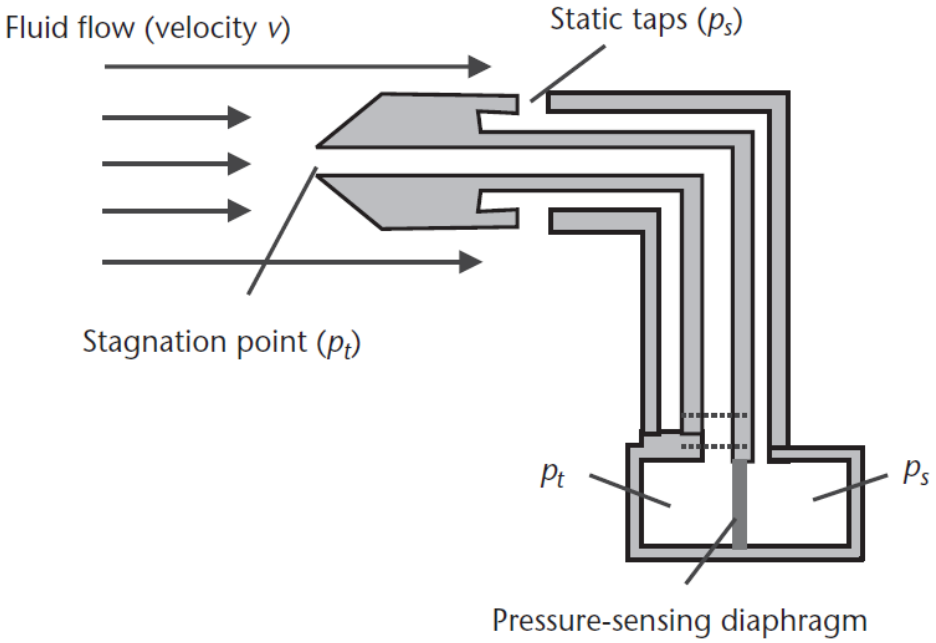
Obr. č. 4.3 Tlakové čidlo s kalibrací atmosférickým tlakem [13]

Pokud bychom křemíkovou membránu nahradili destičkou z barium-titanové keramiky, která vykazuje piezoelektrický jev, to znamená, že při působení určitou silou na destičku bude vznikat elektrické napětí, dostáváme senzor deformací, resp. vibrací [14]. Tento senzor se v automobilismu používá jako čidlo klepání pro předejití tzv. detonačního spalování. Při detonačním spalování jsou v motoru vytvářeny silné tlakové změny poškozující motor. Tyto tlakové změny se šíří pomocí vibrací, které jsou snímány právě pomocí čidla klepání, a tím je umožněno na tyto nechtěné vibrace okamžitě reagovat. Obr. č. 4.4 zobrazuje čidlo klepání firmy Bosch.



Obr. č. 4.4 Čidlo klepání (1 - piezokeramika, 2 - seizmická hmotnost s tlačnými silami F, 3 - pouzdro, 4 - šroub, 5 - připojení, 6 - elektrická přípojka, 7 - blok motoru, V - vibrace) [11]

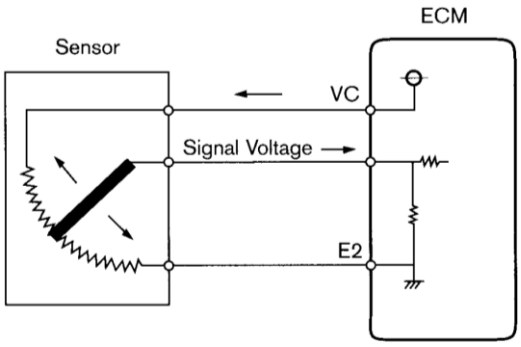
Mezi senzory tlaku patří i Pitotova trubice [9]. Tato trubice, pojmenovaná po svém objeviteli, umožňuje měřit rychlost proudění média, například vzduchu. Závodní týmy ji používají na svých vozech pro měření rychlosti proudění vzduchu v různých místech vozidla, což jim umožňuje co nejlépe odhadnout a následně i nastavit jeho aerodynamiku. Princip tohoto zařízení je jednoduchý a je zobrazen na obr. č. 4.5. Pitotova trubice se skládá ze dvou spojených trubic, kdy jedna je nasměrována proti směru proudění vzduchu, měří dynamický tlak, a druhá je kolmá k směru proudění a měří statický tlak. Rozdíl tlaků v těchto dvou trubicích je vyhodnocován snímačem tlaku a odpovídá rychlosti proudění plynu, případně kapaliny.



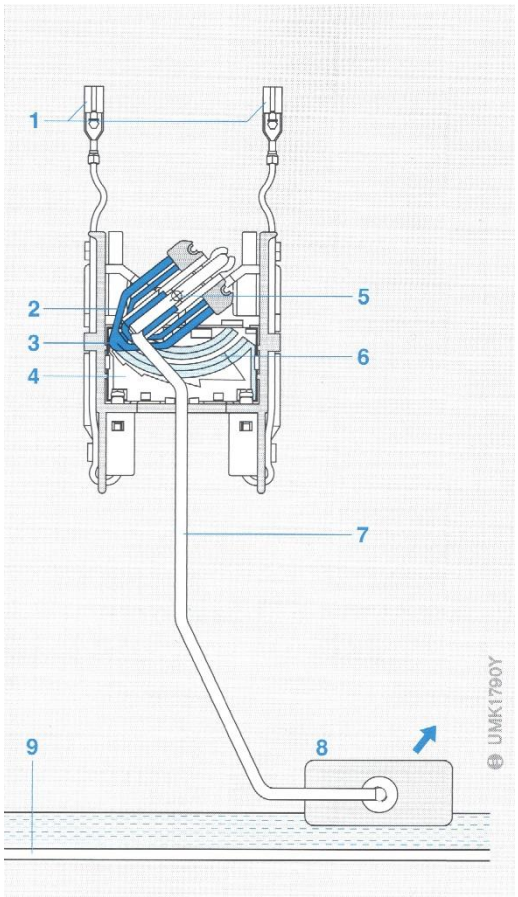
Obr. č. 4.5 Pitotova trubice [9]

4.3 Měření pozice

Při měření pozice dochází k převodu neelektrické veličiny na elektrickou. K tomu slouží jednoduchý snímač, potenciometr, u kterého se působením neelektrické měřené veličiny mění poloha pohyblivého kontaktu, tzv. jezdce, vůči odporové dráze, která může být přímočará nebo kruhová. Dráha potenciometru může být taktéž lineární nebo logaritmická. Pro převod změny odporu na změnu napětí jsou potenciometry zapojeny jako děliče napětí. Princip činnosti a zapojení je znázorněn na obr. č. 4.6. Použití tohoto snímače je typické například pro měření množství paliva v nádrži, viz obr. č. 4.7, polohy jednotlivých pedálů, pozici EGR ventilu, nebo míry stlačení jednotlivých pružících jednotek. Ve všech těchto případech je použito potenciometrů s přímočarou odporovou dráhou.

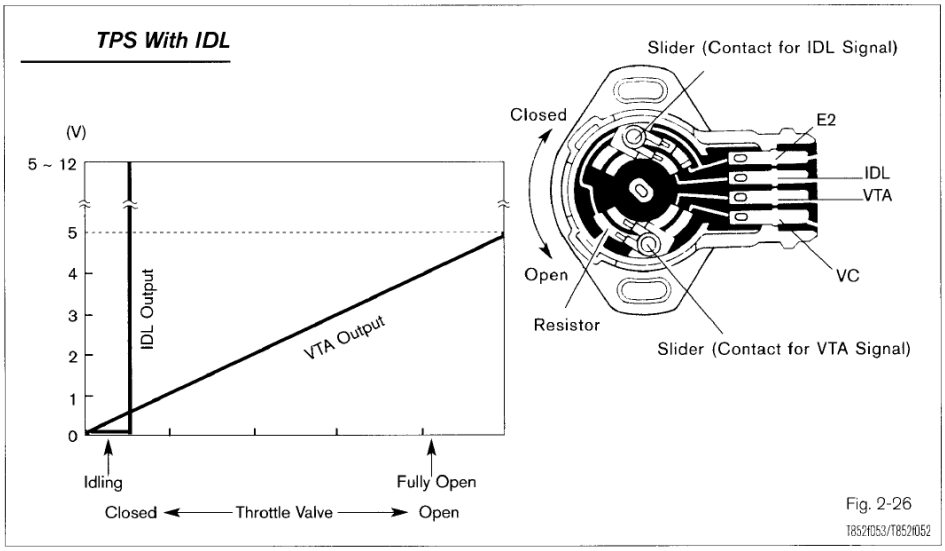


Obr. č. 4.6 Měření polohy potenciometrem [15]



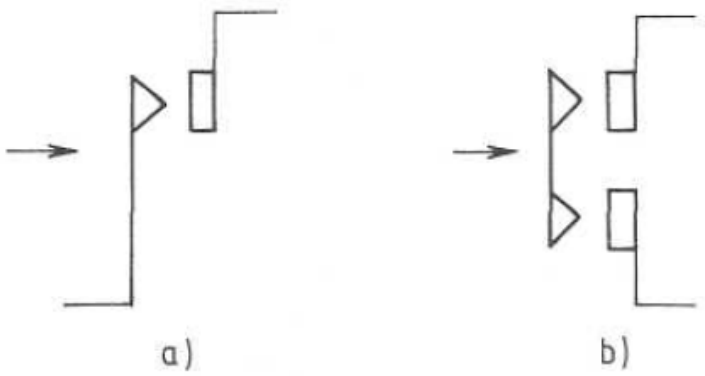
Obr. č. 4.7 Snímač hladiny paliva (1 - elektrické přívody, 2 - pružina běžce, 3 - kontaktní nýt, 4 - deska s rezistory, 5 - hřídel potenciometru, 6 - dvojitý kontakt, 7 - páka plováku, 8 - plovák, 9 - dno nádrže) [11]

Potenciometry s kruhovou odporovou dráhou lze najít například na senzoru natočení volantu, zařazeného převodového stupně, nebo úhlu natočení škrticí klapky. Odporové snímače mohou být doplněny další odporovou dráhou, která zabezpečí přesnější a spolehlivější měření, nebo přináší další funkcionalitu, jako je to tomu například a čidla natočení škrtící klapky, kdy potřebujeme mít hodnotu blížící se krajní hodnotě potenciometru nastavenou na přesnou 0. Princip činnosti je nastíněn na obr. č. 4.8.



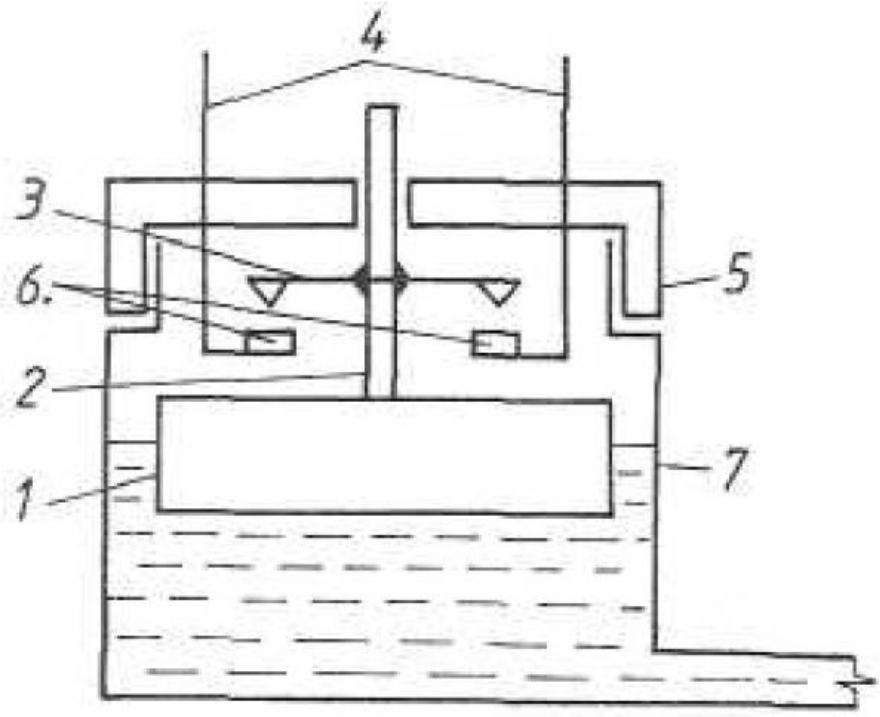
Obr. č. 4.8 snímač natočení škrtící klapky s druhou dráhou pro detekci stavu blízkého počátku [15]

Tento odstavec čerpá z [14]. Nejjednodušším typem odporového snímače je kontaktní spínač [14]. Při změně sledované neelektrické veličiny dochází ke skokové změně odporu spínače v okamžiku sepnutí nebo rozepnutí jeho kontaktů. Vstupní veličina zde není měřena, pouze je indikován její stav. Provedení kontaktních spínačů zobrazuje obr. č. 4.9. Změnou sledované veličiny může dojít buď k spojení vodiče s kostrou automobilu, v takovém případě nám stačí k spínači vést pouze jeden vodič, nebo v případech, kdy se první způsob nedá použít, se použije druhý způsob, zobrazený na obr. č. 4.9b, kdy k spínači vedeme dva vodiče, které se změnou sledované veličiny pomocí vodivého můstku spojí, či rozpojí.



Obr. č. 4.9 Provedení kontaktních spínačů (a – pohyblivý kontakt, b – pohyblivý můstek) [14]

Kontaktních spínačů se dá využít například pro měření kritického množství kapalin, typicky brzdové a chladící, opotřebení brzdových destiček a také pro ovládací tlačítka, kdy dochází k spínání kontaktů. Například u snímačů kritického množství kapaliny je vodivý můstek spojen s plovákem a při poklesu hladiny dojde k spojení vodičů a indikaci kritického množství kapaliny. Princip je zobrazen na obr. č. 4.10.



Obr. č. 4.10 Princip měření kritického množství kapaliny pomocí kontaktního spínače (1 – plovák, 2- táhlo, 3 – kontaktní můstek, 4 – konektory, 5 – víčko nádobky, 6 – pevné kontakty, 7 - nádobka) [14]

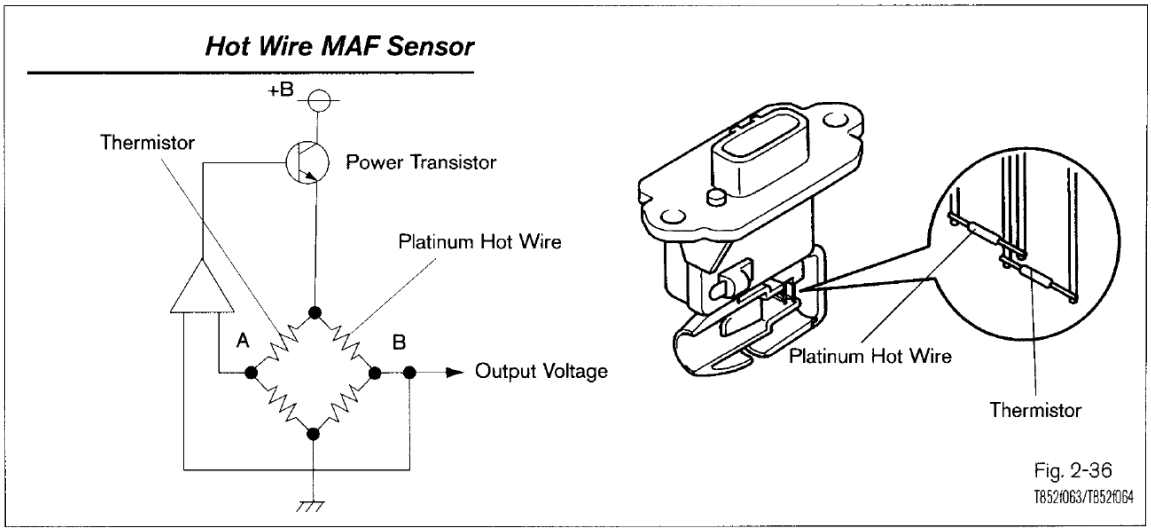
Odporové snímače polohy jsou velmi jednoduchá zařízení, která ovšem relativně dobře plní svou funkci. Nicméně jsou to snímače založená na mechanickém pohybu, proto u nich dochází k opotřebování odporové dráhy. Stejně tak jsou náchylná k vibracím, při nichž dochází k nadzvednutí jezdce, a k změně teplot, jejichž změnou dochází i k změně odporu dráhy [11].

4.4 Měření množství

Pro měření množství je už nutno použít složitější senzory než byly probírány v minulých podkapitolách. Měření množství se používá pro měření vzduchu proudícího do motoru, množství kyslíku ve výfukových plynech, množství nečistot v oleji, nebo také napětí v palubní síti.

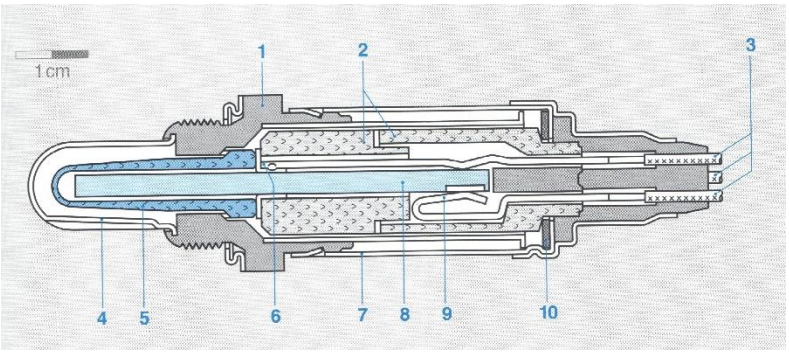
Odstavec čerpá z [11]. Jeden z nejdůležitějších senzorů pro řízení motorů je senzor měřící množství vzduchu proudícího do motoru, protože pro přesné nadávkování paliva pro vytvoření stechiometrické směsi potřebujeme znát přesné množství přijatého vzduchu, resp. kyslíku. Jednou z možností je jeho měření pomocí vytvářeného podtlaku, jak bylo rozebráno v kapitole 4.2. V této podkapitole se ale budeme zabývat dalšími způsoby jeho měření.

V historii byly zkoušeny nejrůznější metody. Jednou z nich je například vychylování lopatek proudícím vzduchem, ale v dnešní době se používá především MAF (Mass Air Flow) senzor. Tento senzor obsahuje platinový drátek o průměru 5 – 7 μm a termistor měřící teplotu vzduchu. Principem funkce je pak umístění platinového drátku do proudícího vzduchu a jeho ohřívání přesně měřeným proudem na teplotu zhruba o 130 °C vyšší než je teplota nasávaného vzduchu, jak je zobrazeno na obr. č. 4.11. Tento drátek je průběžně ochlazován proudícím vzduchem a tato změna teploty je ihned kompenzována zvýšením vyhřívacího proudu. Vyhřívací proud je tedy závislý na aktuálním průtoku vzduchu. Výstupem senzoru je pak napětí převedené z naměřeného vyhřívacího proudu.



Obr. č. 4.11 Princip funkce MAF senzoru [16]

Odstavec čerpá z [11]. Dalším neméně důležitým senzorem v automobilu je senzor měření množství nespáleného kyslíku ve výfukových plynech. Tento senzor se nazývá lambda sonda a udává nám dokonalost spalování ve válci motoru. Díky tomuto senzoru řídící jednotka motoru přesně upravuje poměr směsi paliva a vzduchu. Konstrukce tohoto senzoru je složitá a kvůli použití v agresivním prostředí výfukových plynů musí být použity drahé kovy jako je platina a zirkon, přesněji oxid zirkoničitý s přísadou oxidu vápenatého [14]. Jádrem senzoru je pevný keramický elektrolyt s těmito drahými kovy, který tvoří galvanický článek porovnávající množství kyslíku ve vzduchu s množstvím nespáleného kyslíku ve výfukových plynech viz obr. č. 4.12. Podmínkou pro správnou funkci lambda sondy, je její zahřátí na teplotu vyšší než 400 °C. Proto je sonda doplněna vyhříváním, které nám zaručí její rychlé zahřátí. Po jejím zahřátí je dále ohřívána výfukovými plyny, proto bývá typicky umístěna co nejblíže motoru. V současné době je běžné používání více lambda sond. Jedna sonda se umístí před katalyzátor co nejblíže motoru a slouží pro měření kvality spalování a řízení poměru spalovací směsi a další sonda se umístí za katalyzátor, kde kontroluje jeho bezchybnou funkci.



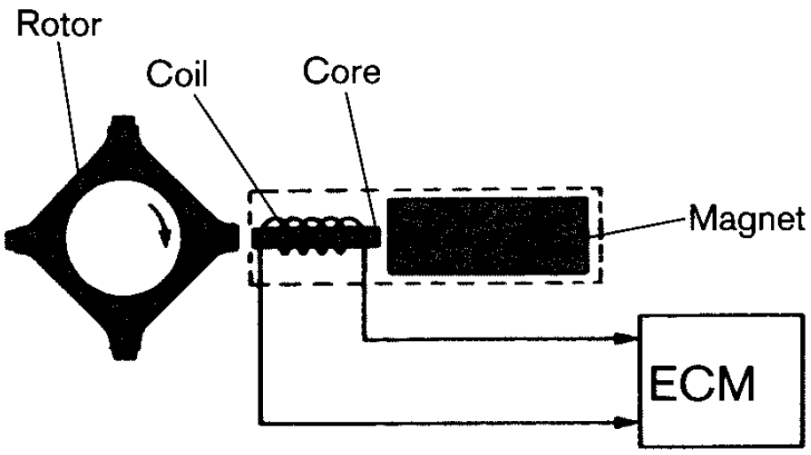
Obr. č. 4.12 Vyhřívaná lambda sonda (1 - pouzdro sondy, 2 - keramická opěrná trubice, 3 - připojovací kabel, 4 - ochranná trubka se štěrbinami, 5 - aktivní keramika sondy, 6 - kontaktní díl, 7 - ochranná objímka, 8 - vyhřívací prvek, 9 - připojovací svorky pro vyhřívací prvek, 10 – talířová pružina) [11]

Do této skupiny senzorů se dále může zařadit senzor napětí v palubní síti, což vlastně není senzor, ale pouze vhodný napěťový převodník, nebo senzor znečištění oleje. Tento senzor je založen na elektromagnetické detekci kovových částic přitahovaných k magnetu tohoto senzoru [17].

4.5 Senzory pohybu

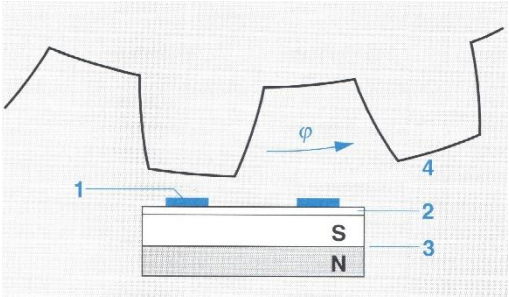
Tato podkapitola čerpá z [11]. Senzory pohybu, nebo lépe řečeno senzory otáček, jsou senzory snímající nějakou rotující část. Jejich výstupem jsou pak otáčky této rotující části. Mohou být založeny na elektromagnetickém jevu (indukční), Hallově jevu, nebo optoelektronickém jevu.

Elektromagnetické senzory, obr. č. 4.13, jako jediné nepotřebují napájení. Skládají se z trvale magnetické části a cívky, která je opakovaně přikládána k magneticky měkkému železu. Otáčením ozubeného kola dochází k změně magnetického toku, což poté na cívce generuje napětí střídavého průběhu podobného sinusovce. Výsledná frekvence nám udává počet otáček ozubeného kola. Nevýhodou tohoto typu snímače je, že velikost výstupního napětí je závislá na otáčkách, proto musíme použít další elektroniku pro úpravu tohoto napětí. Výhodou naopak je, že senzor nepotřebuje napájení a jeho výroba je jednoduchá a levná.



Obr. č. 4.13 Elektromagnetický senzor otáček [18]

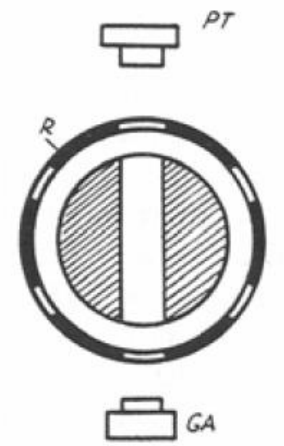
Senzory založené na Hallově jevu využívají dvou tenkých polovodičových destiček, kterými protéká elektrický proud a zároveň prochází magnetické pole z permanentního magnetu. Přibližování a oddalování jednotlivých zubů otáčejícího hřídele dochází k změnám magnetického pole a tím i k změně Hallova napětí. Hallův senzor se často používá v diferenciálním uspořádání, kdy jsou 2 Hallovy senzory umístěny na jednom čipu a elektronika potom vyhodnocuje rozdíl těchto Hallových napětí viz obr. č. 4.14. Stejně jako u elektromagnetických senzorů je i u Hallových senzorů výstupní frekvence napětí závislá na otáčkách, ale pro tyto senzory je výstupní napětí stále konstantní. Další výraznou výhodou použití diferenciálních Hallových senzorů je jejich schopnost detekce směru otáčení. Tím, ale výhody Hallových senzorů končí. Oproti elektromagnetickým senzorům jsou tyto senzory díky použití citlivé elektroniky omezeny rozsahem teplot na max. asi 150 °C a také potřebují napájení a odebírají značný proud.



Obr. č. 4.14 Diferenciální Hallův senzor (1 - Hallovy snímače, 2 - magnetický měkké železo, 3 - permanentní magnet, 4 - ozubené kolo) [11]

Optoelektronické snímače otáček, obr. č. 4.15, se skládají ze zdroje světelného záření, například LED diody, a fotodiody, příp. fototranzistoru. Dopad světelného záření ze zdroje na fotodiodu je přerušován clonkou a řídící elektronika vyhodnocuje impulsy napětí na fotodiodě, které odpovídají otáčkám clonky. Výhodou je konstantní amplituda výstupního napětí, nevýhodou pak požadavky na čistotu snímačů.

Všechny tyto typy snímačů otáček se v automobilech využívají a jejich použití závisí na konkrétních požadavcích na cenu, spolehlivost apod. Můžeme je najít například na klikové hřídeli, kde měří otáčky motoru a dávají impuls k zapalování, nebo na vačkové hřídeli, kde jsou nutné pro řízení variabilního časování ventilů pro jednotlivé válce. Stejně tak na jejich použití narazíme v převodové skříni pro měření rychlosti automobilu a u každého kola pro rozpoznání otáček jednotlivých kol nutných pro asistenční systémy řízení a brždění jako jsou ABS, ESP a další.

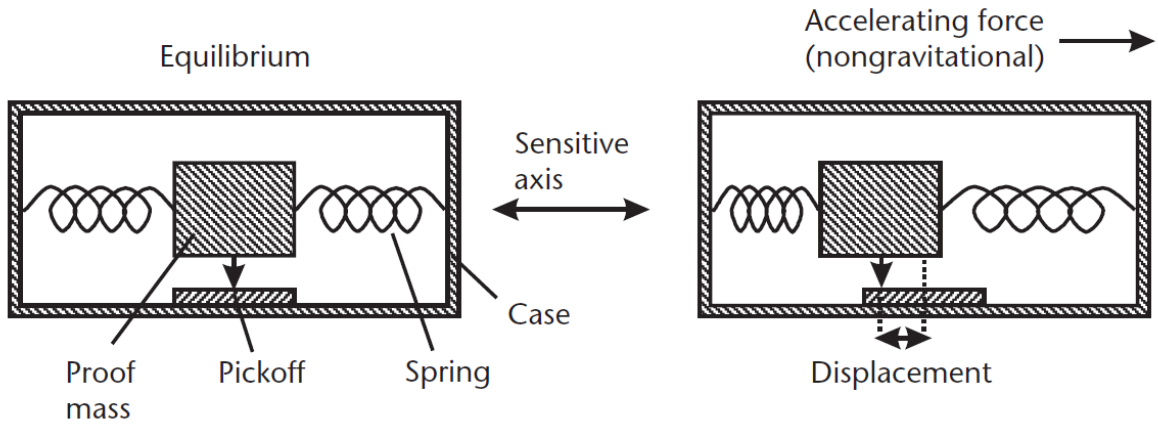


Obr. č. 4.15 Optoelektronický snímač otáček – překreslit [14]

4.6 Speciální senzory

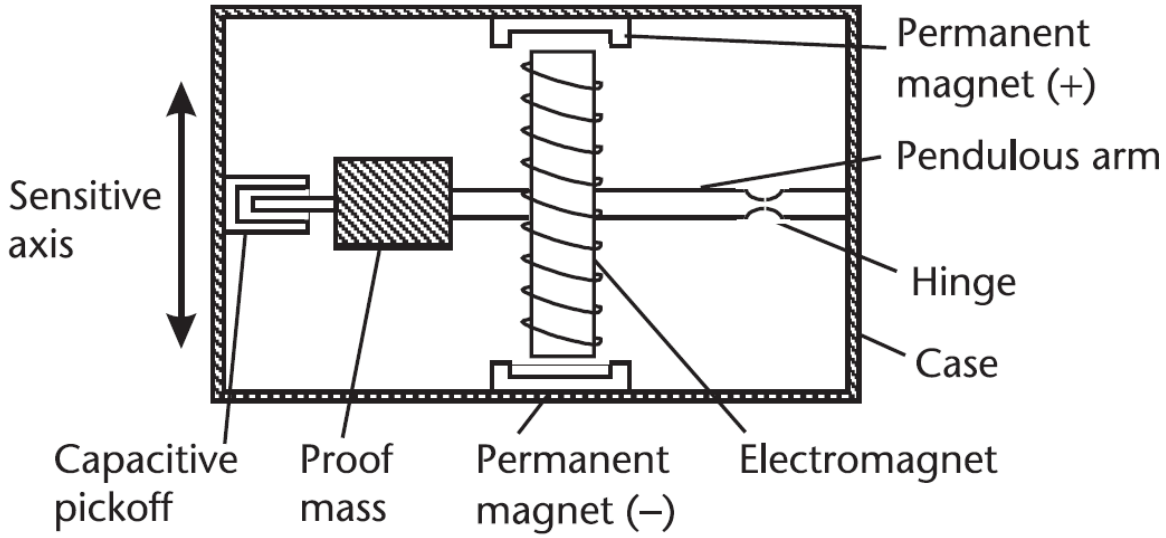
Původně se jednalo o drahé senzory používané pouze ve vojenské technice, ale s nástupem moderní techniky jsou i osobní automobily stále více vybavovány inerciálními senzory a senzory polohy. Inerciální senzory slouží k měření lineárního a úhlového zrychlení.

Odstavec čerpá z [19]. Pro měření lineárního zrychlení v jedné nebo více osách slouží senzor zvaný akcelerometr. Tento senzor se skládá z tělesa, volně zavěšeného z každé strany na pružince. Při vyvíjení zrychlení v ose pružinek dochází ke zmáčknutí jedné a protažení druhé pružinky, nebo naopak. Elektronický snímač pak měří vychýlení tělesa. Toto vychýlení převádí na elektrický signál, který odpovídá aktuálně vyvíjenému zrychlení na osu senzoru, jak je zobrazeno na obr. č. 4.16. Akcelerometry bývají vybaveny více těmito bloky s pružinkami, pro měření zrychlení ve více směrech.



Obr. č. 4.16 Princip funkce akcelerometru pro měření zrychlení v jedné ose [19]

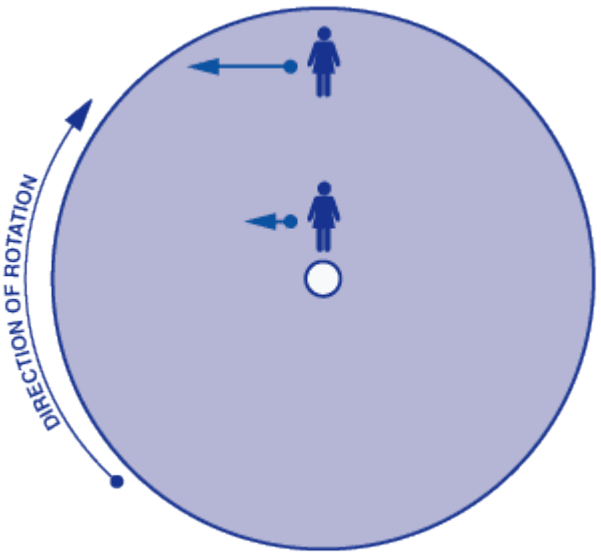
Obr. č. 4.16 je pouze ilustrativní pro zobrazení základní funkce akcelerometru. Ve skutečnosti by vychýlení zavěšeného tělesa ovlivňovalo zrychlení i v ostatních osách, proto se používá akcelerometr s kyvadélkem. Vlastnosti pružinek se navíc s časem i teplotou mění, proto se v reálných případech místo pružinek používá elektromagnet. Takovýto typ akcelerometru je zobrazen na obr. č. 4.17.



Obr. č. 4.17 Akcelerometr s kyvadélkem a elektromagnetem [19]

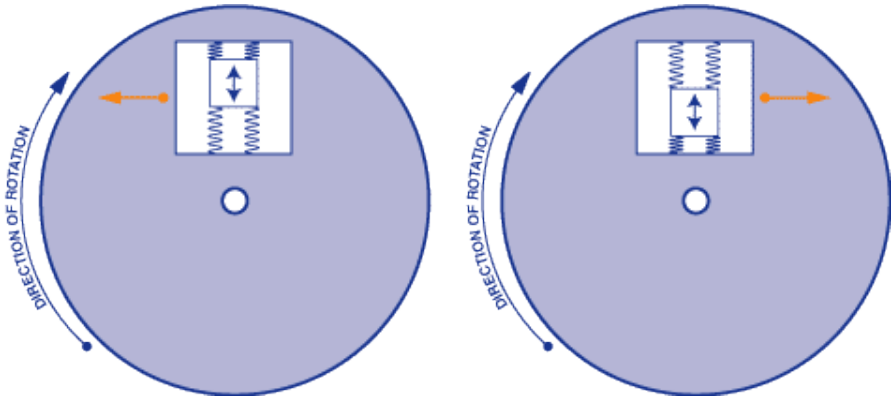
Snímání míry vychýlení tělesa může být snímáno různými metodami závisících na konkrétní aplikaci. Mezi tyto metody patří například kapacitní metoda, piezorezistivní, piezoelektrická, optická, nebo metoda tunelových proudů [9]. Velikost akcelerometrů v automobilech je v řádu jednotek milimetrů, ale pro měření pomalejších a zároveň větších zrychlení, jako například na lodích, můžou akcelerometry dosahovat rozměrů až desítek centimetrů.

Odstavec čerpá z [19]. Pro měření úhlového zrychlení slouží senzory zvané gyroskopy. Gyroskopy pro měření rychlosti otáčení měřeného objektu využívají Coriolisovy síly. Princip této síly je ten, že na objekt pohybující se po rotujícím objektu z jeho středu k jeho okrajům působí vzrůstající Coriolisova síla, která má největší velikost u kraje kotouče. Princip je znázorněn na obr. č. 4.18, kde Coriolisova síla je zaznačena modrou šipkou od postavy.



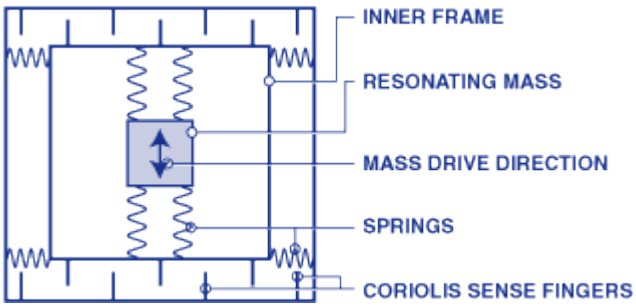
Obr. č. 4.18 Princip Coriolisovy síly [20]

Tohoto účinku se využívá v mechanických gyroskopech, jak je zobrazeno na obr. č. 4.19. Při pohybu tělesa zavěšeného na pružinkách směrem od středu otáčejícího se disku, působí Coriolisova síla směrem doleva. Při pohybu tělesa ke středu otáčení, pak působí tato síla směrem doprava. Velikost a směr této síly je úměrný rychlosti a směru otáčení kotouče, proto jej lze využít k měření velikosti úhlové rychlosti.

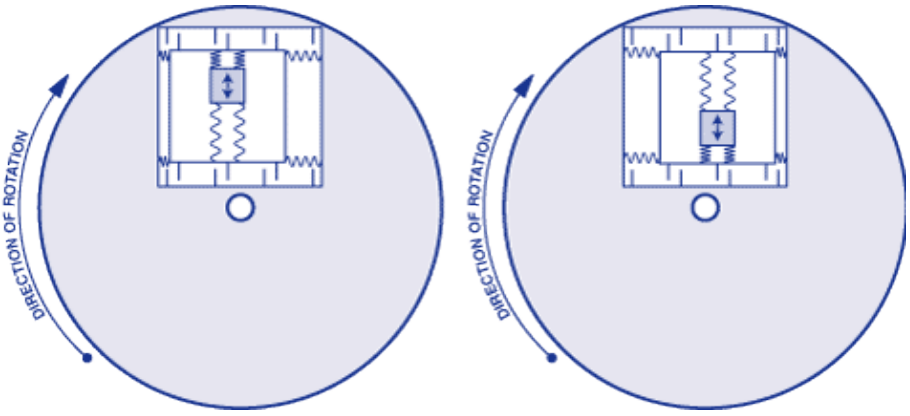


Obr. č. 4.19 Působení Coriolisovy síly na kmitající objekt [20]

Měření této síly pak probíhá tak, že prvek s tělesem kmitajícím o přesné frekvenci je pomocí pružinek zavěšen do dalšího rámečku viz obr. č. 4.20. V tomto vnějším rámečku se měří vychýlení například pomocí kapacitní metody, kdy se mění vzdálenost elektrod vzduchových kondenzátorů. Tato změna kapacity je opět elektronikou převáděná na výstupní napětí, které odpovídá úhlové rychlosti otáčení v stupních za sekundu. Celý tento rámeček je umístěn do senzoru u umožňuje měřit rychlost jeho otáčení jak je zobrazeno na obr. č. 4.21.

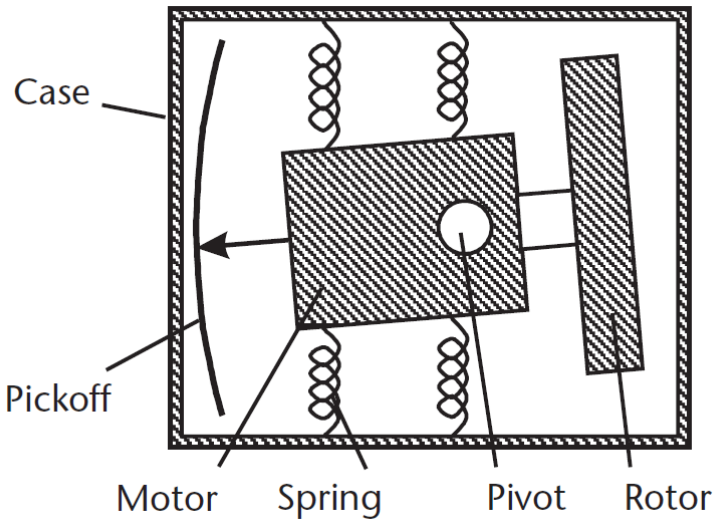


Obr. č. 4.20 Měření Coriolisovy síly působící na prvek s kmitajícím tělesem [20]



Obr. č. 4.21 Provedení gyroskopického senzoru [20]

Odstavec čerpá z [19]. Kromě vibračních gyroskopů popsaných výše se v minulosti používaly také gyroskopy s rotorem, jejichž princip je zobrazen na obr. č. 4.22. Jedná se o motor zavěšený na pružinkách umožňující svůj pohyb pouze v podélné ose. Tento motor roztáčí rotor na vysoké otáčky. Působením točivé síly je soustava motoru a rotoru vychylována ze své osy, což je snímáno a odpovídá velikosti točivého momentu působícím na senzor. Tento princip gyroskopických senzorů byl nahrazen vibračními gyroskopickými senzory a v současné době se již nepoužívá.



Obr. č. 4.22 Gyroskopický senzor s rotorem [19]

Rozdíl mezi akcelerometrickými a gyroskopickými senzory je ten, že akcelerometrické senzory měří hodnotu zrychlení vznikajícího zpomalováním nebo urychlování objektu, kdežto gyroskopické senzory nám měří rychlost ustáleného rotačního pohybu. Tyto dva senzory nám umožňují zjistit přesné informace o pohybu, proto bývají často spojovány do jednoho senzoru a používány společně. Jejich využití v automobilech je například pro systém airbagů, nebo pro stabilizační systémy pro přesné vyhodnocení chování vozu a možnosti na tyto změny systémem adekvátně reagovat.

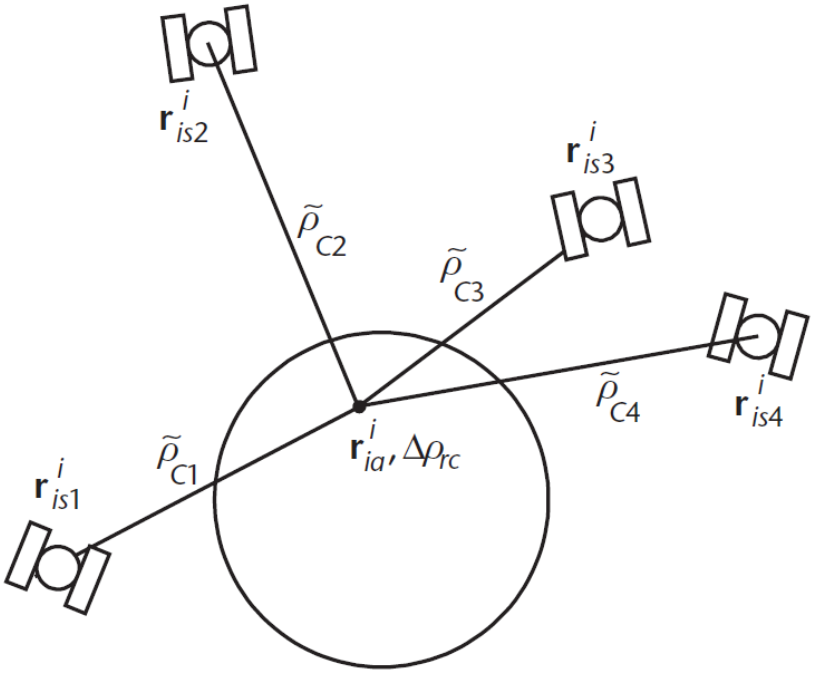
Odstavec čerpá z [19]. Posledním typem speciálních senzorů jsou satelitní navigační systémy. Nejznámějším a nejrozšířenějším z nich je americký systém NAVSTAR GPS, často zkracován na pouze GPS, který původně sloužil pouze pro vojenské účely, ale nyní je dostupný i veřejnosti. Kromě

tohoto systému existují ještě ruský navigační systém GLONASS a evropský navigační systém Galileo. V době psaní této diplomové práce systémy GLONASS ani Galileo ještě nejsou plně funkční, proto se dále budu zabývat pouze navigačním systémem GPS. Princip funkce všech těchto tří navigačních systémů je ale stejný. Celý systém se skládá ze tří segmentů. Kosmický segment, řídící segment a uživatelský segment.

Kosmický segment se skládá z 24 až 36 družic obíhajících Zemi na 6 oběžných drahách ve výšce 20 100 km nad povrchem země rychlostí 3,8 km/s. Každá družice má svoji přesnou polohu a obsahuje velmi přesné atomové hodiny, 12 vysílacích antén pro vysílání geolokačního signálu na Zem a další antény pro komunikaci s pozemními kontrolními stanicemi a ostatními družicemi.

Řídící a kontrolní segment je umístěn na Zemi a skládá se z řídících a monitorovacích stanic. Funkce tohoto segmentu je zasílání povelů družicím, pohyb těchto družic a údržba jejich atomových hodin. Pokud by došlo ke zničení těchto pozemských vojenských stanic řídícího a kontrolního segmentu, jsou družice schopny pracovat ještě 6 měsíců v autonomním módu.

Posledním segmentem systému je uživatelský segment. Uživatelské GPS přijímače přijímají signál z právě viditelných družic obsahující přesnou časovou značku družice. Přijímače jsou díky znalosti přesné pozice družice a obdržené časové značce schopni určit její vzdálenost. Při obdržení tohoto signálu od alespoň tří družic dochází k triangulaci a přesnému zaměření polohy na zemi. V případě obdržení signálu od alespoň 4 družic je kromě zeměpisných souřadnic možné zjistit i aktuální nadmořskou výšku. Princip triangulace je zobrazen na obr. č. 4.23. Výhled GPS přijímače k satelitům na oběžné dráze nesmí být ničím zastíněn, např. budovou, horou apod., jinak dochází k odrazu a zkreslení signálu a přesnost určení výsledné pozice klesá.



Obr. č. 4.23 Triangulace pozice s určením nadmořské výšky [19]

GPS přijímače jsou pasivní zařízení, které pouze přijímají signál z družic a žádný signál nevysílají. V současné době se pohybuje přesnost určení polohy civilních GPS přijímačů v řádu metrů, a jsou omezeny maximální možnou výškou 18 km a maximální rychlostí 515 m/s [86]. Toto omezení je nastaveno pro předejití zneužití tohoto navigačního systému pro navádění raket. Pro vojenský sektor americké armády a další tzv. autorizované uživatele toto omezení neplatí a je jim z družic poskytována přesnější časová značka, která umožní dosáhnutí vyšší přesnosti určení polohy.

Pro kontrolu dlouhodobých výchylek jednotlivých družic v poslední době vznikly systémy EGNOS a WAAS, které se skládají z pozemních monitorovacích stanic s přesnou polohou a geostacionárních družic. Tyto stanice porovnávají svoji přesně danou polohou s polohou získanou z družic a v případě odchylky, či zjištění výpadku nějaké družice tuto informaci pomocí svých družic odesílají k uživateli. Systém EGNOS je systém fungující v Evropě a systém WAAS pak systém vylepšující vlastnosti GPS v Americe.

5 CAN sběrnice

Tato kapitola čerpá z [21] a [22]. V soudobých automobilech se často nachází více jednotek, které potřebují znát data z konkrétního senzoru. Vést ke každé jednotce zvlášť kabely z konkrétního senzoru by bylo nevýhodné, proto data ze všech senzorů jsou sbírána do jedné řídící jednotky, která je pak zpřístupňuje dalším jednotkám. Navíc tyto jednotky mohou potřebovat komunikovat s ostatními jednotkami. Proto pro komunikaci mezi všemi jednotkami bylo nutné vytvořit komunikační sběrnici, která by obsahovala co nejméně vodičů a zároveň zajišťovala spolehlivou komunikaci. S takovouto sběrnicí přišla v roce 1986 firma Bosch a pojmenovala ji CAN (Controller Area Network) [22].

Jedná se o sériovou diferenciální a symetrickou sběrnici komunikující po dvou vodičích označovaných jako CAN-L a CAN-H. K sběrnici je možné připojit až 64 zařízení dosahující při komunikaci na vzdálenosti kratší jak 40metrů rychlosti až 1 Mb/s [23]. Z popisovaných vlastností jako jsou velká odolnost proti šumu, velká přenosová rychlost, možnost komunikace více uzlů a další není divu, že i když byla tato sběrnice původně navrhnuta pro automobily, rozšířila se i do dalších průmyslových odvětví. Ve Finsku ji použila například firma Kone pro řízení výtahů, nebo v Nizozemí firma Philips Medical Systems pro přenos dat mezi rentgenovými přístroji [23]. K tomuto rozšíření pomohly i výrobci mikrokontrolérů, kteří vyrábí levné rozhraní a převodníky pro práci s touto sběrnicí. Standard sběrnice CAN definuje pouze nejnižší dvě vrstvy ISO/OSI modelu a to vrstvu linkovou a fyzickou, ale nedefinuje už vrstvu aplikační. Nicméně nedefinovaná aplikační vrstva je pro automobilový průmysl, který se snaží spíše o vzájemnou nekompatibilitu mezi jednotlivými výrobci, přínosem. S rozšířením této sběrnice i v jiných odvětvích se objevila tendence definovat i aplikační vrstvu, proto vznikl standard CAL (CAN Application Layer) z kterého později i standard CANopen a konkurenční DeviceNet [23]. Oba tyto standardy aplikačních vrstev protokolu CAN nachází uplatnění jak ve vestavěných systémech, tak v průmyslové automatizaci.

5.1 Komunikace na sběrnici

Sběrnice CAN pracuje v multi-master módu, kdy není žádný připojený uzel označený jako master, tudíž každý uzel může přijímat a vysílat data i řídit ostatní uzly. Stejně tak může být libovolný uzel odebrán či připojen do sběrnice za jejího běhu, proto ani výskyt vadného uzlu na sběrnici nemůže ohrozit komunikaci. Tato dvouvodičová sběrnice pracuje s dvěma stavy, recesivním a dominantním. Aktuální stav na sběrnici je dán rozdílovým napětím těchto dvou vodičů, ale logické hodnoty těchto stavů nejsou přesně definovány a liší se v závislosti na použité fyzické vrstvě. Ve většině případů je recesivní stav při rozdílovém napětí U_{diff}=0 V a dominantní stav při rozdílovém napětí U_{diff}=2 V [21]. Jestliže všechny uzly sběrnice vysílají recesivní bit, je na sběrnici recesivní stav, ale jestliže alespoň jeden uzel vysílá dominantní bit, na sběrnici je stav dominantní.

Komunikace mezi jednotlivými uzly probíhá pomocí zpráv. Každá zpráva obsahuje pouze svůj identifikátor, nikoliv svého příjemce či odesílatele. Tento identifikátor určuje data obsažená ve zprávě a také se podle jeho hodnoty určuje priorita zprávy. Jednotlivé uzly si samy mohou zvolit, zda zprávu s konkrétním identifikátorem přijmou či nikoliv, ale vždy musí po správném přijetí zprávy odpovědět

nastavením ACK bitu v přijímané zprávě na dominantní hodnotu. Při uvolnění sběrnice uzel začne vysílat svoji zprávu a zároveň čte data na sběrnici. Pokud by jeho vysílaná hodnota byla recesivní, ale na sběrnici by byla dominantní, tak díky tomu, že zpráva začíná identifikátorem, který je zároveň i ukazatelem priority by to znamenalo, že na sběrnici zároveň vysílá i jiný uzel s vyšší prioritou. Tomuto uzlu musí dát přednost a odložit si odeslání svoji zprávy na další uvolnění sběrnice. Díky tomu nedochází k přerušení vysílání uzlu s vyšší prioritou a tím i k nutnosti znova odeslat jeho zprávu a komunikace není nijak zpožděna.

5.2 Zabezpečení komunikace

Sběrnice CAN se vyskytuje ve velmi zarušených prostředích, kde se dá předpokládat častý výskyt chyb v přenosu, proto je vybavena pěti systémy pro detekci chyb. Jedná se o tyto mechanismy:

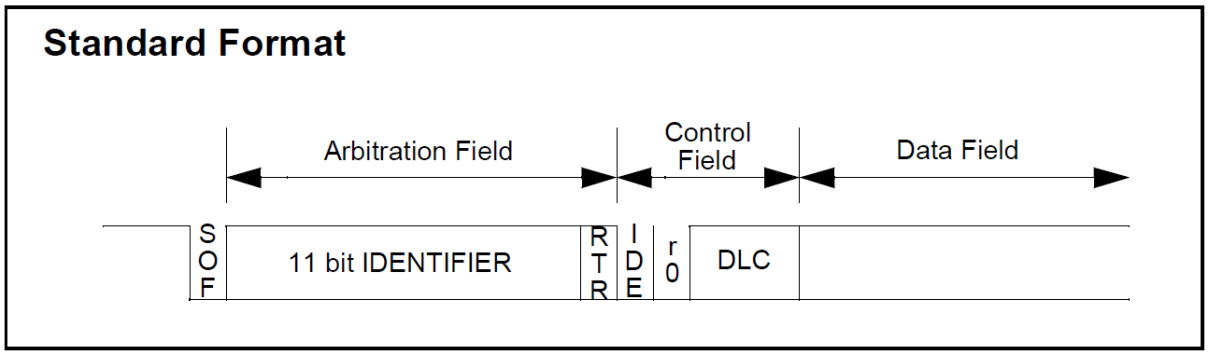
a) **Monitoring** – při odesílání dat jsou zároveň i kontrolována data na sběrnici. Pokud se od vysílaných dat liší a neprobíhá právě řízení přístupu k sběrnici, kdy se data liší z důvodů současného vysílání uzlu s vyšší prioritou, je vygenerována chyba.
b) **CRC kód** – každá vysílaná zpráva obsahuje CRC kód. Při obdržení zprávy se špatným CRC kódem dojde k vygenerování CRC chyby.
c) **Bit stuffing** – neboli vkládání bitu. Pokud se má na sběrnici vysílat pět stejných po sobě jdoucích bitů stejné úrovně, přidá se za ně další bit opačné úrovně. To umožní kromě detekce chyb také správné sesynchronizování jednotlivých uzlů.
d) **Kontrola zprávy** – kontroluje se formát přijaté zprávy a v případě detekování špatného formátu je vygenerována chyba
e) **Potvrzení přijetí zprávy** – ACK. Všechny zařízení připojená k sběrnici musí po obdržení zprávy bez chyby v ní okamžitě změnit ACK bit na dominantní hodnotu a tím potvrdit vysílači správné odeslání.

Každý uzel v sobě má interní počítadlo svých chyb a podle něj nastavuje svůj stav jako Aktivní, Pasivní, nebo Odpojený. Aktivní i Pasivní uzly mohou komunikovat na sběrnici. Aktivní uzly v případě detekce chyby vygenerují sled šesti po sobě následujících bitů dominantní úrovně, čímž zničí právě přenášenou zprávu a informují všechny uzly o výskytu chyby. Pasivní uzly na rozdíl od Aktivních vygenerují pouze sled šesti po sobě následujících bitů recesivní úrovně, čímž je přenášená zpráva zachována. Uzly ve stavu Odpojený se nemohou účastnit komunikace na sběrnici.

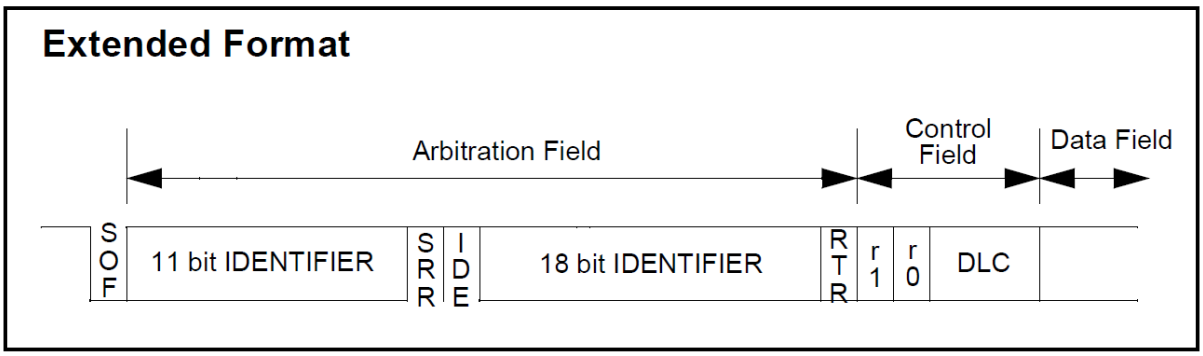
5.3 Typy zpráv

Tento odstavec čerpá z [22]. Standard CAN definuje dva typy zpráv, 2.0A a 2.0B. Specifikace 2.0A obsahuje pouze standardní zprávy, zatímco specifikace 2.0B obsahuje navíc i rozšířené zprávy. Tyto dva typy zpráv se liší především délkou identifikátoru, který je 11 bitů u jedné a 29 bitů u druhé specifikace. Rozdíl mezi těmito 2 specifikacemi je zobrazen na obr. č. 5.1 a obr. č. 5.2. Nejčastěji používané jsou datové zprávy. Tyto zprávy mohou obsahovat až 8 Bytů dat. Pokud bychom ve zprávě

neodeslali žádná data, pouze identifikátor, jednalo by se o zprávu, kterou žádáme o poslání dat s identifikátorem totožným jako je identifikátor zasílané zprávy. Dalšími dvěma typy zpráv je zpráva o chybě popsaná v podkapitole 5.2 a zpráva o přetížení, která je stejná jako zpráva o chybě, ale je vysílána ve chvíli kdy se začíná vysílat nová zpráva, tj. po obdržení konce předešlé zprávy.



Obr. č. 5.1 Struktura standardní zprávy CAN [22]



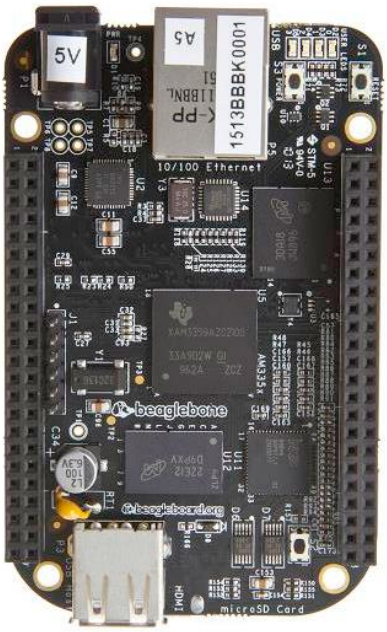
Obr. č. 5.2 Struktura rozšířené zprávy CAN [22]

6 Návrh HW

Jádrem telemetrie je komerčně prodávaný mikropočítač BeagleBone Black. Tento mikropočítač je rozšířen o tzv. CAN Bus Cape sloužící pro komunikaci se sběrnicí CAN a dalším rozšiřujícím modulem obsahujícím inerciální senzory a GPS modul.

6.1 BeagleBone Black

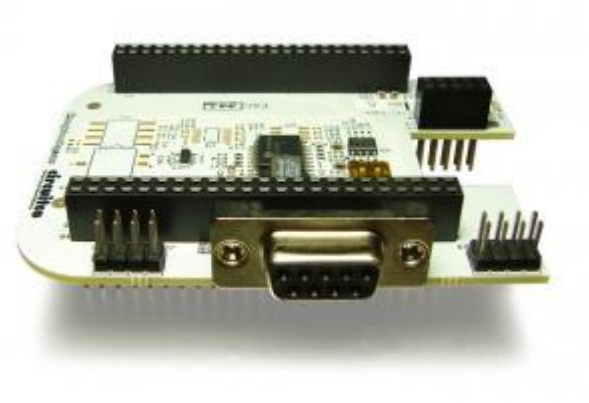
Tato kapitola čerpá z [24]. BeagleBone Black zobrazený na obr. č. 6.1 je malý a výkonný mikropočítač vyvíjený organizací BeagleBoard.org. Tento mikropočítač je osazen výkonným mikroprocesorem ARM Cortex A8 běžícím na frekvenci 1GHz, 512MB velkou DDR3 RAM pamětí a 2GB pamětí typu Flash. Dále tento mikropočítač disponuje řadou rozhraní jako je například microHDMI pro připojení externího monitoru, USB Host a USB Client, Ethernetovou přípojkou a dvěma 46 pinovými konektory pro připojení dalších komponent. Pro řízení mikropočítače jsem použil distribuci Linuxu Ångström, která je přímo určena pro vestavěné systémy a malá zařízení. Veškerá práce s tímto operačním systémem probíhala po připojení mikropočítače přes UTP Ethernetový kabel přes příkazovou řádku. Později, po zprovoznění Wi-Fi modulu, zařízení kromě napájení nepotřebovalo žádné další připojení a bylo možné se k němu připojovat bezdrátově.



Obr. č. 6.1 Mikropočítač BeagleBone Black [25]

6.2 CAN Bus Cape

K mikropočítači BeagleBone je poskytována celá řada rozšiřujících modulů označovaných jako Cape. Jedním z nich je právě CAN Bus Cape sloužící pro komunikaci se sběrnicí CAN. Tento modul je zobrazen na obr. č. 6.2. Jedná se v podstatě pouze o CAN převodník, výstupní buffer a ROM paměť. Po připojení k mikropočítači je přečtena informace z této paměti a spustí se skript příslušející ke konkrétnímu rozšiřujícímu modulu, v tomto případě k modulu CAN. Tento skript se nazývá Device Tree Source a bude popsán v kapitole 6.5. Komunikace s tímto rozšiřujícím modulem je realizována pomocí UART sběrnice. Pro jednodušší komunikaci jsem vytvořil knihovnu `libcancom.so` obsahující API (Application Programming Interface) pro sběrnici CAN. Toto API bude detailně rozebráno v kapitole 7.3.



Obr. č. 6.2 CAN Bus Cape [26]

6.3 Bezdrátové rozhraní

Pro telemetrická zařízení používaná v závodech F1 se pro bezdrátový přenos dat využívá nejčastěji frekvence 1,5 GHz, případně jiná frekvence v závislosti na pravidlech konkrétního závodu [6]. Bohužel v České republice není možné používat tuto frekvenci bez patřičných povolení, proto jsem se rozhodl jako rozhraní pro bezdrátovou komunikaci telemetrie s klientským počítačem použít technologií Wi-Fi pracující na frekvenci 2,4 GHz. Tuto frekvenci je možné v České republice volně využívat i bez povolení. Jako Wi-Fi modul jsem nakonec použil modul firmy Logic Supply UWN200. Mezi ostatními testovanými moduly jsem zvolil právě tento modul, protože vzhledem k použití operačního systému Ångström určeného opravdu pouze pro malá vestavěná zařízení byl problém s neexistujícími ovladači pro bezdrátová rozhraní. Po úpravě ovladačů pro modul UWN200 se mi tyto ovladače podařilo nainstalovat a tím i modul zprovoznit. Tento modul navíc disponuje výstupem na externí anténu a vyniká svými malými rozměry a hmotností. V telemetrickém zařízení tento modul pracuje v režimu Infrastruktury, kdy se podle pevně nastavených pravidel ihned po spuštění připojí k přístupovému bodu (AP). Původní myšlenku, kdy mělo samotné telemetrické zařízení pracovat v režimu AP, jsem změnil, protože režim Infrastruktury nese výhodu v možnosti

využít více přístupových bodů rozmístěných po závodní trati a tím pokrýt celý sledovaný prostor. Na straně klientské aplikace je Wi-Fi rozhraní řešeno externím modulem AirLive WN-370USB. Tento modul se ke klientskému počítači připojuje přes rozhraní USB a nepotřebuje externí napájení. To je velmi výhodné, protože testování formule Dragon IV se často provádí v podmínkách, kde není k dispozici elektrická síť. Dalšími nespornými výhodami tohoto modulu je jeho výstup pro připojení externí antény a možnost pracovat v softwarovém AP módu, kdy je pomocí softwaru emulováno chování stejné jako běžného AP. Konfigurace modulu je zobrazena v tabulce 6.1.

Mód	AP
Název sítě	TU_Brno_Racing
Kanál	8 (2447)
Heslo sítě	zavodnici
Ověření v síti	WPA-PSK
Šifrování	AES
IP adresa AP	192.168.159.1
IP adresa serveru (Formule)	192.168.159.2

Tabulka 6.1Konfigurace Wi-Fi modulu

6.4 GPS Modul

Závodní okruhy jsou velmi rozmanité a ve většině případů není možné, aby formule byla po celou dobu své jízdy viditelná. Ve skutečnosti je většinou viditelná pouze velmi malou část tratě, kdy projíždí kolem boxů. Ale informace o aktuální poloze formule je pro nás bezesporu také velmi důležitá, proto je telemetrie doplněna GPS modulem, který nám tuto informaci může poskytnout. Z celé škály nabízených GPS modulů jsem vybral a následně použil modul firmy RF Solutions JUPITER-610F. Klíčové vlastnosti tohoto modulu jsou uvedeny v tabulce 6.2.

Počet kanálů	65
Rychlost sériového portu	4800 – 115 200 bps
Rychlost aktualizace	10 Hz
Přesnost	2,5m
Rychlost získání pozice	1s, 5s, 29s

Tabulka 6.2 Parametry GPS modulu [27]

Tento odstavec čerpá z [28]. Z vlastností modulu si můžeme povšimnout relativně krátké doby získání pozice. Tato doba závisí na tom, zda se jedná o tzv. „Hot start“, „Warm start“, nebo „Cold start“. Nejrychlejší získání pozice je v případě, kdy se jedná o „Warm start“. Toto nastane, pokud si GPS modul pamatuje od svého vypnutí svou polohu, polohu viditelných satelitů, UTC čas a zároveň pokud je GPS poloha po obětovném zapnutí stejná. V případě „Warm startu“ je situace podobná jako

v předchozím případě s tím rozdílem, že GPS modul si nepamatuje viditelné satelity. Doba získání aktuální polohy je v tomto případě vyšší, ale díky znalosti předchozí platné polohy má GPS modul informaci o tom, které satelity vyhledávat a tudíž je jejich vyhledání rychlejší. V posledním případě, v případě „Cold startu“, je doba získání aktuální polohy nejdelší, protože GPS modul nemá žádnou informaci, která by mu mohla pomoci urychlit vyhledání viditelných satelitů. Testováním tohoto GPS modulu jsem zjistil, že doba připojení k satelitům je opravdu velmi krátká a pohybuje se v řádech desítek sekund a to i v prostředích, kde GPS anténa nemá čistý výhled k satelitům. K takto rychlé době získání aktuální polohy zajisté pomáhá i použití aktivní antény a knoflíkové baterie na GPS modulu. Tato baterie zajišťuje napájení obvodu reálného času a paměti SRAM uchovávající informace o poslední známé platné poloze.

Komunikace s modulem probíhá přes UART sběrnici a pro její zefektivnění jsem vytvořil knihovnu `libserialcom.so`, která bude popsána v kapitole 7.3. Nastavení komunikace je zobrazeno v tabulce 6.3 a probíhá opět pomocí mechanismu Device Tree Overlay, který je detailně popsán v kapitole 6.5.

Rozhraní	UART – sériové, plně duplexní
Rychlost přenosu	115 200 (nastavitelné)
Start bit	1
Stop bit	1
Data bit	8
Parita	není
Tvar přenášených dat	ASCII NMEA0183 Ver.:3.01
Update Rate	10 (nastavitelné)
Formát výstupních zpráv	GLL

Tabulka 6.3 Konfigurace UART komunikace pro GPS modul

Rychlost komunikace společně s rychlostí obnovování dat je možné bezdrátově a za běhu měnit z klientské aplikace. Ostatní nastavení jsou pevně nastaveny pomocí binárních zpráv. Formát těchto zpráv je popsán v [29]. Po nastavení modulu binárními zprávami již probíhá komunikace pomocí protokolu NMEA0183. Jedná se o protokol vzniklý pro námořní plavidla definující komunikaci mezi jednotlivými zařízeními, jako jsou např. sonary, autopiloti, nebo právě GPS zařízení. Použitý modul je určen pro náročné podmínky v automobilovém a námořním průmyslu, proto i on využívá pro svou komunikaci protokolu NMEA0183. Komunikace tímto protokolem probíhá pomocí zasílání přesného sledu ASCII znaků. GPS modul je nakonfigurován tak, aby v intervalu nastavitelném z klientské aplikace neustále zasílal přes UART rozhraní zprávu typu GLL protokolu NMEA0183. Tato zpráva obsahuje informace o aktuální poloze, času a správnosti obdržených dat. Oproti jiným typům zpráv neobsahuje podrobnější informace například o stavu jednotlivých satelitů, nicméně tyto informace pro nás nejsou důležité a pouze by zpomalovaly přenos důležitých dat. Tvar této zprávy je definován níže v tabulkách č. 6.4 a č. 6.5. Celé znění standardu včetně tvaru ostatních zpráv je pak možné najít v [30].

$GPGLL,ddmm.mmmm,a,dddmm.mmmm,a,hhmmss.sss,A,a*hh<CR><LF>								
0	1	2	3	4	5	6	7	8

Tabulka 6.4 Tvar zprávy typu GLL podle standardu NMEA0183 [27]

Kde:

Číslo pole	Označení	Popis	Příklad
0	$GP GLL	- identifikátor odesílatele - typ zasílané zprávy	
1	Zeměpisná šířka	ddmm.mmmm	4250.5589 ⇔ 42°50,5589“
2	Sever/Jih	N = North (Sever) S = South (Jih)	
3	Zeměpisná délka	dddmm.mmmm	14718.5084 ⇔ 147°18,5084“
4	Východ/Západ	E = East (východ) W = West (západ)	
5	Čas UTC	hhmmss.sss	092204,999 ⇔ 9:22:4,999
6	Stav	A = data jsou platná V = data nejsou platná	
7	Aktuální mód	N = data nejsou platná A = autonomní mód D = diferenciální mód E = odhadový mód S = simulační mód	
8	Kontrolní součet		

Tabulka 6.5 Význam dat zprávy typu GLL [27]

Modul je vybaven konektorem MMCX pro připojení externí antény. K tomuto konektoru jsem přes redukci z konektoru MMCX na konektor SMA připojil externí anténu uBlox ANN-MS. Jedná se o aktivní anténu poskytující zesílení až 27dB a zajišťující tak spolehlivý signál i za zhoršených podmínek [31]. Připojení GPS modulu i napájení zesilovače externí antény jsem provedl navržením nového rozšiřujícího modulu, ke kterému je GPS modul společně s dalšími inerciálními senzory připojen. Tento rozšiřující modul je detailně popsán v kapitole 6.6.

6.5 Nastavení a popis HW - Device Tree Overlay

Distribuce Ångström používá pro popis hardwaru techniku zvanou „Device Tree Overlay“. Device Tree Overlay nám umožňuje definovat Device Tree Source konfigurační soubory (dále jen DTS). Tyto soubory jsou umístěny ve složce `/lib/firmware/` a obsahují informace o tom, jak přesně nastavit HW pro konkrétní činnost. Jedná se o nastavení jednotlivých pinů, jejich módu, pull-up i pull-down rezistorů a další nastavení nutná pro správnou činnost. Pro přehlednost zde neuvádím

obsah všech konfiguračních souborů. Všechny tyto soubory jsou umístěny v příloze této diplomové práce a kdykoliv čtenáři k dispozici. Device Tree Overlay jsem použil všude tam, kde jsem potřeboval komunikovat se vstupně/výstupními piny mikropočítače jak je zobrazeno v tabulce 6.6.

Číslo I/O pinů	Název rozhraní	Funkce	Název DTS souboru
P9.26, P9.24	UART1	CAN Bus Cape	BB-BONE-SERL-01-00A2.dts
P9.11, P9.13	UART4	GPS modul	BB-UART4-00A0.dts
P8.10, P8.11, P8.12, P8.13, P8.14, P8.15, P8.16, P8.17	I2C1, další piny	Inerciální senzory	TELEMETRY-IO-00A0.dts

Tabulka 6.6 Soubory pro popis HW – Device Tree Overlay

Po vytvoření konfiguračního DTS souboru a jeho nahrání do složky `/lib/firmware/` je nutná jeho kompilace, která vytvoří soubor DTBO. Způsob kompilace je zobrazen níže:

```
dtc -O dtb -@ /lib/firmware/BB-BONE-SERL-01-00A2.dts >
/lib/firmware/BB-BONE-SERL-01-00A2.dtbo
```

Nyní již stačí aktivovat konfigurační soubor. Například pro aktivaci CAN Bus Cape to je:

```
echo BB-BONE-SERL-01-00A2 >
sudo tee  sys/devices/bone_capemgr.*/slots
```

Při každém dalším spuštění mikropočítače se již konfigurační soubory automaticky nahrají a provede se nastavení hardwaru.

6.6 Rozšiřující modul s inerciálními senzory

Popisovaná telemetrie pracuje také s akcelerometrem a gyroskopem, které budou dále označovány souhrnným názvem inerciální senzory. Pro propojení těchto inerciálních senzorů a GPS modulu s mikropočítačem jsem navrhl a vytvořil rozšiřující modul, který toto spojení zajišťuje. Schéma zapojení tohoto modulu vytvořené v programu NI Multisim je možné shlédnout v Příloha č. 2. Příloha č. 3 pak zobrazuje vzhled hotového modulu. Rozšiřující modul obsahuje dva základní bloky. Prvním blokem je napájecí část, která zajišťuje napájení mikropočítače. Druhá část pak zajišťuje komunikaci s inerciálními senzory a GPS modulem a stejně tak i poskytuje napájení externí aktivní anténě pro tento modul. Následující kapitoly podrobně rozebírají oba bloky.

6.6.1 Napájecí blok

Jedná se o relativně jednoduchý blok zajišťující napájení mikropočítače přesným napětím 5 V z palubní 12V sítě formule. Hlavním prvkem je zde stabilizátor napětí na 5 V. Po zhlédnutí tabulky 6.7 zobrazující maximální proudy odebírané jednotlivými bloky celé telemetrie je zřejmé, že napájecí blok musí poskytovat proud alespoň 1,6 A. Proto jsem se rozhodl použít stabilizátor L78S05CV, který stabilizuje vstupní napětí v rozmezí 10 – 35 V na výstupní napětí 5 V a dokáže pracovat s proudem až 2 A [32].

Blok	Maximální proudový odběr [mA]
Mikropočítač BeagleBone Black	1000
Wi-Fi modul	500
GPS modul	70
Aktivní anténa GPS modulu	13
Gyroskop	6,1
Akcelerometr	0,11
Celkem	1589,21

Tabulka 6.7 Proudové odběry jednotlivých bloků [32][27][31][33][34]

Napájecí obvod je navíc na vstupu doplněn o diodu zabraňující poškození obvodu přepólováním vstupu. Další dioda je v závěrném směru zapojena mezi vstupem a výstupem. Při zkratu na vstupních svorkách je vstupní napětí obvodu nulové, ale výstupní napětí je díky parazitní kapacitě nenulové a tedy vyšší než vstupní napětí. Tato situace by nám poškodila stabilizátor, proto jsem zapojení osadil další diodou zapojenou v závěrném směru mezi vstupem a výstupem stabilizátoru. V případě vyššího výstupního napětí než vstupního je dioda otevřena a napětí na výstupu je okamžitě sníženo na hodnotu vstupního napětí a nedojde tak k proražení stabilizátoru.

6.6.2 Komunikační blok

Druhý blok se dá rozdělit na další části a to konkrétně na část akcelerometru, část gyroskopu a část GPS modulu.
Nejjednodušší částí je část GPS modulu, která se skládá pouze z napájecích a zemnících vodičů zajišťujících napájení GPS modulu a zesilovače signálu antény a z propojení modulu s UART sběrnicí mikropočítače. Samotný modul byl detailně popsán v kapitole 6.4.

Tento odstavec čerpá z [33]. Pro měření úhlového zrychlení jsem použil gyroskopický senzor L3G4200D. Jedná se o MEMS senzor měřící zrychlení ve všech 3 osách a umožňující měřit úhlové

zrychlení už od 8,75 mdps tedy od 0,00875 stupňů za sekundu. Toto vysoké rozlišení nám nemusí vyhovovat pro všechny aplikace a provozní podmínky, proto senzor umožňuje toto rozlišení a tím i rozsah měnit. Konkrétně je možné nastavit rozsah od 250 dps do 2000 dps. Samotnou změnu je pak možné provést přímo z klientského počítače pomocí vyvinuté aplikace. Stejně tak je možné z této aplikace nastavit interval aktualizace hodnoty úhlového zrychlení. Výsledná 16bitová hodnota je přenášena přes I^2C sběrnici sdílenou společně s akcelerometrem do mikropočítače. Pro eliminaci rychlých a nežádoucích zákmitů, které gyroskop detekuje, jsem obvod doplnil o externí low-pass filtr pro obvod fázového závěsu. Tento filtr dokáže tyto nechtěné skokové impulzy společně s dalším interním high-pass a low-pass filtrem vyfiltrovat a tím zpřesnit celé měření. Tento gyroskop je svou malou hmotností, rychlostí i přesností ještě zvýšenou interním sledováním teploty čipu přímo určen pro aplikace tohoto typu.

Tento odstavec čerpá z [33]. Jako akcelerometr jsem použil opět senzor MEMS z dílny firmy ST Microelectronics, tentokrát s označením LSM303DLHC. Tento senzor kromě akcelerometru obsahuje i magnetometr, který se dá použít jako elektronický kompas. Oba tyto obvody jsou přístupné přes svoji adresu na I^2C sběrnici. Jak gyroskop, tak i akcelerometr s kompasem jsou připojeny na stejnou I^2C sběrnici. Komunikace s nimi pak probíhá pomocí jejich adresy. Tyto adresy jsou zobrazeny v tabulce 6.8. Oba tyto obvody měří ve všech 3 osách a naměřenou 16bitovou hodnotu ukládají do výstupních registrů, z kterých jsou pak přes I^2C sběrnici čteny.

Zařízení	Adresa
Gyroskop	0x68
Akcelerometr	0x19
Kompas	0x1E

Tabulka 6.8 Zařízení na I^2C sběrnici a jejich adresy [33][34]

Opět je možné nastavit všechny důležité parametry z klientské aplikace. Jedná se především o rozsah měření, který u akcelerometru může být od 2 G až do 16 G a u kompasu pak od 1,3 gauss až do 8,1 gauss. Hodnoty kompasu pro nás nejsou až tak důležité, ale důležité je měřit lineární zrychlení akcelerometrem. Podle něj můžeme určit intenzitu brždění, prudkost zatáčení, nebo sílu zrychlení. V předchozí verzi formule Dragon bylo naměřeno lineární zrychlení až 2 G, proto akcelerometr bude nejčastěji nastaven na rozsah 2 G, nebo 4 G. Podobně jako gyroskop tak i tento senzor obsahuje vestavěný obvod pro měření teploty čipu, podle které upravuje naměřené hodnoty. V tomto případě, ale teploměr není nakalibrován, proto můžeme měřit pouze zda teplota klesá či stoupá a o kolik. Přesná hodnota teploty ve stupních Celsia nám není známa a ani ji nepotřebujeme znát, protože je to teplota čipu, která slouží pouze pro automatickou kalibraci výstupních hodnot senzoru.

Tento odstavec čerpá z [35]. Jak obvod gyroskopu, tak obvod akcelerometru je navíc doplněn o tzv. blokovací kondenzátory. Tyto kondenzátory omezují vliv parazitních indukčností na napájecích vodičích elektronických součástek, proto musí být umístěny co nejblíže k těmto součástkám. Tyto integrované obvody mají velkou impulzní spotřebu, proto pro předejití vzniku úbytku napětí na napájecích vodičích jsou zde použity tyto kondenzátory, které nám slouží jako zdroj napětí a tyto úbytky napětí vyrovnávají. Pokud bychom tyto kondenzátory nepoužily, přívodní napájecí vodiče by

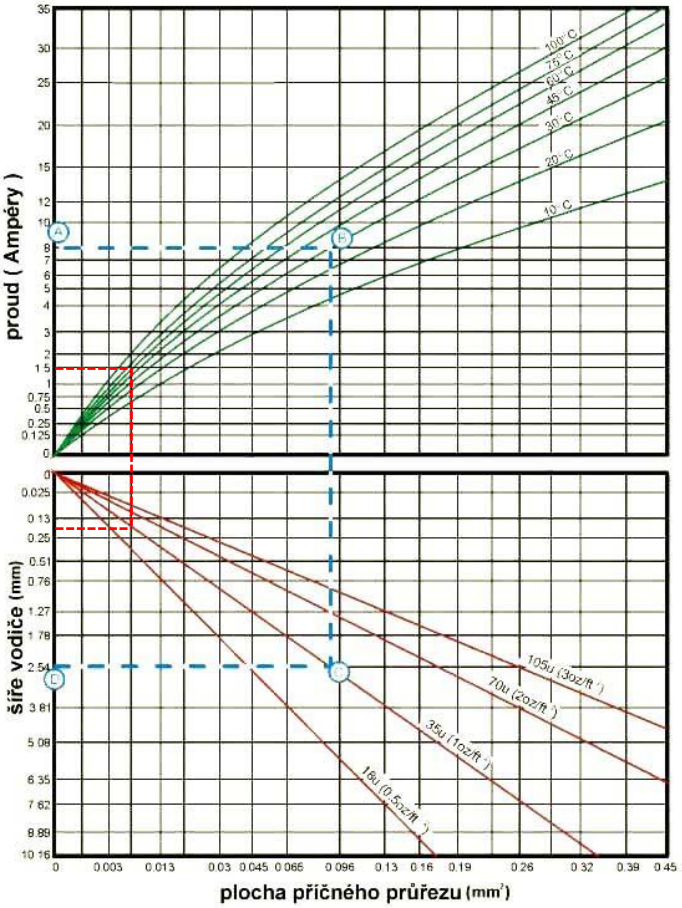
se staly zdrojem rušení a způsobovali by hazardní stavy a rozkmitání obvodu. Každý kondenzátor má kromě své kapacity i parazitní indukčnost a odpor. Se stoupající kapacitou stoupají i tyto parazitní složky, proto pro dokonalé pokrytí celého frekvenčního spektra je vhodné použít kondenzátorů více. Po prostudování odborné literatury [35] a technické dokumentace senzorů jsem se rozhodl použít pro každý senzor dva blokovací kondenzátory o hodnotě 10 uF a 100 nF. Tyto kondenzátory by měly pokrýt celou používanou část frekvenčního spektra a zaručit tak spolehlivou funkčnost senzorů, což bylo prakticky otestováno a potvrzeno.

Pro napájení celého tohoto rozšiřujícího modulu a tím i jednotlivých komponent umístěných na tomto modulu je použito napájecí napětí 3,3 V poskytované mikropočítačem. Stejně tak je mikropočítač použit pro nastavení pull-up rezistorů sběrnice I^2C, které jsou programově nastaveny viz kapitola 6.5. V následující kapitole bude popsán fyzický návrh a realizace rozšiřujícího modulu.

6.6.3 Návrh a realizace DPS rozšiřujícího modulu

Rozšiřující modul byl navrhován s ohledem na vhodné připojení k mikropočítači BeagleBone Black i s ohledem na kompatibilitu s použitím tohoto modulu a mikropočítače BeagleBone v jiných aplikacích. K návrhu DPS jsem použil program NI Ultiboard. Deska plošného spoje tohoto modulu je zobrazena v příloze č. 4 a její tvar vychází z předepsaného tvaru pro rozšiřující moduly mikropočítače BeagleBone. Tím je dosaženo univerzálnosti použití tohoto modulu. Modul je k mikropočítači případně k jinému rozšiřujícímu modulu připojen pomocí dvojice propojovacích konektorů a každý propojovací konektor se ještě skládá z dvou řad oboustranných kolíků. S ohledem na výsledné rozměry a také hmotnost byly pro tento modul použity pouze oboustranné kolíky a již nebyla použita dutinková lišta pro připojení dalšího rozšiřujícího modulu. Proto tento rozšiřující modul musí být vždy poslední z řady rozšiřující modulů.

Při návrhu plošného spoje jsem kladl důraz na co nejkvalitnější návrh. Po vytvoření obrysu plošného spoje a montážních otvorů, které jsou přesně dané pravidly tvorby rozšiřujících obvodů mikropočítače BeagleBone, jsem začal s rozmísťováním jednotlivých součástek. Nejprve jsem začal s komponentami, jejichž poloha je pevně daná. To znamená s oboustrannými kolíky, které musí přesně pasovat do dutinkových lišt mikropočítače a umístěním stabilizátoru napětí. Ten musí být umístěn v místech, kde je plošný spoj vyříznutý, aby jej neohříval, a zároveň musí mít dostatečné místo i pro umístění svého chladiče. Další komponentou byl GPS modul, jehož poloha byla také jednoznačná. Modul totiž obsahuje konektor na externí anténu, která musí být kvůli rušení co nejdál od napájecího obvodu a zároveň na okraji desky plošného spoje. Zbylá část plošného spoje byla rozdělena na část pro napájecí obvod a část pro inerciální senzory. Tyto dvě části jsou od sebe odděleny „vylitou“ vodící plochou připojenou k záporné polaritě napětí. Napájecí obvod bude pracovat s proudy maximálně 2A jak bylo vysvětleno v kapitole 6.6.1 a plošný spoj obsahuje měděnou fólii o tloušťce 0,35um. Uvažujme tedy reálný stálý proudový odběr, který se může pohybovat okolo 1,5A a maximálním povolené ohřátí vodivé cesty o 40°C. Poté z tabulky dodané výrobce plošného spoje a zobrazené na obr. č. 6.3 vychází minimální šířka vodivé cesty 0,2 mm [37]. Proto zvolená šířka vodivých cest 10 milů, tedy 0,254 mm je dostačující i pro napájecí větev, ve které tečou největší proudy, i v případě proudových špiček.



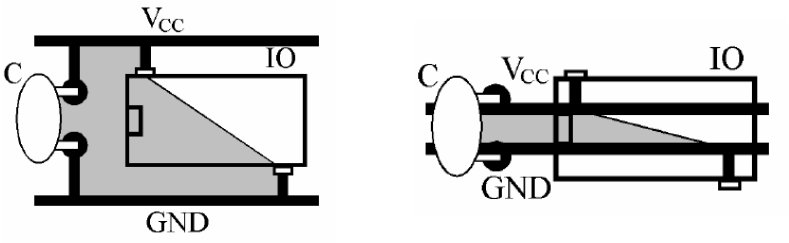
Obr. č. 6.3 Volba šířky cesty v závislosti na proudu, ohřátí a tloušťce cesty [37]

Maximální napěťové zatížení v tomto případě není potřeba řešit, protože pracujeme s napětím maximálně 14 V. Technologie výroby nedovoluje vytvořit mezery mezi vodivými cestami v takové velikosti, aby se pro toto nízké napětí muselo maximální napěťové zatížení hlídat.

Složitější situace nastává v případě elektromagnetické kompatibility (EMC). V mezinárodním elektrotechnickém slovníku [36] je pojem elektromagnetická kompatibilita definován jako schopnost zařízení nebo systému fungovat vyhovujícím způsobem ve svém elektromagnetickém prostředí bez vytváření nepřípustného elektromagnetického rušení čehokoliv v tomto prostředí. Elektromagnetické rušení je zde pak definováno jako jakýkoliv elektromagnetický jev, který může zhoršit provoz přístroje, zařízení, nebo systému, anebo nepříznivě ovlivnit živou nebo neživou hmotu. Každý elektrický obvod, kterým protéká elektrický proud, nebo je zdrojem elektrického napětí, je zároveň i zdrojem rušení. Cílem bylo navrhnout plošný spoj tak, aby toto vyzařování bylo co nejmenší. Zároveň bude vytvořené zařízení v prostředí s velmi vysokým rušením zapalováním motoru, proto je nutné navrhnout obvod i s přihlédnutím k těmto faktorům.

Elektromagnetické vyzařování vzniká v každé proudové smyčce, přímém vodiči i vysíláním periodického signálu na vodiči. Návrh obvodu tedy musíme provádět s ohledem na všechny tyto aspekty. Minimalizaci vyzařování v přímém vodiči provedeme minimalizací proudu tekoucího tímto vodičem zvolením správných součástek. Vyzařování vzniklé vysíláním periodického signálu na

vodiči můžeme snížit nepoužíváním příliš rychlých obvodů a snížením frekvence signálu na vodiči. Nejčastější a nejvyšší míra vyzařování avšak vzniká v proudových smyčkách. Proto jsem nejvyšší část návrhu plošného spoje věnoval právě této problematice. V závislosti na velikosti protékajícího proudu v každé smyčce vzniká vyzařování, kterému nemůžeme nijak zamezit, ale můžeme jej alespoň snížit až na zanedbatelnou hodnotu. Toho docílíme především zkrácením těchto smyček. Čím je smyčka kratší, tím je i vyzařované rušení nižší. Z toho důvodu někteří výrobci elektronických součástek umisťují napájecí vývody naproti sobě a ne do uhlopříčky jak tomu bylo dříve. Obr. č. 6.4 převzatý z [35] zobrazuje správné a špatné zapojení napájecích vývodů integrovaného obvodu. Pokud srovnáme oba obrázky vidíme, že v druhém případě je plocha proudové smyčky podstatně nižší než v prvním případě. Stejně tak nám ke snížení vyzařování můžou pomoci i rozlévané vodivé plochy, blokovací kondenzátory popsané v kapitole 6.6.2, správné rozmístění součástek i vedení napájecích vodičů. Odolnost zařízení proti vnějšímu rušení můžeme zvýšit použitím filtračních kondenzátorů, rozlitím vodivých ploch připojených k zápornému napětí, nebo stíněním.



Obr. č. 6.4 Velikost proudové smyčky při špatném a správném vedení vodičů [35]

Vyzařování způsobené periodickým signálem a vysoké frekvenci se dá eliminovat umístěním zemnícího vodiče do těsné blízkosti. V případech kdy není možné do těsné blízkosti tento vodič umístit, je možné tuto situaci vyřešit rozlitím vodivé vrstvy připojené na zemnící vodič do sousední vrstvy. V zařízení jsem použil dvouvrstvou desku plošného spoje, proto jsem této možnosti s výhodou využil. Vodiče sběrnic jsem v návrhu oddělil zemnícím vodičem ze všech stran včetně rozlití vodivé plochy do druhé vrstvy. Tím jsem předešel rušení přenosu dat o vysoké frekvenci.

Důležité bylo také neopomenout vyzařování plošného spoje do boku. Tomu bylo zabráněno rozlitím vodivé zemnící plochy po celém okraji plošného spoje po obou vrstvách. Opět kvůli předejití velkým proudovým smyčkám, jsem tyto plochy vzájemně propojil prokovem na více místech. Volná místa plošného spoje jsem také vyplnil vodivými zemnícími plochami a to z důvodu co největšího odrušení zařízení ale také kvůli šetření frézovacího nástroje při výrobě, který by v opačném případě musel frézovat celou volnou plochu.

Vzhledem k použití zařízení v podmínkách náročných na vibrace byly, až stabilizátor napětí napájecího obvodu a již zmíněné oboustranné kolíky, použity SMD součástky. Použití SMD součástek také významnou mírou snižuje velikost vyzařování, protože se zmenšením součástek se zmenší i velikost proudových smyček.

Po návrhu plošného spoje byla výroba desky plošného spoje zadána externí firmě. Po obdržení vyrobeného plošného spoje jsem pokračoval s vývojem ve školní laboratoři. Po nanesení pájecí pasty

jsem pomocí profesionální osazovací stanice ESSEMTEC EXPERT-FP osadil plošný spoj SMD součástkami. K tomuto úkonu byla nutná precizní práce, protože bylo nutné osadit i inerciální senzory s pouzdrem LGA-14 a LGA-16, které na pouhých 16mm2 obsahují 14 resp. 16 vývodů. Po úspěšném osazení součástek do pájecí pasty jsem provedl pájení přetavením. K tomu jsem také využil školní laboratoř a přetavovací pec SMD-2007 s přetavením řízeným mikroprocesorem. Toto řízení teploty pece mikroprocesorem bylo velmi důležité, protože jak akcelerometr, tak i gyroskop jsou součástky s velkou citlivostí na vlhkost, přesněji jejich MSL = 3. Toto označení znamená, že pokud je senzor na více jak 7 dní vystaven vzdušné vlhkosti (30°C, 60%) je riziko, že při pájení přetavením dojde k tzv. popkorn efektu. Při tomto efektu se vlivem působení tepla zplynuje nasátá vlhkost senzoru a při dalším zahřívání dojde k rozpínání plynů a prasknutí senzoru. Tomuto nenávratnému poškození senzoru se dá předejít opětovným vysušením senzoru před samotným pájení, nebo šetrném zahřívání při pájení přetavením.

6.6.4 Fyzická zástavba zařízení

Vytvořené telemetrické zařízení obsahuje citlivou elektroniku a je velmi náchylné na vibrace, vlhkost a další nepříznivé vlivy. Proto byla pomocí CAD nástrojů navrhnuta krabička pro toto zařízení. Její výroba pak probíhala metodou 3D tisku. Pro potlačení vibrací je zařízení v krabičce uchyceno na pružných článcích, které eliminují jejich velkou část. Stejně tak krabička poskytuje ochranu proti vlhkosti i mechanickému poškození elektroniky a zajišťuje pevné uchycení k rámu vozu. Její výsledná podoba je zobrazena v příloze č. 5.

7 Firmware

Každý vestavěný systém potřebuje řídící program, který se nazývá firmware. Slovo firmware bylo původně vytvořeno pro odlišení od vysokoúrovňového softwaru. Jedná se o software, který je určen pro konkrétní hardware. Norma IEEE [38] jej pak definuje jako kombinaci hardwarového zařízení a počítačových instrukcí a dat, které jsou v zařízení umístěny jako software jen pro čtení. Popisované telemetrické zařízení obsahuje firmware na více úrovních. První úrovní je samotný operační systém Ångström, který zajišťuje funkci samotného mikropočítače. Další úrovní je pak úroveň samotného kódu, který již provádí konkrétní požadované operace a dělá tak z univerzálního mikropočítače, úzce zaměřené zařízení provádějící pouze funkci telemetrie. Vrstva operačního systému je univerzální, vhodná pro nejrůznější aplikace, proto se v tomto textu budu zabývat pouze vrstvou řídícího kódu.

Řídící kód se skládá především z kódu napsaného v jazyce C# a zkompilovaného pomocí kompilátoru MONO. Pro programování na nižší úrovni nutného pro komunikaci s hardwarem jsem vytvořil knihovny v jazyce C++. Posledním typem souborů, které řídí mikropočítač jsou nejrůznější konfigurační soubory jako například soubory DTS (viz kapitola 6.5), soubory pro nastavení sítě, nebo skripty pro automatickou konfiguraci systému a spouštění řídícího kódu. Všechny tyto typy souborů budou rozebrány v následujících kapitolách. Pro přehlednost zde nebudu uvádět kompletní zdrojový kód ani přesný význam parametrů všech metod. Tyto informace je možné najít na přiloženém CD přímo ve zdrojových souborech.

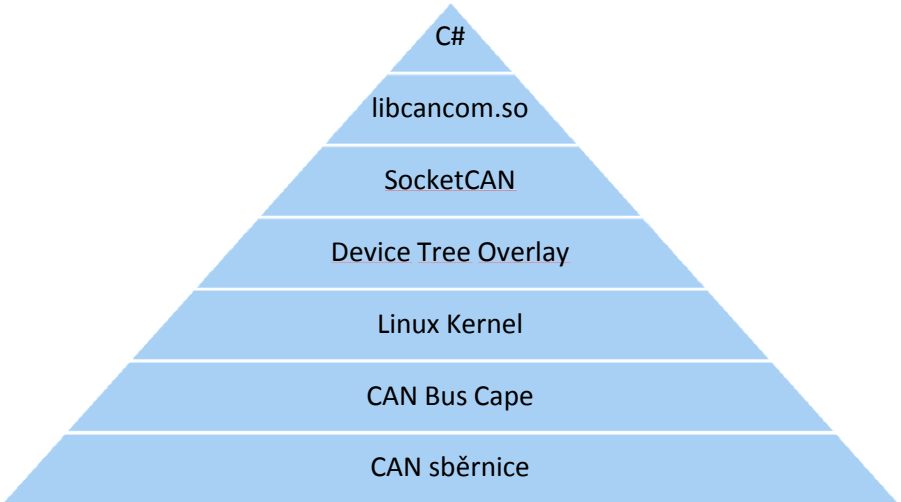
7.1 Knihovna pro komunikaci s rozhraním CAN

Pro komunikaci s HW nelze vhodně využít programovací jazyky na vyšší úrovni jako je například jazyk C#, ale je nutné použít jazyk na nižší úrovni. K tomu je vhodný programovací jazyk C++, proto i knihovny pro komunikaci s hardwarem jsem napsal v tomto jazyku. První z popisovaných knihoven je knihovna pro komunikaci s rozhraním CAN – `libcancom.so` zkompilovaná ze souboru `libcancom.cpp`. Pro fyzickou komunikaci s CAN sběrnicí je použit modul CAN Bus Cape viz kapitola 6.2. Komunikace mezi tímto modulem a mikropočítačem je nakonfigurována pomocí souborů Device Tree Overlay popsaných v kapitole 6.5. Proto tato knihovna už jen poskytuje vhodné API pro hlavní řídící kód napsaný v jazyce C#. Ke komunikaci využívá knihovny `SocketCAN` a obsahuje metody zobrazené v tabulce 7.1. Tabulka obsahuje pouze základní popis funkce jednotlivých funkcí. Kompletní přehled včetně významu a použití jednotlivých proměnných je uveden ve zdrojovém souboru samotné knihovny.

Metoda / Popis
`int CanConnect(int *sck, char ifname[])` **Vytvoří připojení k soketu**
`void CanClose(int sck)` **Ukončí připojení k soketu**
`void CanShutdown(int sck, int how)` **Zruší příjem dat**
`int CanSend(int sck, unsigned int id, unsigned char data[8], unsigned char DLC)` **Odešle zprávu otevřenému soketu**
`int CanSendTo(unsigned int id, unsigned char data[8], unsigned char DLC, char ifname[])` **Vytvoří připojení k soketu a odešle na něj zprávu**
`int CanReceive(int sck, unsigned char* data, unsigned int* id)` **Přijme zprávu z otevřeného soketu**
`int CanReceiveFrom(char ifname[], unsigned char* data, unsigned int* id)` **Vytvoří připojení k soketu a přijme z něj zprávu**

Tabulka 7.1 Funkce knihovny libcancom.so

V kapitole 5 bylo rozebráno, že v současné době existuje více standardů pro komunikaci na sběrnici CAN. Jejich hlavní rozdíl je v délce dat hlavičky. Existují totiž i tzv. rozšířené rámce, kdy délka hlavičky není 11 bitů, jako je tomu u standardních rámců, ale 29 bitů. Vytvořená knihovna dokáže pracovat s oběma typy těchto zpráv. Po zavolání metody `CanSend`, případně `CanSendTo` pro odeslání zprávy, metoda podle identifikátoru pozná, zda se jedná o rozšířený, nebo standardní typ zprávy a podle toho upraví výslednou podobu zprávy. Tuto zprávu dále předává nižší programové vrstvě k odeslání na sběrnici. Stejná situace platí i u metod `CanReceive` a `CanReceiveFrom` pro příjem zpráv. Metody automaticky rozeznají typ přijaté zprávy, přečtou z nich data a ty již v dekódované a čisté podobě parametry předávají vyšší programovací vrstvě. Obr. č. 7.1 názorně zobrazuje vrstvy příjmu CAN zpráv.



Obr. č. 7.1 Vrstvy komunikace s CAN sběrnicí

7.2 Knihovna pro komunikaci s rozhraním I²C

Pro komunikaci s I²C sběrnicí slouží knihovna `libi2c.so` vzniklá zkompilování souboru `libi2c.cpp`. Tato knihovna poskytuje API pro komunikaci se zařízeními na I²C sběrnici. Akcelerometr, gyroskop i kompas, tedy všechny inerciální senzory, komunikují s mikropočítačem přes jednu stejnou I²C sběrnici. Knihovna zaštiťuje komunikaci se všemi těmito senzory. Stejně jako v případě knihovny `libcancom.so` pro komunikaci s CAN sběrnicí jsem i tuto knihovna napsal v jazyce C++. Po nastavení hardwaru opět pomocí Device Tree Overlay popsaného v kapitole 6.5, je sběrnice systémem zpřístupněna a dá se k ní přistupovat jako k souboru umístěném v `/dev/i2c-2`. Knihovna opět nabízí množství užitečných funkcí. Jejich hlavičky jsou zobrazeny v tabulce 7.2. Podrobná dokumentace je opět uvedena přímo v zdrojovém textu knihovny přiložené k této práci na CD nosiči.

Metoda / Popis	
`int I2COpen(char *name)`	
	Vytvoří připojení k I²C sběrnici
`void I2CClose(int fd)`	
	Ukončí připojení k I²C sběrnici
`int I2CSetDeviceAddress(int fd, int deviceAddress)`	
	Nastaví aktivní zařízení sběrnice
`short TwoComp(unsigned char MSB, unsigned char LSB)`	
	Dékoduje přijatá data uložená v dvojkovém doplňku
`void I2CDump(int fd, int bytes, int time)`	
	Opakovaně vypisuje hodnoty registrů aktivního zařízení
`int I2CReadBytes(int fd, int regAddr, unsigned char* buffer, int bytes)`	
	Přečte zadaný počet Bytů z aktivního zařízení
`int I2CWriteByte(int fd, unsigned char* buffer, char length)`	
	Zapíše hodnotu do registru aktivního zařízení
`int GyroInit(int fd)`	
	Inicializuje gyroskop
`int I2CSetAddressToGyro(int fd)`	
	Nastaví gyroskop jako aktivní zařízení na sběrnici
`int GyroReadAngularRate(int fd, int &x, int &y, int &z)`	
	Přečte hodnotu úhlového zrychlení
`int GyroReadTemp(int fd, unsigned char &temp)`	
	Přečte teplotu gyroskopu
`int AccelInit(int fd)`	
	Inicializuje akcelerometr
`int I2CSetAddressToAccel(int fd)`	
	Nastaví akcelerometr jako aktivní zařízení na sběrnici
`int AccelReadAcceleration(int fd, int &x, int &y, int &z)`	
	Přečte hodnotu lineárního zrychlení
`int CompassInit(int fd)`	
	Inicializuje kompas

`int I2CSetAddressToCompass(int fd)`	
	Nastaví kompas jako aktivní zařízení na sběrnici
`int CompassReadMagField(int fd, int &x, int &y, int &z)`	
	Přečte hodnotu magnetického pole z kompasu

Tabulka 7.2 Metody knihovny libi2c.so

Metody uvedené výše dekódují data do jejich správné podoby a příslušnými parametry metod je předávají vyšším funkcím. Tyto metody jsou potřeba, protože senzory poskytují data zakódovaná a například pro kompas data nejsou ani ve správném pořadí. Samotná serverová aplikace v jazyce C v jazyce C# pak nemusí řešit dekódování dat, ale pouze čte již nachystaná data od těchto metod. Inicializační metody nastavují senzory pro jejich správnou funkci. Jedná se především o jejich spuštění, nastavení rychlosti aktualizace dat, povolení filtrů a další. Knihovna také nabízí metody pro přímý přístup k registrům senzorů. Z těchto registrů je možné číst i zapisovat do nich hodnoty bezdrátově z klientské aplikace. Je ale velmi důležité provádět hlavně zápisy do těchto registrů s nejvyšší opatrností. Zápis dat do špatných registrů by mohl vést až k poškození samotného senzoru. Metoda `I2CReadBytes` umožňuje přečíst více registrů v jedné dávce. Pokud je její parametr udávající počet čtených registrů nastaven na hodnotu vyšší než 1, je komunikace na sběrnici nastavena na přenos v dávkách a požadovaný počet registrů je přenesen v jedné dávce. Toho využívám například při čtení naměřených hodnot registrů pro osu X, Y a Z. Registry těchto os jsou ihned za sebou a lze tedy jejich hodnoty přenést zároveň pouze jednou transakcí.

7.3 Knihovna pro komunikaci přes sběrnici UART

Poslední knihovnou pro komunikaci s HW je knihovna `libserialcom.so` resp. `libserialcom.cpp`. Tato knihovna slouží pro komunikaci přes sériovou UART sběrnici, kterou používám pouze pro komunikaci s GPS modulem. Pro správnou funkci knihovny je opět nutné správně nastavit hardware pomocí techniky Device Tree Overlay popsané v kapitole 6.5. Jedna z klíčových vlastností operačního systému Linux, která říká, že vše v systému lze chápat jako soubor, nám v tomto případě komunikaci se sběrnicí velmi ulehčuje [39]. Po správném nastavení hardwaru je vytvořen soubor `/dev/ttyO4` (pracuji se sběrnicí UART4). K sběrnici UART poté knihovna přistupuje stejně jako by se jednalo o tento soubor. V tabulce 7.3 jsou uvedeny metody této knihovny.

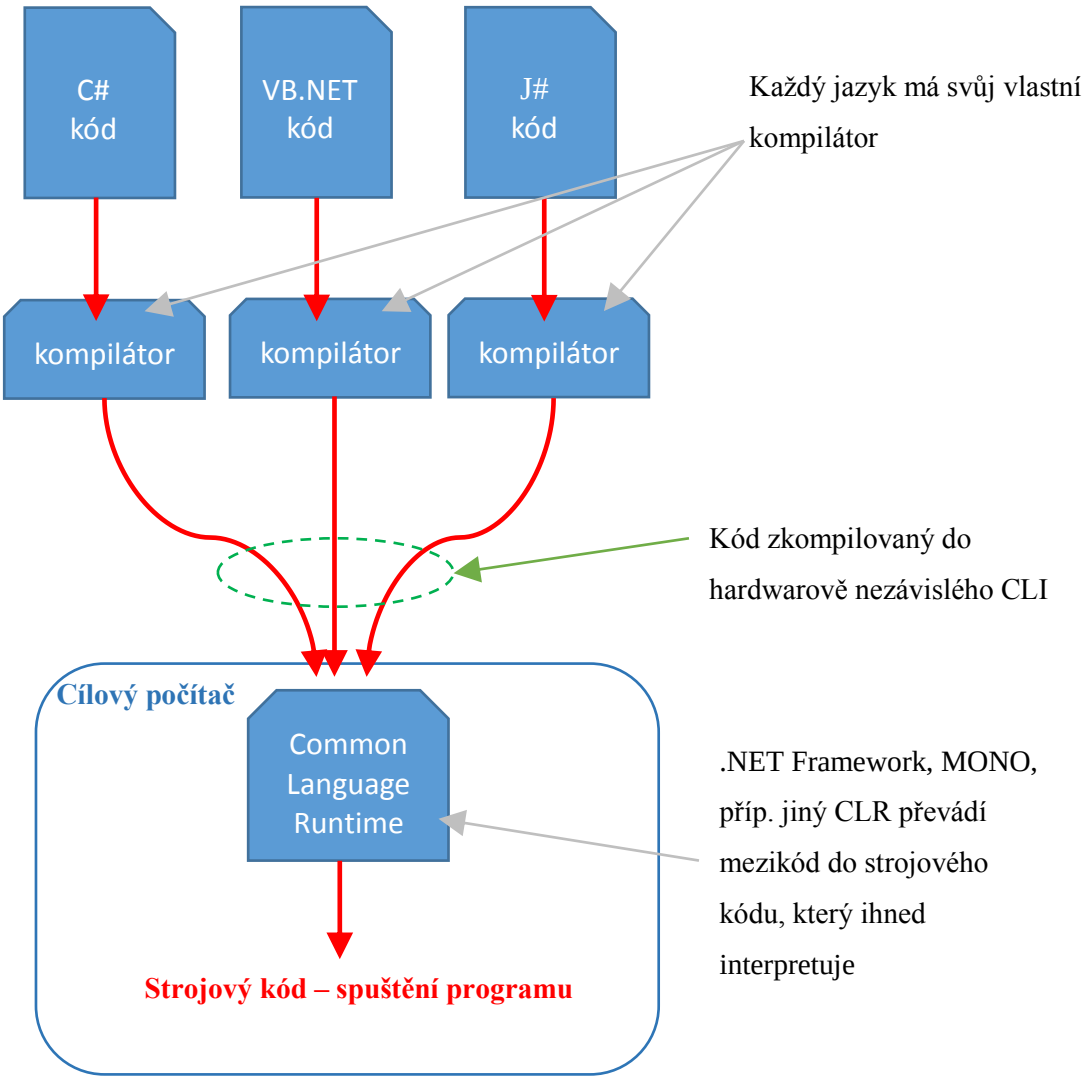
Metoda / Popis	
`int SerialConnect(char *portname, int baudRate)`	
	Vytvoří připojení k UART sběrnici
`int SerialRead(int fd, char *buf, int bufsize)`	
	Čte data z vytvořeného spojení s UART sběrnicí
`int SerialWrite(int fd, char *buf, int length)`	
	Zapisuje data do vytvořeného spojení s UART sběrnicí

Tabulka 7.3 Metody knihovny libserialcom.so

Jak už sám počet metod napovídá, jedná se o nejmenší z vytvořených C++ knihoven. Obsahuje pouze metody pro vytvoření spojení a čtení/zápis. Nejdůležitější metodou je zde metoda `SerialConnect` sloužící pro vytvoření připojení k UART sběrnici. Tato metoda provádí i počáteční inicializaci sběrnice, kdy je nutné nastavit všechny parametry přenosu. Mezi tyto parametry patří především rychlost přenosu dat po sběrnici, nastavení délky dat, parity a další komunikační nastavení. Stejně tak je důležité nastavit vlastnosti přenášených dat, jako je jejich formát, ukončovací znaky a podobně. Všechny tyto nastavení jsou předávány knihovně `termios.h`, která je součástí jádra operačního systému a zajišťuje komunikaci se sběrnicí na ještě nižší úrovni než knihovna `libserialcom.so`.

7.4 Řídící program serveru

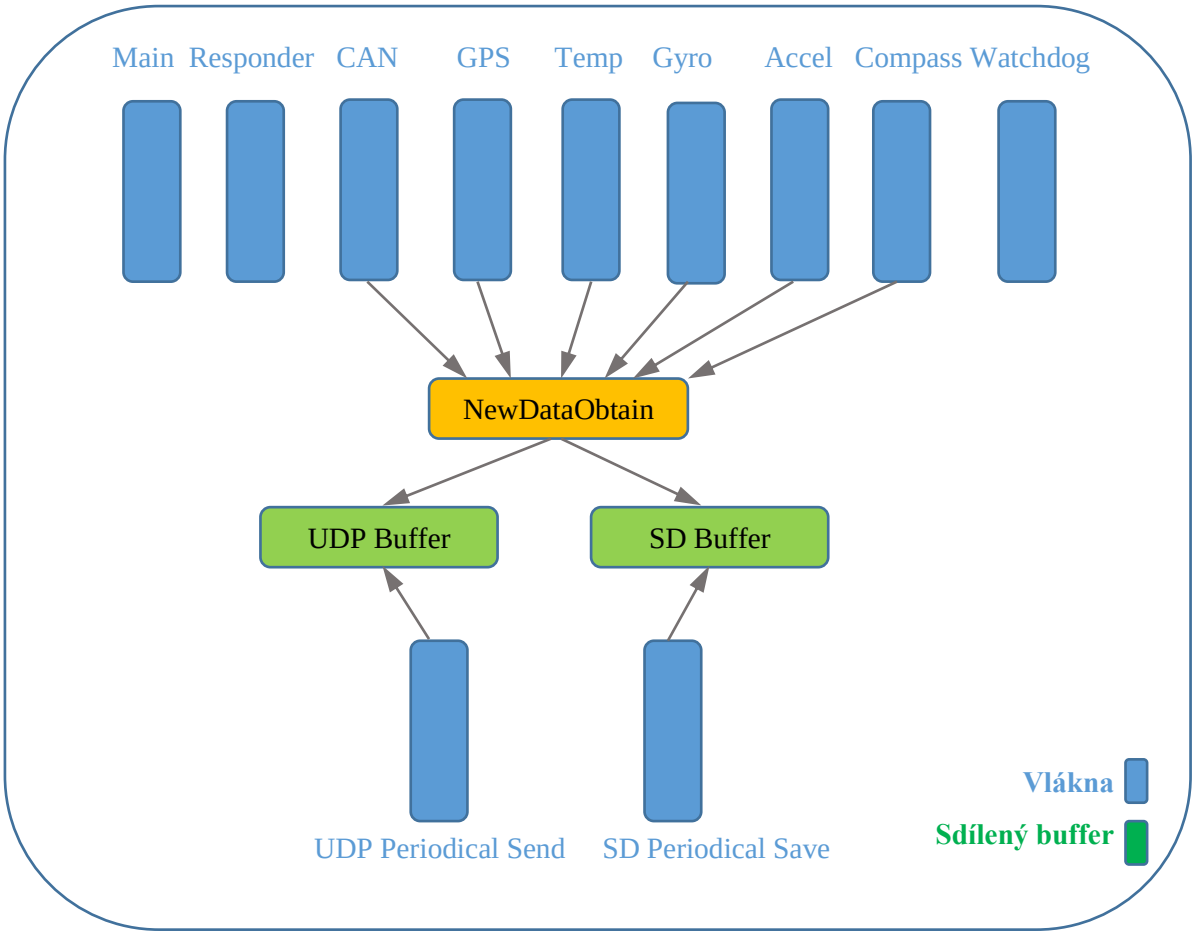
Nyní se dostáváme k hlavní části celého mikropočítače a tou je řídící program. Tento program jsem napsal v jazyce C# a na platformě .NET 4.0. Jazyk C# je jazyk vyvinutý firmou Microsoft pro operační systémy Windows. Tento jazyk není kompilován přímo do strojového kódu specifického pro konkrétní platformu, ale využívá tzv. CLI, neboli Common Language Interface. To znamená, že zdrojový kód není zkompilován přímo do strojového kódu, ale do CIL (Common Intermediate Language). Tento kód je společný pro všechny kompatibilní jazyky z dílny Microsoftu. Může se jednat například o jazyk Visual Basic, Visual J# a další. Tento mezikód je hardwarově nezávislý a je určen pro spuštění v systému VES (Virtual Execution System). Systém VES pak může být implementován na různé platformy. Jazyk je určen především pro systémy Windows, proto hlavní implementací tohoto interpretačního systému VES je .NET Framework. Další implementace vznikla například v projektu Mono, který je multiplatformní a funguje i na většině distribucí systému Linux. Všechny tyto implementace jsou označovaná souhrnným názvem CLR (Common Language Runtime). Pokud chceme tedy spustit program napsaný v jazyce C#, nejprve ho zkompilujeme do hardwarově nezávislého mezikódu. Při každém spuštění programu pak dojde i k spuštění jednoho z CLR, který mezikód ihned převádí do strojového kódu a spuštěný program interpretuje. Průběh spuštění programu od jeho napsání v jazyce C# až do jeho interpretace na cílovém počítači není tak triviální jako je tomu například u jazyka C++ a čtenář se může jednoduše do procesu zamotat, proto je k tomuto textu doplněn obr. č. 7.2 znázorňující celý proces spuštění programu.



Obr. č. 7.2 Proces spuštění programu od jeho návrhu až po samotné spuštění

Kód řídícího programu jsem psal v jazyce C# a jako CLR jsem, vzhledem k nasazení na distribuci operačního systému Linux, použil Mono společně s vývojovým prostředím Mono Develop.

Jedná se o konzolovou aplikaci, která musí řešit současně více výpočtů. Proto každému bloku kódu zajišťující určitou logickou část bylo přiděleno samostatné vlákno. Tyto vlákna mezi sebou komunikují a vytváří tak plynulý a bezchybný chod. Vlákna jsou zobrazena na obr. č. 7.3. Aplikace obsahuje i další logické bloky, které budou společně s jednotlivými vlákny popsány v následujících podkapitolách. Veškerý zdrojový kód je dostupný na přiloženém CD.



Obr. č. 7.3 Vlákna serverové aplikace

7.4.1 Konfigurační soubor

Prvním rozebíraným logickým blokem bude blok poskytující konfigurační nastavení. Tento blok je na prvním místě protože jej využívají všechny ostatní logické bloky při načítání konfiguračních dat. Serverová aplikace nemá žádné uživatelské rozhraní. Veškeré nastavení a řízení činnosti se provádí z klientské aplikace nastavováním konfiguračních proměnných. Pro jejich uchování v době odpojeného napájení jsem použil XML soubor. Stále musíme mít na paměti, že se jedná o firmware pro vestavěné zařízení a zápis veškerých dat probíhá do paměti typu Flash, která má omezený počet zápisů i čtení. Proto ihned po startu programu se data z tohoto konfiguračního souboru načtou do paměti RAM. Veškeré další čtení dat již probíhá z paměti RAM bez přístupu na Flash paměť. Pokud server přijme požadavek na nastavení konfiguračních hodnot z klientské aplikace, dojde k aktualizaci hodnoty proměnné jak v paměti RAM, tak i v konfiguračním souboru. Zápis do souboru provádím okamžitě po přijetí konfigurační hodnoty, abych předešel ztrátě této informace při nenadálém výpadku napájení. Nastavení nové hodnoty bude probíhat pouze výjimečně a na povel uživatele, proto nehrozí zničení Flash paměti častými zápisy.

7.4.2 Watchdog

V případě, kdy se v řídící aplikaci vyskytne chyba a aplikace je ukončena, jsem potřeboval mechanismus, který systém uvede opět do provozu. Tato chyba se ale nemusí vyskytnout pouze v řídící aplikaci. Může dojít k chybě v samotném operačním systému. Pro všechny tyto situace používám mechanismus zvaný Watchdog. Jedná se o systémovou funkci, která se aktivuje otevřením souboru `/dev/watchdog`. Poté se spustí časovač, a pokud nedojde po dobu 1 minuty k žádnému zápisu do tohoto souboru, je proveden restart celého systému. Serverová aplikace obsahuje metodu běžící v samostatném vlákně, která v pravidelných 30sekundových intervalech do tohoto souboru zapisuje. V případě nečekaného ukončení aplikace je ukončeno i zapisovací vlákno a celý systém je restartován. Tato funkcionalita lze vypnout z klientské aplikace. Vypnutí se pak provádí zapsáním písmene „V“ do souboru a jeho následném zavření.

7.4.3 Respondér

Pro zajištění stálého kontaktu klientské a serverové aplikace slouží metoda `Responder`. V této metodě dochází v nekonečné smyčce ke sledování klientem vyslaných dat a odpovídání na tyto data. Tuto komunikaci jsem realizoval pomocí protokolu TCP, který nám zajistí oproti protokolu UDP spolehlivé přenesení dat. Komunikační protokol, který jsem vytvořil pro respondér přijímá 3 typy zpráv.

Prvním typem přijímaných zpráv jsou tzv. „Are you alive“ zprávy. Tyto zprávy slouží pro ověření komunikace mezi serverem a klientem. Klient zasílá serveru zprávu „Telemetry, are you alive?“ a očekává odpověď „Telemetry is ready!“.

Druhým typem zpráv jsou zprávy pro získání nebo nastavení hodnoty konkrétní konfigurační proměnné. Pro získání hodnoty konfigurační proměnné ze serveru zasílá klient zprávu ve tvaru

```
#Nazev_promenne$#
```

a server odesílá odpověď ve tvaru

```
#Nazev_promenne$hodnota#
```

Kde „hodnota“ je hodnota proměnné získána z paměti RAM. Pokud by server obdržel zprávu od klienta v tomto tvaru

```
#Nazev_promenne$hodnota#
```

znamenalo by to, že klient požaduje změnu nastavení. Server provede požadovanou změnu a pošle klientovi zpět zprávu ve stejném znění. Výjimku v této komunikaci tvoří pouze zprávy, jejichž název proměnné je „CANMessagesPriority1“, „CANMessagesPriority2“, nebo „CANMessagesPriority3“. Význam těchto zpráv je vysvětlen v kapitole 7.4.5. Tyto zprávy se vyznačují možností zasílat jako platnou hodnotu prázdné pole, proto pro odlišení od zasílání nové hodnoty proměnné a dotazu na

hodnotu proměnné je místo prázdného pole použit znak „?“. Zpráva dotazující nastavení proměnné „CANMessagesPriority1“ serveru pak vypadá takto

```
#CANMessagesPriority1$?#
```

Speciální a zároveň posledním typem zpráv jsou zprávy přímo nastavující hardware. U těchto zpráv je před název proměnné umístěn speciální znak „@“. Jinak se komunikace neliší od komunikace s běžnou proměnnou. Tvar zprávy pak může vpadat například takto

```
#@accelX$True#
```

Konkrétně tato zpráva slouží k nastavení akcelerometru pro povolení snímání dat v ose X.

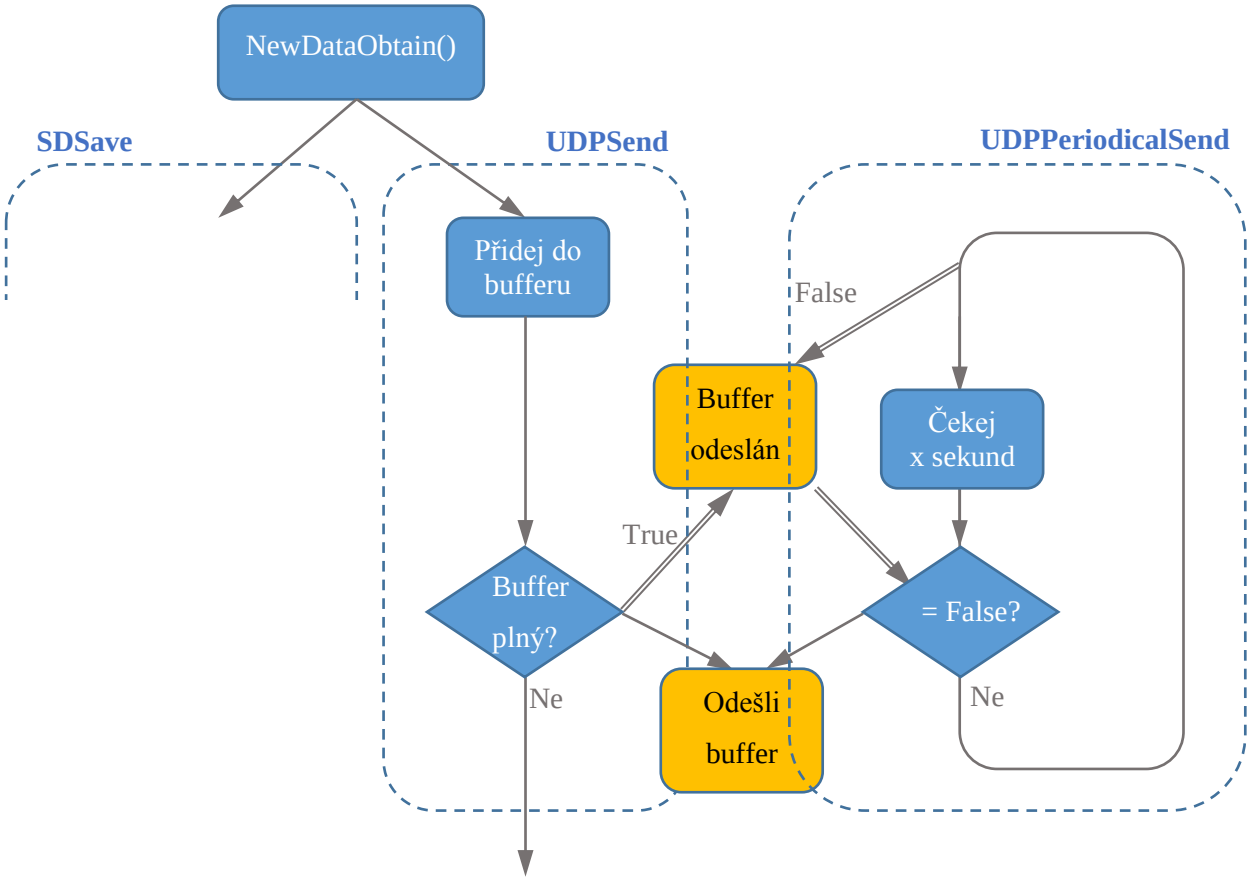
7.4.4 Inerciální senzory

Dalšími logickými bloky jsou 4 bloky pro data z jednotlivých inerciálních senzorů. Jedná se o blok pro získávání údajů z akcelerometru, kompasu, gyroskopu a teploty gyroskopu. Každý tento blok běží v samostatném vlákně, ale jejich funkce je velmi podobná proto je rozeberu společně. Veškeré metody pro práci s inerciálními senzory i s knihovnou `libi2c.so` jsou umístěny do třídy `InertialSensors`. Hlavní `Main` třída obsahuje pouze metody pro tyto 4 vlákna, které přesně v zadaném intervalu získávají data voláním příslušné metody třídy `InertialSensors`. Získaná data ukládají do pole, které následně předávají metodě `NewDataObtain`.

Tato metoda slouží pro sběr všech nově naměřených dat, ať už se jedná o inerciální senzory, GPS modul, nebo CAN zprávy zachycené na sběrnici formule. V této metodě je v závislosti na konfiguračních proměnných volána funkce `UDPSend` pro odesílání a funkce `SDSave` pro ukládání dat. Obě tyto funkce pracují podobným způsobem, proto budou opět rozebrány společně.

Po zavolání metody s nově obdrženými daty jsou tyto data přidány do bufferu. Po naplnění bufferu je celý odeslán na klienta, příp. uložen na Flash paměť. Vzhledem k získávání dat ve více vláknech je pro předejití možných chyb umožněno v jeden okamžik přistupovat k bufferu pouze jednomu vláknu. Toho jsem docílil použitím zámků.

V případech, kdy je z nejrůznějších důvodů přerušeno přijímání nových dat, nebo nepřijdou žádná nová data v nastaveném časovém limitu, by již přijatá data nebyla odeslána až do doby zaplnění bufferu. Proto jsem použil jak pro odesílání dat, tak pro jejich ukládání na Flash paměť další metodu spuštěnou ve vlákně. Tato metoda v nastavených intervalech kontroluje, zda se data správně odesílají resp. ukládají. V případě, že dojde k situaci, kdy buffer nebyl po zadaný časový okamžik odeslán/uložen, dojde k okamžitému odeslání resp. uložení stávajících dat bez ohledu na to, zda je buffer plný či nikoliv. Princip funkce této metody je zobrazen na obr. č. 7.4.



Obr. č. 7.4Princip hlídání odeslání bufferu

7.4.5 Získávání CAN dat

V dalším vlákně běží metoda pro získávání dat z CAN sběrnice. Tyto data můžeme získávat dvěma způsoby. Prvním způsobem je příjem pouze vyžádaných dat. Druhým pak příjem všech dat zachycených na CAN sběrnici. Pro první případ je v konfiguračním nastavení uloženo pole obsahující identifikátory žádaných zpráv. Pro nižší zatížení sběrnice používám tři priority zpráv. V konfiguračním nastavení je pak pro každou prioritu zpráv vlastní pole obsahující identifikátory konkrétní priority. Zavedením priorit zpráv jsem docílil zrychlení aktualizace důležitých hodnot. Hodnoty, které mají pomalou změnu, mají prioritu 3 a jejich data jsou na sběrnici vyžadována méně častěji. Mezi takovéto data mohou patřit například data nesoucí teplotu. Mezi rychlé děje, u kterých naopak potřebujeme rychlou odezvu, se řadí například data obsahující otáčky motoru. Rozdělení priorit jsem zvolil tak, že po každých deseti přijatých zprávách s prioritou 1 se odešle jedna zpráva s prioritou 2. A po každých pěti přijatých zprávách s prioritou 2 se odešle jedna zpráva s prioritou 3. Metoda tedy postupně prochází všechna pole a vysílá požadavek na zprávu s konkrétním identifikátorem. Pokud po zadanou dobu neobdržíme odpověď na vyslaný dotaz, nebo pokud počet odpovědí s jiným identifikátorem než je žádaný překročí zadanou hodnotu, aktuální identifikátor zprávy přeskakujeme a přecházíme k dalšímu identifikátoru v poli. Procházení všech polí se cyklicky opakuje v nekonečné smyčce. Pokud bychom v konfiguračním nastavení zvolili druhou možnost, tedy

příjem všech zpráv na sběrnici, metoda bude kromě dotazování zpráv s konkrétním identifikátorem také číst veškeré zprávy na sběrnici a ty dál předávat metodě `NewDataObtain` pro další zpracování.

7.4.6 Získávání údajů z GPS modulu

Hlavní třída `Main` aplikace obsahuje metodu `GPSFunction`, která běží ve vlákně a podobně jako v případě inerciálních senzorů pouze dotazuje data z třídy `GPS`. V této `GPS` třidě jsou obsaženy všechny metody zajišťující získání dat z GPS modulu, jejich zpracování a v neposlední řadě i řízení a nastavování samotného modulu. Výjimku tvoří pouze konfigurace modulu binárními zprávami, která je obsažena v třídě `GpsBinaryMessagesClass`. Pomocí binárních zpráv můžeme provést i ne zcela obvyklá nastavení modulu a je nutné je využívat s opatrností. I když se nejedná o standardní nastavení modulu, byl jsem nucen některá z nich měnit. Především jsem měnil konfiguraci samotné UART sběrnice, nastavení formátu a frekvence výstupních zpráv a další. Binární zprávy využívám i k zjištění aktuální obnovovací frekvence a její zobrazení v klientské aplikaci.
Po vytvoření komunikace s GPS modulem je z hlavního vlákna periodický volána metoda třídy `GPS` pro zjišťování aktuální polohy. Tato metoda přijme data, provede kontrolu, zda se jedná o zprávu z GPS modulu zaslanou ve formátu NMEA0183, zkontroluje hodnotu kontrolního součtu a nakonec zavolá metodu pro zjištění typu zprávy a její dekódování. Výsledná poloha je uložena do struktury `Loc` a předána zpět hlavnímu vláknu. Struktura `Loc` definovaná v také v třídě `GPS` slouží pro uložení informace o poloze. Navíc obsahuje údaj obsahující přesný čas příjmu zprávy GPS modulem a také příznak značící platnost získané polohy. Pro přenos i uložení zprávy v její co nejmenší velikosti jsem ji bez ztráty jakýkoliv dat zakódoval následujícím způsobem. Předpokládám-li, že všechna data ve zprávě obsahují svojí maximální hodnotu, pak získávám data:

Čas: 23:59:59.999
Zeměpisná šířka: 89°99.9999‘ N
Zeměpisná šířka: 179°99.9999‘ E

Při kódování vynechávám všechny nenumerické znaky. Zbylé číslice pak uložím jako 4bitové hodnoty. Pro rozeznání kladných a záporných hodnot, resp. severu a jihu a východu a západu použiji následující převodní tabulku:

N = 12 (1100b)
S = 13 (1101b)
E = 14 (1110b)
W = 15 (1111b)

Záměrně jsem použil nejvyšší hodnoty uložitelné na 4 bitech, protože nejvyšší možná číslice, číslice 9, bude zakódována jako číslo 9, tedy jako 1001b. Hodnoty severu/jihu a východu/západu tedy mohu použít v kódovaném slově jako zarážky pro ověření správnosti. Podoba výsledného zakódovaného slova je zobrazena v tabulce 7.4. Modré rámečky značí velikosti 1 Bytu, proto výsledná velikost celého slova je pouze 15 Bytů oproti délce 38 Bytů, což by zpráva zabírala bez komprese. Předposlední nibble obsahuje příznak platnosti. Poslední nibble je pak pouze zarovnání na celé Byty.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2 3	5 9	5 9	9 9	9 8	9 9	9 9	9 9	9 12	1 7	9 9	9 9	9 9	9 14	1 0
čas				Zeměpisná šířka				N	Zeměpisná výška				E	V

Tabulka 7.4 Zakódovaní informace o poloze

7.4.7 Odesílání naměřených hodnot

Všechny naměřené hodnoty včetně hodnot získaných z CAN sběrnice formule jsou odesílání pomocí protokolu UDP. Na rozdíl od přenášení nastavení mezi klientem a serverem, kde jsem pro komunikaci použil protokol TCP, jsem pro odesílání naměřených dat zvolil protokol UDP. Tento protokol neobsahuje techniky, které by zaručili doručení vyslané zprávy příjemci. Na druhou stranu nepoužitím těchto mechanismů ve velké míře klesá potřebná režie přenosu. Pro zobrazování Live dat je potřeba co nejrychlejší spojení, tedy spojení s co nejnižší režií. Zároveň ztráta paketů nám výrazným způsobem neovlivní zobrazování dat. Serverová aplikace počítá i s možností připojení více než jednoho klienta. Z toho důvodu jsou data vysílána na broadcastové adrese sítě. Díky tomu, že protokol UDP je nespojovaný, jsou data ze serveru odesílány pouze jednou a to i v případě připojení mnoha klientů. Po zvážení všech aspektů protokolu TCP a UDP bylo použití protokolu UDP jasnou volbou. Odeslání dat probíhá z vyhrazeného bufferu po jeho zaplnění. Při hledání správné velikosti bufferu je vhodné zvolit kompromis mezi rychlostí a spolehlivostí. Každý odeslaný UDP rámec obsahuje hlavičku o velikosti 4 Bytů. Pokud bychom například nastavili velikost bufferu na 16 Bytů, tak 20 % z odesílané zprávy by bylo využito hlavičkou. Pokud bychom natavili velikosti bufferu například na 800 Bytů, v odesílané zprávě by hlavička zabírala pouze 0,5%, ale při ztrátě paketu bychom přišly o velké množství dat a zobrazování by nebylo plynulé. Velikost tohoto bufferu je možné nastavovat z klientské aplikace, a tak komunikaci přizpůsobit konkrétním podmínkám. Odesílaná věta se skládá z různých zpráv, jejichž struktura je zobrazena v tabulce 7.5.

ID	SIZE	data
1B		1 až 15 Bytů dat

Tabulka 7.5 Struktura přenášených zpráv

První nibble zprávy udává ID přenášené zprávy a jejich význam je zobrazen v tabulce 7.6. Druhý nibble pak obsahuje počet Bytů dat zprávy. Minimální hodnota je 1 Byte. Maximální hodnota je pak 15 Bytů dat, což je největší číslo uložitelné do 1 nibblu. Této délky dat využívám pro zasílání zpráv obsahující údaje o poloze z GPS modulu. Z charakteru zpráv je zřejmé, že výsledná délka zprávy může být 2 až 16 Bytů.

ID	Název
1	CAN zprávy
2	Teplota gyroskopu
3	Úhlové zrychlení gyroskopu
4	Lineární zrychlení akcelerometru
5	Magnetické pole kompasu
6	GPS poloha

Tabulka 7.6 Význam identifikátorů přenášených zpráv

7.4.8 Logování chování serveru

Posledním logickým blokem je blok logování. Ten poskytuje metody pro ukládání každé významné změny a chyby při běhu programu. V případě problémů je pak možné z těchto zalogovaných událostí zjistit jejich příčinu. Princip logování popíši na příkladu navázání úspěšného spojení mezi serverem a klientem pomoci protokolu UDP. Uvažujme, že máme proměnnou `Globals.status.UdpConnected`, jejíž hodnotu chceme změnit. Zároveň se změnou hodnoty chceme ale i zaznamenat tuto změnu.

K tomu potřebujeme vytvořit pro každou proměnnou metodu, která bude nastavovat hodnotu konkrétní proměnné. Pro tento případ by to byla funkce zobrazená níže.

```
public static void UdpConnected (bool b) { Globals.status.UdpConnected = b; }

public static void StatusChanged<T> (Action<T> action, T value)
{
	action (value);
	if (Globals.config.LoggingEnable == 1) {
		Logging(action.Method.Name, value.ToString ());
	}
}
```

Pro změnu hodnoty proměnné i zalogování této změny volám metodu `StatusChanged<T>.`

```
StatusChanged(UdpConnected, false);
```

Na prvním řádku se provede zavolání metody, která nastaví hodnotu příslušné proměnné. Další řádky už obsahují jen kontrolu povolení logování a volání metody pro uložení změny. Této metodě předávám název proměnné i novou hodnotu. Povšimněte si parametrů metody `StatusChanged<T>,` kdy místo proměnné předáváme referenci na metodu. Tuto část kódu zde zmiňuji, protože reference na metody, neboli delegáti, jsou zajímavou vlastností jazyka C#, kterou se tento jazyk liší i od svého největšího konkurenta jazyka JAVA.
Metoda `Logging` pro ukládání záznamů má vlastní buffer pro ukládání dat na Flash paměť. Po zaplnění bufferu se v dávce provede jeho uložení. Ukládání v dávkách je rychlejší než ukládání po jednotlivých zprávách a hlavně šetří životnost Flash paměti snížením počtu zápisů.

7.5 Metriky

Tato kapitola zobrazuje metriky serverové aplikace. Metriky pro knihovny jsou zobrazeny v tabulce 7.7, pro třídy řídící aplikace pak v tabulce 7.8. Součet těchto metrik obsahuje tabulka 7.9.

Název knihovny	Počet metod	Počet řádků
libcancom.cpp	7	268
libi2c.cpp	18	499
libserialcom.cpp	4	151

Tabulka 7.7 Metriky knihoven serverové aplikace

Název třídy/souboru	Počet metod	Počet řádků
App.config	-	52
GPS.cs	17	524
GpsBinaryMessagesClass.cs	2	141
InertialSensors.cs	26	532
Main.cs	33	1630

Tabulka 7.8 Metriky tříd serverové aplikace

	Počet metod	Počet řádků
Knihovny	29	918
Třídy/soubory	78	2879
Celkem	**107**	**3797**

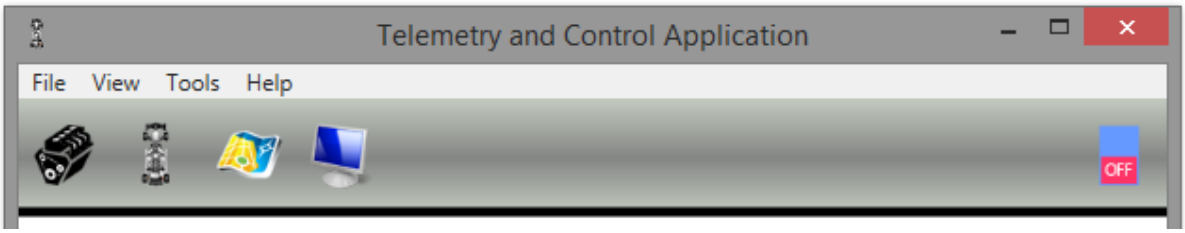
Tabulka 7.9 Celkové hodnoty metriky pro serverovou aplikaci

8 Návrh klientské aplikace

Samotný hardware by nebyl využitelný bez aplikace umožňující zobrazování získaných dat a komunikaci se zařízením. Tato aplikace zaujímá nejvyšší část této diplomové práce a její vývoj zabral nejvíce času. Pro programování jsem zvolil programovací jazyk C# a v době psaní této práce nejnovější platformu .NET 4.5. Aplikace je určená pro operační systémy Windows a poskytuje uživateli přehledné grafické rozhraní pro komunikaci s telemetrickým zařízením. Pro tvorbu tohoto rozhraní jsem použil technologií WPF. Cílem této technologie je oddělit od sebe funkčnost a vzhled aplikace. Každé okno se proto skládá ze dvou souborů. Vzhled aplikace je popsán značkovacím jazykem XAML, který je založený na jazyku XML a velmi podobný jazyku HTML. Samotná funkcionalita okna je poté popsána ve zdrojovém kódu. Tento kód označovaný také jako Code-behind může být v různém programovacím jazyce. Já jsem pro svoji aplikaci použil jazyk C#. Pro ladění aplikace jsem používal vývojové studio MS Visual Studio 2014.

8.1 Hlavní okno – MainWindow

Hlavním oknem aplikace, které se zobrazí ihned po jejím spuštění, je okno `MainWindow`. Toto okno se skládá především z hlavního a grafického menu. Grafické menu je zobrazeno na obr. č. 8.1 a umožňuje uživateli přecházet mezi jednotlivými okny aplikace. Tyto přechody jsou řešeny plynulým přechodem mezi jedním oknem na druhé. Po kliknutí na položku grafického menu je zavolána metoda `onMouseDown` třídy `TransitionClass`. Tato metoda nejprve zjistí název a pozici v menu předchozího zobrazovaného okna. Podle této informace se přechod provede buď směrem doprava, nebo doleva. Místo výrazu přechod bychom mohli používat i výraz přesun, protože přechod jsem realizoval jako horizontální vytlačení aktuálního okna zobrazovaným oknem. Po získání informace o směru přechodu je z nastavení přečtena jeho konfigurace a animovaným přechodem se v hlavním okně zobrazí požadované okno. Veškerá logika těchto přechodů je řešena v třídě `TransitionClass`.



Obr. č. 8.1 Hlavní a grafické menu klientské aplikace

Hlavní menu obsahuje stejné možnosti jako grafické menu navíc ale nabízí přístup do nastavení, které bude podrobně vysvětleno v kapitole 8.6. Posledním důležitým prvkem v tomto okně je obrázek vypínače umístěný na pravé straně menu. Tento vypínač slouží k zapnutí a vypnutí příjmu dat ze serveru. Po kliknutí na tento obrázek se změní jeho barva v závislosti na tom, zda došlo k povolení nebo zakázání přenosu. Proto nám tento „vypínač“ slouží i jako vizuální kontrola aktuálního stavu

aplikace. Kromě těchto dvou menu toto okno obsahuje ještě vyhrazený prostor pro zobrazení jednotlivých oken. Veškerá programová logika hlavního okna se skládá pouze z obsluhy menu, povolování resp. zakazování přenosu a zobrazování vložených oken.

Po povolení příjmu UDP dat ze serveru je v novém vlákně volána metoda `UDPRead` třídy `WiFiFunctions`. Tato třída otevře UDP spojení se serverem, a až do doby než je příjem „vypínačem" zrušen, neustále čte všechna data vysílaná serverem. Po obdržení nových dat ze sítě jsou volány metody `SaveData` a `DecodeData` třídy `LoggingClass`. Funkce metody `SaveData` je stejná jako funkce serverové metody `SDSave` potažmo `UDPSend` popsaná v kapitole 7.4.4. Opět metoda obsahuje buffer, který je plněn obdrženými daty a po jeho zaplnění dojde k uložení do souboru.

Metoda `DecodeData` slouží k dekódování přijatých dat a jejich uložení do příslušných „globálních" proměnných. Ve skutečnosti se nejedná o globální proměnné ve správném slova smyslu, protože v jazyce C# neexistují globální proměnné. Vytvořil jsem statickou třídu `Globals`, která v aplikaci tím že je statická nahrazuje globální proměnné. Ukládání těchto dekódovaných dat pak tedy probíhá do příslušných proměnných struktury `Data` umístěné v této statické třídě. Tato struktura obsahující vždy nejnovější data poskytuje všem ostatním třídám vhodné rozhraní pro přístup k aktuálním datům. Formát kódování dat odesílaných přes protokol UDP je popsáno v kapitole 7.4.7. Přijatá UDP zpráva může obsahovat více zpráv od různých senzorů, proto jsou při dekódování tyto zprávy postupně procházeny a zpracovávány. Jednotlivé získané zprávy jsou dále rozděleny na identifikátor zprávy, délku zprávy a samotná data senzoru. Podle identifikátoru zprávy je pak volána příslušná část kódu.

Pokud se jedná o zprávu akcelerometru, gyroskopu, nebo kompasu, jsou přijatá data převedena na správnou hodnotu podle nastaveného rozsahu. Tento převod provádím jednoduchým vzorcem č. 8.1, kde *value* je přijatá hodnota, $value_{MAX}$ je maximální možná přijatá hodnota a *full_scale* je nastavený rozsah. V závislosti na nastavení povolení logování jsou tyto data následně ukládána.

$$result = \frac{value}{value_{MAX} * full\_scale}$$

Rovnice 8.1 Převod dat do správného rozsahu

V případě obdržení dat s teplotou gyroskopu není nutný další převod a data jsou opět v závislosti na nastavení povolení ukládání těchto dat uloženy či zahozeny.

V případě příjmu informace o aktuální pozici se dekódování mírně komplikuje, protože kromě samotného dekódování dat ze zprávy je nutné také přijatá data ukládat do kolekce obsahující historii přijatých pozic. Z této kolekce pak dochází k vykreslování trasy pohybu na mapě. Pozice jsou v kolekci uloženy jako instance třídy `Location`, proto je nejprve nutné zavolat metodu `ToLocation` třídy `Functions`. Tato metoda převede pozici zadanou v textové podobě do nové instance třídy `Location`, kterou vrátí zpět volající metodě. Tato instance je následně ukládána do kolekce navštívených bodů. Pokud velikost kolekce přesáhne maximální počet ukládaných bodů, je při každém dalším přidání nové pozice smazána jedna poslední pozice a kolekce si tak udržuje stále

stejnou velikost. V závislosti na nastavení může být na aktuálně přijatou pozici vykreslen také bod značící pozici formule na mapě.

Nejsložitější případ nastává při příjmu CAN zprávy. V tomto případě se nejprve projde seznam přijímaných CAN zpráv popsaný v kapitole 8.6.6. Po nalezení id zprávy v tomto seznamu jsou načtena i příslušná nastavení k tomuto typu zprávy. Jedná se o počet datových bitů, typ zprávy, měřítko a offset. Pokud se jedná o data se znaménkem, je pro dekódování hodnoty zavolána metoda `TwoComp` třídy `Functions`. Pokud se jedná o data typu `float`, tedy typ s plovoucí řádovou čárkou, musí být zakódovány pomocí standardu IEEE 754 viz [42]. Pro dekódování je volána metoda `ConvertToDouble` opět třídy `Functions`. Tato metoda převede obdržené bitové pole do proměnné typu `double`, přičemž metoda umí pracovat se všemi třemi typy přesnosti čísel (poloviční, normální i dvojitá přesnost). Pokud jsou data bezznaménkového typu, je možné použít implicitní přetypování a není nutné žádné další metody. V případech všech typů je výsledná hodnota převedena pomocí vzorce č. 8.2 do své správné podoby. Získaná hodnota dat je vydělena měřítkem a následně je k ní přičten offset. Těmito dvěma proměnnými získáváme možnost přesného převedení přijaté hodnoty na konkrétní měřenou veličinu.

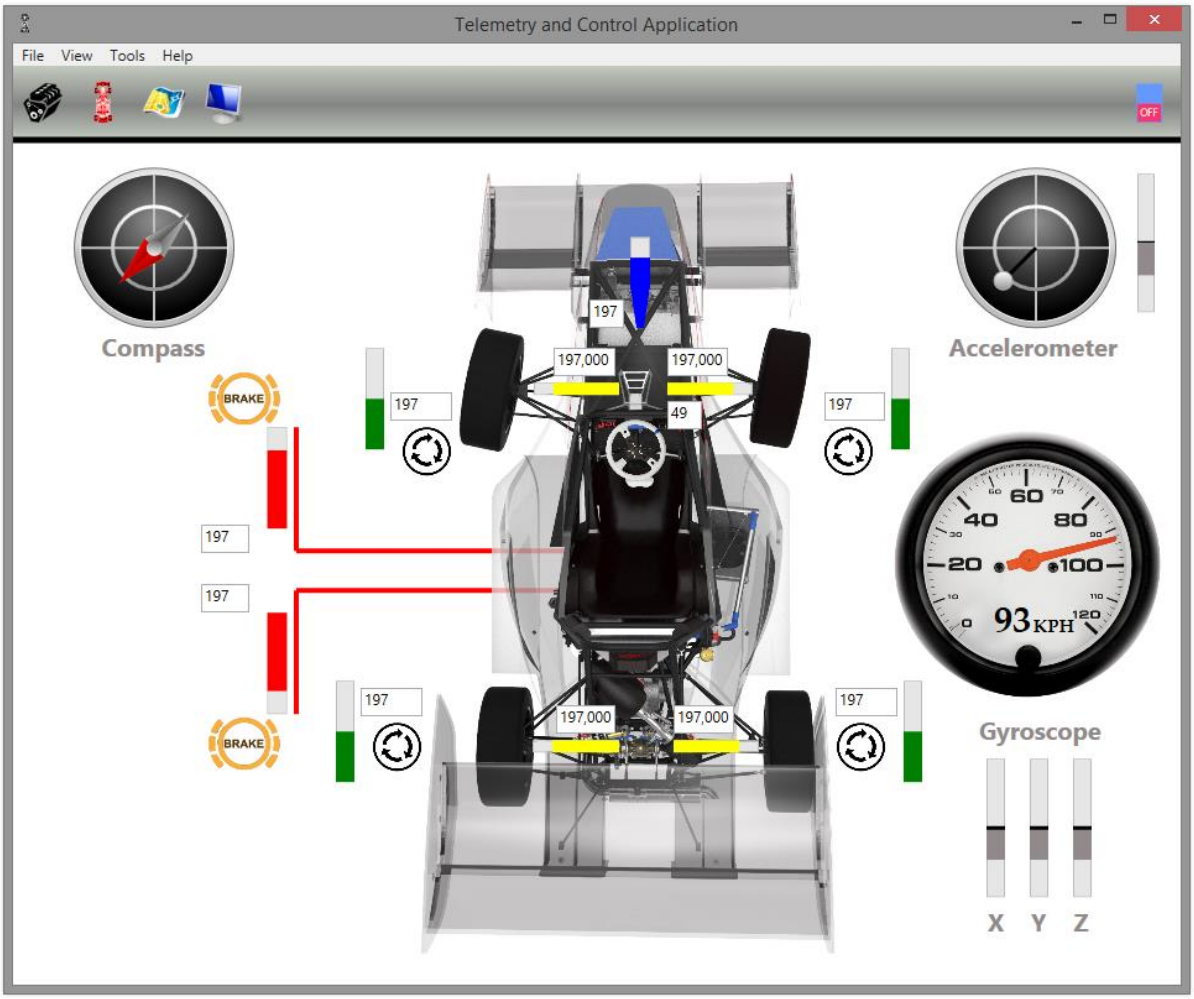
$$result = \frac{value}{scale} + offset$$

Rovnice 8.2 Převod získané hodnoty na reálnou veličinu

Posledním krokem je uložení získaných dat z CAN zprávy. Pokud se jedná o známou zprávu, tj. její alias je shodný s doporučenými aliasy (viz kapitola 8.6.6), je přijatá hodnota uložena přímo do příslušné proměnné v třídě `Globals`. Pokud se jedná o obecnou CAN zprávu, která se nezobrazuje v grafickém rozhraní ale pouze v tabulce `DataView` (viz kapitola 8.5), přistupuje se rovnou k dalšímu kroku, ve kterém je aktualizována aktuální i minimální a maximální hodnota konkrétní CAN zprávy v kolekci `Globals.data.canMessages`. Pokud zpráva ještě v kolekci není, dojde k jejímu přidání.

8.2 Zobrazení šasi – TopView

První z řady možných zobrazitelných oken je okno `TopView`. Toto okno zobrazuje senzory a jejich hodnoty na modelu formule Dragon IV. Vzhled okna je zobrazen na obr. č. 8.2.



Obr. č. 8.2Zobrazení stavu senzorů na šasi formule

Naměřené hodnoty jsou zobrazovány jak naměřenou číslicovou hodnotou, tak i graficky. Nejvýraznějším prvkem okna je tachometr zobrazující aktuální rychlost formule. Při závodech je na tratích dosahováno průměrné rychlosti 55 km/h. Teoretická maximální rychlost formule se pohybuje okolo 120km/h. Reálná je asi o 20 km/h nižší. Proto jsem rozsah tachometru zvolil do 120 km/h. Levý a pravý roh okna zabírají terčíky zobrazující data z kompasu a akcelerometru. Lineární zrychlení získané akcelerometrem je v osách X a Y zobrazováno jako pohyb kuličky po terčíku. V případě nulového zrychlení v ose X i Y je kulička umístěna uprostřed terčíku. V případě dopředného zrychlení formule se pak pohybuje vertikálně. U bočního zrychlení vznikajícího v zatáčkách je její pohyb horizontální. Vyjádřením zrychlení zároveň v obou osách dostáváme plynulé dvourozměrné zobrazovaní zrychlení v terčíku. Zobrazení zrychlení v ose Z jsem řešil pomocí dvou ProgressBarů. Při nulovém zrychlení jsou oba ProgressBary prázdné. Při zvětšení zrychlení se jeden z ProgressBarů začne plnit podle toho, zda se jedná o kladné nebo záporné zrychlení. V případě kompasu je výpočet úhlu natočení střelky v závislosti na hodnotách os X a Y zobrazen rovnicí č. 8.3. Po výpočtu úhlu už stačí pouze převést hodnotu do správného kvadrantu a střelku zobrazit.

$$\alpha = \arctan\left(\frac{y}{x}\right) \ [°]$$

Rovnice 8.3 Výpočet úhlu natočení střelky v závislosti na hodnotách os X a Y

Z inerciálních senzorů jsou v tomto okně zobrazeny také data úhlového zrychlení z gyroskopu. Tyto data zobrazuji pomocí dvou ProgressBarů pro každou z os, stejně jako je řešeno zobrazování zrychlení osy Z u akcelerometru. Dalšími zobrazovanými hodnotami jsou tlak v předním a zadním brzdovém okruhu, aerodynamický tlak změřený v přední části formule Pitotovou trubicí a zdvih všech 4 tlumičů. Všechny tyto hodnoty jsou zobrazovány pomocí ProgressBarů. Zajímavostí v tomto okně je zobrazení rychlosti otáčení jednotlivých kol. Tyto rychlosti jsou zobrazeny pouze číselnou hodnotou, ale každé u každého kola je navíc ProgressBar znázorňující prokluz konkrétního kola. Tuto funkcionalitu řeším tak, že nejprve vypočítám průměrnou rychlost otáčení všech kol. Tato rychlost odpovídá přesně polovině ukazatele. Pokud kolo začne prokluzovat a jeho rychlost otáčení se proto bude lišit od ostatních kol, hodnota ukazatele se u něj zvýší nebo sníží podle toho, zda se kolo točí rychleji nebo pomaleji než ostatní kola. Ukazatele prokluzu dosáhnou maximální hodnoty, pokud se kolo točí alespoň dvakrát rychleji, nebo dvakrát pomaleji než je průměrná rychlost otáčení všech kol. Pro ještě lepší přehlednost se po dosáhnutí prokluzu většího než je zadaná hodnota změní ukazatele na modrou, v případě rychlejší otáčení kola, nebo na oranžovou v případě že se kolo točí pomaleji, tedy je ve smyku. Tato hraniční hodnota se dá nastavit v nastavení, jak bude probráno v kapitole 8.6.7. Poslední, ale neméně zajímavým zobrazovacím prvek, je zobrazení úhlu natočení volantu. Hodnota tohoto úhlu je opět tak jako u ostatních senzorů zobrazena ve své číselné podobě. Navíc se v závislosti na této hodnotě otáčí volant i kola vozidla a tím je dosaženo nejlepší možné interpretace naměřených dat.

O nastavování grafického rozhraní se stará Code-behind. Po zobrazení okna se v novém vlákně spustí metoda `RefreshUI`, která v pravidelných intervalech provádí aktualizaci grafického rozhraní. Pokud dojde k přepnutí do jiného okna aplikace, tak tato metoda není ukončena, ale pouze pozastavena. Při opětovném zobrazení tohoto okna je metoda opět spuštěna bez nutnosti ji znova vytvářet. Tím se ušetří výkonové prostředky a celý běh aplikace je pak plynulejší. Pro získání aktuálních dat metoda přistupuje do struktury `Data` statické třídy `Globals`, ze které čte nejnovější přijatá data, jak bylo vysvětleno v předchozí kapitole. Za účelem plynulé práce s grafickým rozhraním je tato metoda spuštěna ve vlákně. To nám přináší výhodu v podobě nezamrznutí grafického rozhraní v době, kdy metoda provádí výpočty. Na druhou stranu nám to ale mírně komplikuje situaci, kdy metoda ve vlákně chce přistupovat k prvkům grafického rozhraní, které ale vlastní jiné vlákno. V grafickém uživatelském rozhraní založeném na WPF není dovoleno přistupovat k objektům grafického rozhraní z jiného vlákna, než které tyto objekty vytvořilo. Existuje ale řešení v podobě metody `Dispatcher.Invoke` a delegátů. Pokud chceme nastavit určitou vlastnost prvku jiného vlákna, musíme přesměrovat nastavování vlastnosti z aktuálního vlákna na vlákno, které tento prvek vlastní.

Předání metody jinému vláknu provádím pomocí delegátů. Jak již naznačila kapitola 7.4.8 delegáti, neboli reference na metodu, jsou užitečnou vlastností jazyka C#. V jazyce C++ by se dal chápat jako typově bezpečný ukazatel na funkci. Delegáti v jazyce C# předepisují přesný tvar metod. Pokud metoda odpovídá deklaraci delegáta, lze se delegátem na tuto metodu odkazovat.

Vysvětlení předvedu na příkladu. Pokud by se nejednalo o vícevláknovou aplikaci, šlo by nastavit stav checkBoxu jednoduše přímým přístupem k jeho vlastnosti.

```
checkBox.IsChecked = True;
```

Uvažujme ale, že potřebujeme nastavit hodnotu checkBoxu z jiného vlákna než které tento checkBox vytvořilo. V tomto případě je nutné použít kód zobrazený níže a nastavení ovládacího prvku provést zavoláním této metody a předání ovládacího prvku a hodnoty jeho vlastnosti parametrem.

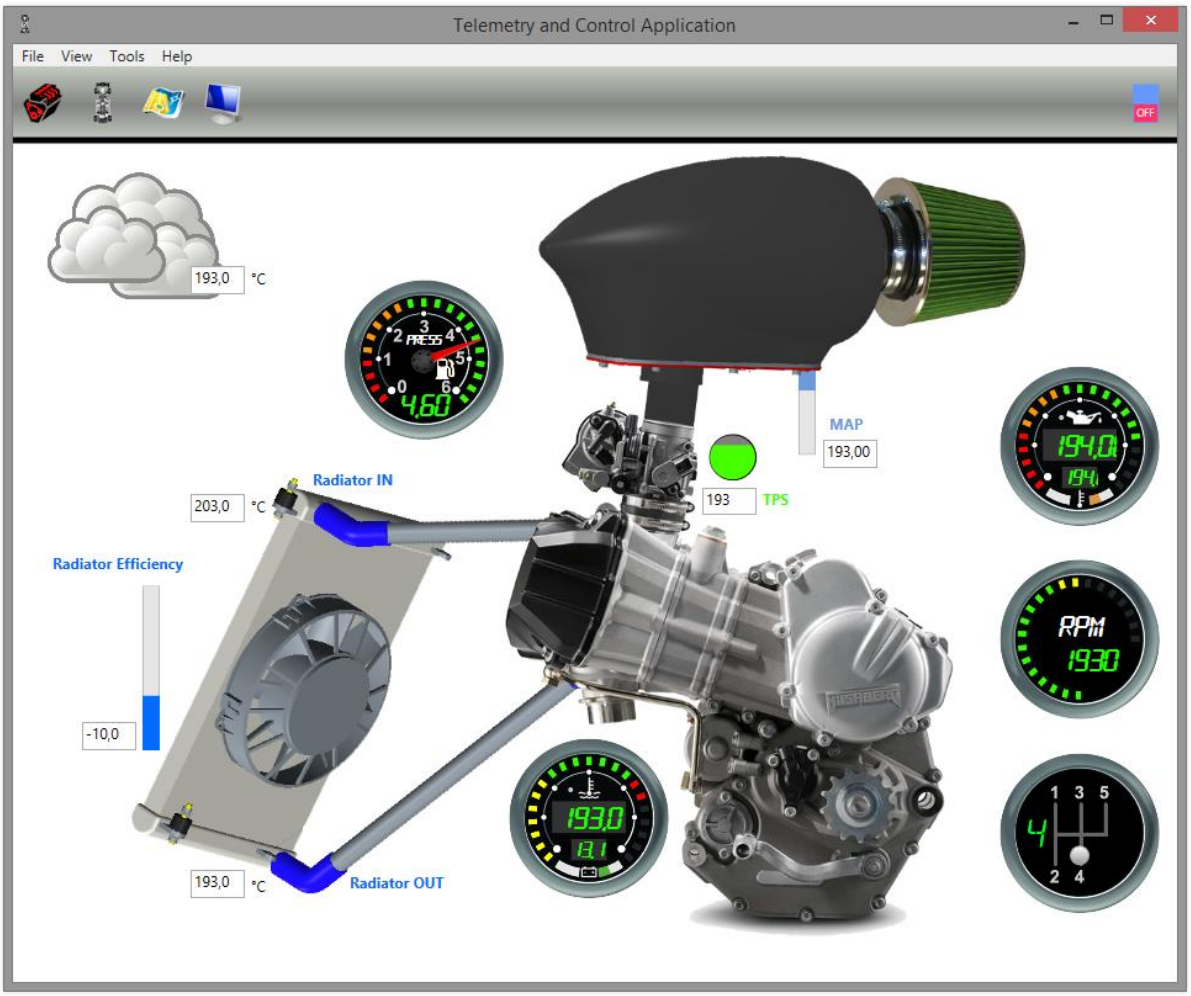
```
//Zde vytvořím delegáta metody, kterou budu předávat druhému vláknu
private delegate void TSetCheckBoxDelegate(CheckBox checkBox, bool value);

//Metoda, která provede nastavení hodnoty checkBoxu podle zadané hodnoty
public static void TSetCheckBox(CheckBox checkBox, bool value)
{
	if (checkBox == null) //Pokud nedostanu žádný prvek, metoda končí
		return;

	 if (checkBox.Dispatcher.CheckAccess())    //Zjištění, zda checkBox patří
stejnému vláknu jako tato metoda
	{	//Pokud ano, nastav novou hodnotu checkBoxu
		checkBox.IsChecked = value;
	}
	else   //checkBox vlastní jiné vlákno
	{
		//Inicializuji delegáta metodou, kterou budu předávat
		TSetCheckBoxDelegate d = new TSetCheckBoxDelegate(TSetCheckBox);

		//Metodu reprezentovanou delegátem předám vláknu, kterému checkBox
		patří
		 checkBox.Dispatcher.Invoke(d, new object[] { checkBox, value });
	}
}
```

Metoda nejprve zkontroluje, zda má přístup k ovládacímu prvku. Pokud ne a prvek vlastní jiné vlákno, vytvoří delegáta referující tuto metodu. Delegát je předán vláknu, které jej vlastní a metoda v aktivním vlákně končí. V druhém vlákně je znova spuštěna tato metoda. Tentokrát metoda vyhodnotí, že aktuální vlákno vlastní ovládací prvek a provede nastavení jeho vlastnosti.

Jedná se o celkem elegantní způsob přístupu k prvkům jiného vlákna. Nevýhodou ovšem je nutnost psát tento úsek kódu pro každou nastavovanou vlastnost. Všechny tyto metody a delegáti jsou v aplikaci uloženy v třídě `Delegates`.

8.3 Zobrazení motoru – EngineView

Dalším zobrazovaným oknem je okno `EngineView` zobrazující data ze senzorů týkajících se motoru. Vzhled okna je zobrazen na obr. č. 8.3.



Obr. č. 8.3 Zobrazení senzorů motoru

Dominantou tohoto okna je motor z motocyklu Husaberg FE 570 stejný, jako je použit na formuli Dragon IV. Kolem tohoto motoru jsou rozmístěné grafické prvky zobrazující naměřenou hodnotu. K motoru je napojen chladič, u kterého zobrazuji teplotu na jeho vstupu i výstupu. Z těchto dvou hodnot můžu vyjádřit efektivitu chladiče, která je zobrazována jak v číselné podobě, tak graficky ProgressBarem. Čím je vyšší efektivita chlazení, tím je vyšší i hodnota ProgressBaru. Maximální efektivita chlazení je zobrazována při ochlazení výstupního média o více, než je nastavená hodnota. Všechna grafická okna aplikace jsou konfigurovatelná z uživatelského nastavení, které umožňuje jejich přizpůsobení konkrétnímu použití. Tyto nastavení budou detailně popsány v kapitole 8.6.8. Kromě teploty chladícího média na vstupu a výstupu chladiče je v tomto okně zobrazována ještě teplota vzduchu, umístěna v levém horním rohu okna, a teplota chladícího média v hlavě motoru. Tato teplota je znázorněna grafickým ukazatelem. Optimální teplota zobrazená zelenou barvou se pohybuje od 30 % do 60 % z maximálního rozsahu měření teploty. Nižší teplota označená žlutou barvou je stejně tak jako teplota vyšší označena červenou barvou nevhodná k práci mechanických komponent motoru a motor nesmí být při těchto teplotách zatěžován maximální zátěží. Ve spodní části tohoto ukazatele je menší ukazatel zobrazující napětí na baterii. V případě posunu ukazatele směrem doprava a jeho zbarvení zelenou barvou dochází k nabíjení baterie. V opačném případě, kdy je ukazatel zbarven červenou barvou dobíjecí soustava motoru nestíhá baterii dobíjet a dochází k

jejímu vybíjení. V případě dosažení kritických hodnot ať už se jedná o překročení maximální možné pracovní teploty motoru, nebo výrazném vybíjení baterie, dojde k rozsvícení varovné kontrolky.

S měřením teploty chladícího média souvisí i měření teploty motorového oleje, což je zobrazeno společně s hodnotou tlaku oleje na dalším ukazateli. Teplota oleje je pro nás pouze orientační a zajímá nás v pouze v případě překročení maximálních hodnot, proto je její ukazatel umístěn ve spodní části měřáku a řešen v podobném stylu jako ukazatel dobíjení baterie. Hlavní část zabírá ukazatel tlaku oleje, což je pro nás důležité sledovat hlavně v zatáčkách, kdy může vlivem odstředivé síly dojít k přelití oleje na jednu stranu. Olejové čerpadlo pak nasaje místo oleje vzduch. Při déle trvající odstředivé síle a výpadku mazání může dojít až k zadření motoru, jak jsme zjistili při testování formule Dragon III. Tlak oleje požadujeme co nejvyšší. Jeho optimální hodnota začíná na 50 % rozsahu ukazatele. Při nižších hodnotách tlaku není motor správně mazán a tato situace je indikována opět rozsvícením varovné kontrolky.

Okno obsahuje ještě jeden měřák tlaku. Tentokrát se jedná o měření tlaku paliva. Tento ručičkový měřák zobrazuje aktuální tlak paliva proudícího ke vstřikovači a dokáže nám odhalit závadu jak na regulátoru tlaku, tak na samotném čerpadle. Zadření čerpadla se nám stalo osudným při loňských závodech v Maďarsku, kdy nebyla žádná možnost jak tento tlak monitorovat a hlídat. S telemetrií vyvinutou v rámci této diplomové práce je možné sledovat tlak paliva v každou chvíli a předejít tak možnému zadření palivového čerpadla a nedojetí závodu. Se samotným chodem motoru souvisí ukazatele aktuálních otáček, otevření škrtící klapky a podtlaku v sání změřeného čidlem MAP. Pro zobrazení naměřených dat ve všech těchto ukazatelích je nutné tyto data převést z měřeného rozsahu do rozsahu jednotlivých ukazatelů. K tomu jsem použil rovnici 8.4. Výsledek rovnice je poté v %.

$$value = \frac{actual\ - min}{max - min} * 100\ [\%]$$

Kde: actual = aktuální naměřené hodnota
min = minimální možná naměřitelná hodnota
max = maximální možná naměřitelná hodnota

Rovnice 8.4 Výpočet procentuální hodnoty z naměřené hodnoty

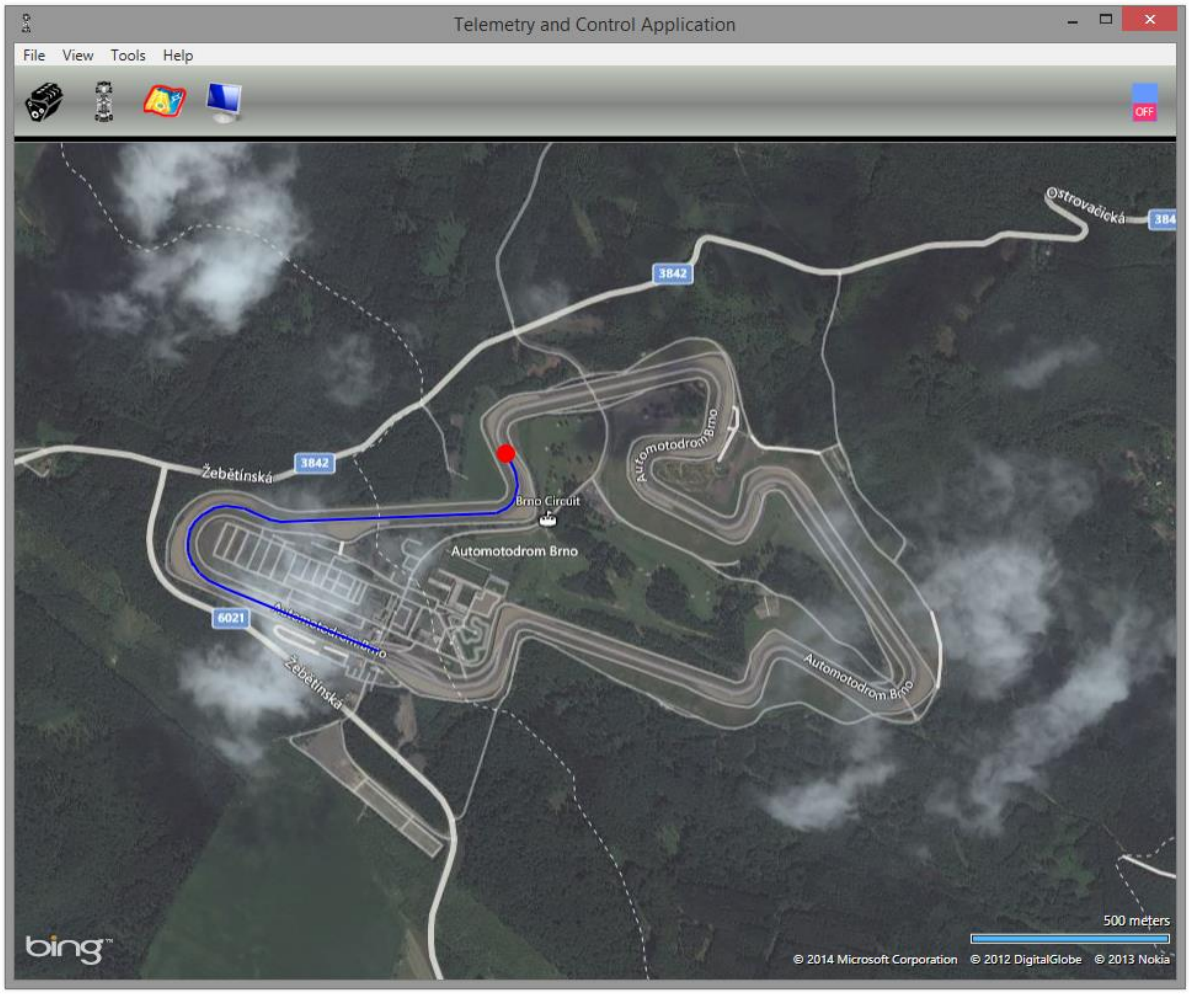
Posledním zobrazovacím prvkem je v tomto okně zobrazení zařazené rychlosti. Prvku zobrazujícímu zařazenou rychlost jsem dal vzhled automobilové řadící kulisy a zařazená rychlost je proto z tohoto prvku na první pohled viditelná. Zařazená rychlost je v motoru měřena lineárním potenciometrem měřícím natočení řadícího válce. Tato hodnota je přenášena až do klientské aplikace, kde probíhá její zobrazení. Použitím lineárního potenciometru odpovídá každé rychlosti přesně vymezený interval hodnot. Tento interval je společně s mnoha dalším nastavením možné měnit v nastavení aplikace.

Princip aktualizace dat grafického rozhraní je stejný jako v případě zobrazení dat na šasi vozu. Opět jsem použil metodu běžící ve vlastním vlákně, která periodicky v zadaném intervalu získává data a tyto data pomocí delegátů předává vláknu obstarávajícím grafické rozhraní k provedení změn. Po změně aktivního okna na jiné je toto vlákno pozastaveno a nezatěžuje svými výpočty ostatní vlákna.

8.4 Poloha na mapě – MapView

Údaj o aktuální poloze získaný z GPS modulu je v tomto okně zobrazován přímo na mapě. Společně s tímto bodem je na mapě zobrazována i historie pohybu bodu a tím je umožněno sledování trasy vozidla. Po obdržení nové polohy je bod zobrazující aktuální polohu na mapě ihned aktualizován a zobrazen na své nové poloze. Pokud navíc bod změní svoji polohu o více než je zadaná hodnota, je k tomuto bodu vykreslena trasa z poslední známé polohy. Touto technikou předcházím chaotickému vykreslování trasy v jednom místě v době, kdy se formule nepohybuje, případně se pohybuje velmi pomalu. Projetá trasa není do mapových podkladů pevně zakreslována, ale jsou ukládány pouze průjezdní body, z kterých se trasa po každé aktualizaci mapy vykresluje do vlastní vrstvy. Tento přístup jsem použil z toho důvodu, aby vykreslovaná trasa měla vždy stejnou šířku bez ohledu na zvětšení mapy či změnu mapových podkladů. Pokud chceme vidět trasu detailněji, jednoduše stačí zvětšit mapu. Šířka trasy se nezvětší zároveň s mapovými podklady, ale zůstane neměnná a tím nám nezakryje detaily mapy. Naopak pokud nám mapa stačí pouze pro představu o trase a nepotřebujeme znát každý její detail, stačí mapu zmenšit. Šířka trasy se opět nemění, resp. vzhledem k mapovým podkladům se změní a dostáváme ideální zobrazení trasy ve všech rozlišeních. Po každé změně v zobrazení map, tj. změny viditelné části mapy, nebo přiblížení/oddálení, dojde k uložení této hodnoty. Po přepnutí do jiného okna a následného návratu zpět k zobrazení mapy zůstanou mapy v takovém zobrazení, v jakém byly zanechány při poslední návštěvě tohoto okna.

Jako mapové podklady jsem použil mapy Bing. Pro získání mapových podkladů služby Bing je nutné nejprve vytvořit vývojářský účet na adrese www.bingmapsportal.com. Po jeho vytvoření jsou pro nekomerční a vzdělávací projekty tyto mapové podklady zdarma přístupné. Jejich omezením je maximální možný počet uživatelů zároveň přistupujících k mapám, maximální počet přístupů za den a v neposlední řadě také použití těchto podkladů pouze pro 3 rozdílné aplikace. Omezení počtu přístupů se i v této základní verzi map pohybuje v řádu tisíců, což je pro vyvíjenou aplikaci naprosto dostačující. Pro samotnou práci s těmito mapovými podklady jsem použil vývojový nástroj „Bing Maps Windows Presentation Foundation (WPF) Control, Version 1.0“ přístupný na [40]. Tento nástroj ulehčuje práci s mapovými podklady. Jeho nevýhoda je ale v nutnosti spuštění pouze při aktivním internetovém spojení. Proto i pro zobrazování polohy formule na mapě je nutné mít aktivní internetové připojení. Příklad zobrazení okna s mapami je zobrazen na obr. č. 8.4. V závislosti na nastavení zobrazování map se výsledný vzhled může lišit. V nastavení je možné nastavit typ mapových podkladů, velikost bodu znázorňující aktuální polohu a další. Více o těchto nastaveních v kapitole 8.6.9.



Obr. č. 8.4 Zobrazení aktuální polohy formule a projeté trasy

8.5 Zobrazení všech dat v tabulce – DataView

Posledním z oken přístupných z grafického menu je okno `DataView`. Toto okno zobrazuje všechny přijaté hodnoty v přehledné tabulce typu `DataGrid`. Pro zobrazení konkrétních dat v této tabulce musí být jejich zobrazení povoleno. Toto povolování resp. zakazování se provádí opět v nastavení aplikace viz kapitola 8.6.5. Kromě zobrazení názvu a aktuální hodnoty zobrazuje tato tabulka také minimální a maximální naměřené hodnoty pro každá obdržená data. Pro uživatelsky přívětivé ovládání tabulky poskytuje tabulka možnost libovolného přesouvání sloupců nebo již zmiňované zakázání zobrazování nepotřebných dat.

Po otevření tohoto okna se v případě, že ještě neexistuje, vytvoří vlákno pro metodu `DataViewRefresher` třídy `DataViewClass`. Tato metoda cyklicky prochází kolekci všech přijatých dat. Pokud je povoleno zobrazování konkrétní proměnné, je její hodnota i s názvem uložena do objektu `Globals.dataFields`. Tento objekt je typu `BindingList` a po zavolání metody `Refresh` zobrazovací tabulky jsou data v tabulce automaticky aktualizována daty z tohoto objektu. Metoda `DataViewRefresher` provádějící operace nad tabulkou opět běží ve svém vlastním vlákně, proto jak pro přidávání dat do kolekce, tak i pro volání metody pro jejich následnou aktualizaci v tabulce jsem

musel použít předávání delegátů grafickému vláknu, stejně jak tomu bylo v případech zobrazení dat v grafickém prostředí motoru i šasi formule. Samotné přidávání dat do kolekce probíhá tak, že nejprve se prochází pevně dané zprávy. Mezi tyto zprávy patří data z inerciálních senzorů a GPS modulu. Poté se prochází pole všech zpráv přijatých po CAN sběrnici. Pokud je zobrazení dané zprávy povoleno, algoritmus zkontroluje, zda se tato konkrétní zpráva již nachází v kolekci na stejné pozici, na jakou je přidávána. Pokud ano, znamená to, že od minulého přidávání dat do kolekce nedošlo k povolení ani k zakázání žádné z proměnných předcházející této proměnné. V opačném případě, kdy došlo k povolení nové proměnné a algoritmus na tuto proměnnou narazí, přestane přidávat další data do kolekce, kolekci zcela vyprázdní a začne ji znova plnit. Změna v povolení nebo zakázání zobrazení konkrétní zprávy probíhá pouze výjimečně. Ve velké většině případů dochází pouze k aktualizaci zpráv, které jsou stále ve stejném pořadí. Tento přístup k přidávání dat do kolekce jsem tedy zvolil z toho důvodu, že ušetřím prostředky pro vyprazdňování a opětovné plnění kolekce při každé obnově jejich hodnot. Tento přístup také nevyžaduje vyhledávání aktualizované hodnoty v kolekci, protože předpokládá, že se data nachází na stejné pozici jako v předešlé iteraci. Výsledná smyčka aktualizující data je tedy velmi rychlá a spotřebovává pouze malé množství prostředků procesoru. Zároveň je tento způsob velmi spolehlivý, protože k jediné nekonzistenci dat, ke které může dojít, je nevykreslení celé části tabulky, ale pouze její části do nově povolené nebo zakázané proměnné. K této situaci dochází pouze při změně zobrazovaných proměnných. Doba trvání této nekonzistence tabulky s přijatými data závisí na rychlosti vykreslování, kterou si může uživatel nastavit, ale většinou se pohybuje okolo 10 Hz. Nekonzistence tabulky tedy nastává na pouze 100ms a pouze v okamžiku změny zobrazovaných proměnných. Když srovnáme tuto zanedbatelnou nekonzistenci s dosahovanou efektivitou zobrazovaní, jeví se toto řešení jako velmi elegantní.

8.6 Uživatelské nastavení aplikace

V aplikaci jsem kladl důraz na co největší možnost přizpůsobení aplikace, proto aplikace obsahuje mnoho možností nastavení. Všechna tato nastavení jsou přístupná z menu pod položkou „Settings“. Po jejich zobrazení je zobrazeno okno `OptionsPage`. Veškerá nastavení jsou přístupná z tohoto okna. Levá část okna obsahuje sloupeček obsahující kategorie nastavení. V pravé části okna jsou pak zobrazována nastavení příslušející konkrétní kategorii. Při změně kategorie v levém sloupci okamžitě dochází k zobrazení příslušných nastavení. Toto jsem řešil tak, že pro každou položku v levém sloupečku jsem vytvořil vlastní stránku, kterou jsem po kliknutí na příslušnou kategorií zobrazil. Abych docílil správného chování, bylo potřeba doplnit třídu tohoto okna o obslužný kód. Třída obsahuje strukturu `MySettings`, která pro každé zobrazované nastavení obsahuje proměnnou uchovávající hodnotu tohoto nastavení. Při zobrazení okna jsou hodnoty ze souboru nastavení (viz kapitola 8.7) načteny do této lokální struktury. Okna obsahující jednotlivá nastavení pak načítají a při změně i ukládají nastavení do této struktury. Po potvrzení nastavení kliknutím na tlačítko „Ok“ jsou nová data ze struktury nahrána zpět do konfiguračního souboru. Tuto techniku s využitím pomocné struktury využívám z toho důvodu, protože v případě, že bych tuto strukturu nevyužil a data nahrával přímo do konfiguračního souboru, by nebylo možné vzít změnu nastavení zpět tlačítkem „Cancel“. Při změně nastavení týkajícího se serveru, je při potvrzení změn tlačítkem „Ok“ provedeno porovnání nové hodnoty s předchozí a v případě změny je tato změna odeslána na server. Kromě potvrzovacího tlačítka „Ok“ a rušícího tlačítka „Cancel“ okno obsahuje ještě tlačítko „Default“, které nastaví

hodnoty do jejich defaultní hodnoty. Následující podkapitoly se budou podrobně věnovat jednotlivým zobrazovaným oknům nastavení.

8.6.1 Nastavení – Windows

První a nejjednodušší stránka slouží k nastavení animace přepínání mezi okny. Obsahuje posuvník umožňující plynulé nastavení rychlosti přechodu od 50 ms až do 1 s. Stejně tak je zde možné animované přechody mezi okny zcela vypnout. Grafická podoba hlavičky této stránky je vykreslována podle jednotného nastavení uloženého v souboru `App.xaml`. Stejným způsoben jsou vykreslovány i hlavičky dalších stránek s nastavením. Použitím jednotných stylů uložených na jednom místě jsem ušetřil opakované definování stylu hlavičky a v případě potřeby její změny stačí změnit pouze styly na tomto jednom místě a není třeba měnit každou stránku zvlášť. Princip práce s nastavením je u této stránky totožný s ostatními stránkami nastavení. Při zobrazení okna dojde k načtení hodnot nastavení z příslušných proměnných struktury `MySettings` třídy `OptionsPage`. V případě změny nastavení dojde k aktualizaci hodnoty v této struktuře. Samotné uložení nastavení je v režii třídy `OptionsPage` a závisí na konkrétní provedené události.

8.6.2 Nastavení – Network

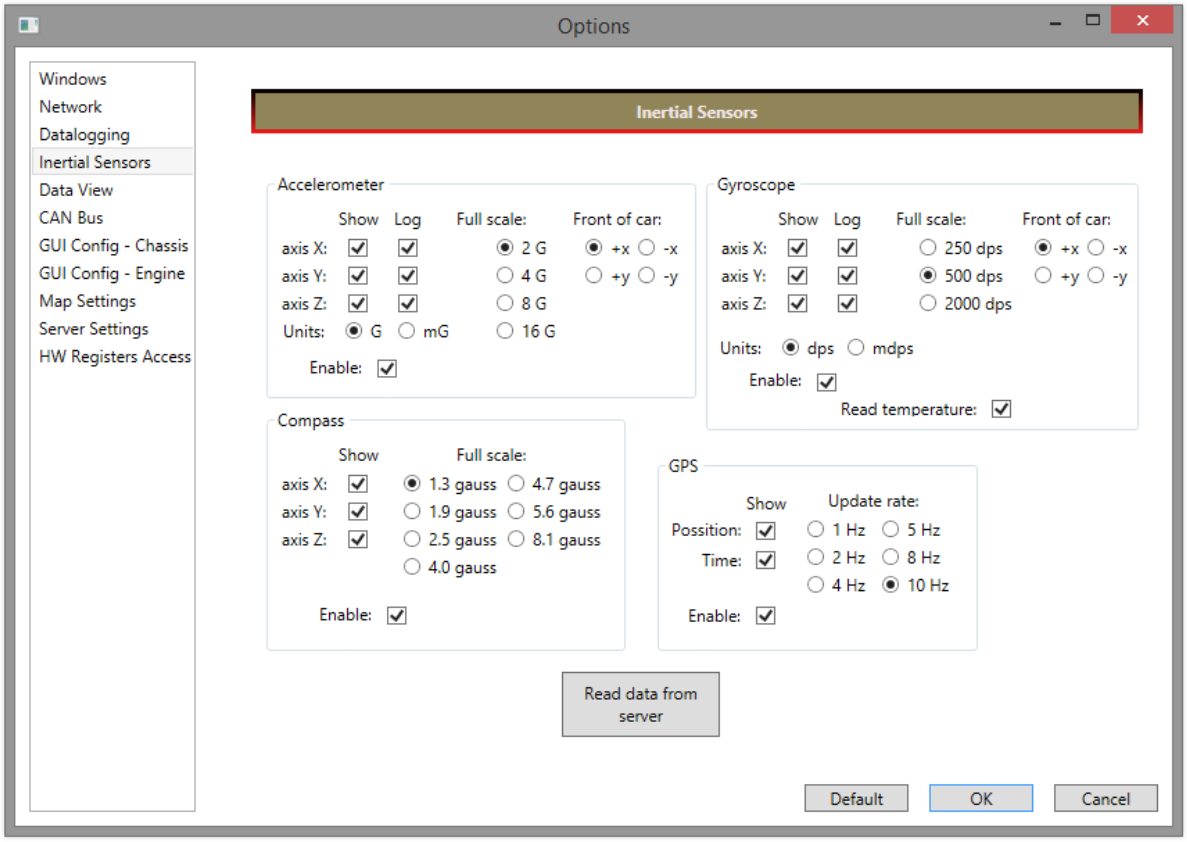
Tato stránka slouží k nastavení a ověření bezdrátového spojení se serverem. Před zobrazením této stránky jsou nejprve zjištěny IP adresy všech lokálních síťových rozhraní. Ty jsou následně zobrazeny v rozbalovacím menu. Uživatel může buď vybrat síťové rozhraní pro komunikaci se serverem z tohoto rozbalovacího menu, nebo zadat jeho IP adresu manuálně. Dále je nutné v tomto okně zadat IP adresu serveru a port pro komunikaci. Po nastavení všech údajů komunikaci je vhodné pomocí tlačítka „Check connection“ taktéž umístěného na této stránce ověřit správnost konfigurace. Po kliknutí na toto tlačítko dojde ke změně jeho barvy na šedou a otevře se okno `CheckWiFiConnectionWindow`. Toto okno ihned po spuštění vytvoří nové vlákno, ve kterém s příslušnými parametry zavolá metodu `SendOverTCP` třídy `WiFiFunction`. Tato metoda odešle pomocí protokolu TCP na server zprávu „Telemetry, are you alive?“ a očekává od serveru odpověď „Telemetry is ready!“. Pokud se nepodaří do určitého časového intervalu obdržet tuto odpověď, je zobrazena chybová zpráva. V opačném případě je zobrazena zpráva potvrzující fungující komunikaci mezi klientem a serverem. V případě, že ověření komunikace trvá dlouho a uživatel již nechce nadále čekat, je mu v okně `CheckWiFiConnectionWindow` nabídnuto tlačítko „Abort“, které okamžitě ukončí vlákno provádějící kontrolu spojení. Protože ale toto okno nemá přímý přístup k vláknu provádějícímu kontrolu připojení, řeším ukončení vlákna pomocí společné sdílené proměnné typu `CancellationTokenSource`. Běžící vlákno hlídá stav této proměnné a v případě zjištění jejího stavu typu „Cancel“ dojde k okamžitému ukončení jeho běhu.

8.6.3 Nastavení – datalogging

Stejně jako server tak i aplikace na klientském počítači umožňuje nahrávat a uchovávat získaná data ze senzorů formule. Tato stránka umožňuje povolení resp. zakázání tohoto logování. Výsledný soubor má příponu `*.log` a jeho umístění na disku lze nastavit z tohoto okna. Kromě nastavení cesty k tomuto souboru zde můžeme nastavit i jeho maximální velikost. V případě, že dojde k překročení této zadané velikosti, vytvoří se záloha s příponou `*.bck` předchozího logovacího souboru a je uživatelovi zobrazena informační zpráva. Logování ale pokračuje dál do nového souboru. Dalším důležitým nastavením této stránky je nastavení velikosti bufferu. Veškerá přijatá data se budou nejprve ukládat do tohoto bufferu, který se bude až do svého naplnění udržovat v paměti RAM. Po naplnění bufferu dojde k jeho uložení do logovacího souboru.

8.6.4 Nastavení – Inertial Sensors

Na této stránce je možné nastavit veškeré nastavení inerciálních senzorů i GPS modulu. Vzhled stránky je zobrazen na obr. č. 8.5. Nastavení pro každý senzor jsou rozděleny do jednotlivých bloků, což ulehčuje orientaci. Akcelerometr i gyroskop umožňují povolení či zakázání zobrazování nebo logování v jednotlivých osách. Pokud povolíme logování v konkrétní ose, znamená to, že tato možnost bude na serverové aplikaci i v hardwaru povolena a příslušná data na serveru ukládány. Pokud bychom povolili i zobrazování dat z konkrétní osy, budou tyto data zobrazeny a případně i ukládány v klientské aplikaci. Pokud budeme chtít data z určité osy zobrazovat, musíme pochopitelně povolit i jejich logování a stejně tak povolit i celý senzor. Toto je důležité připomenout, nicméně aplikace tyto situace řeší automaticky a uživatelovi není dovoleno zadat špatné nastavení. Modul kompasu neobsahuje možnost pro povolení resp. zakázání logování v jednotlivých osách, protože toto nastavení není samotným hardwarem podporováno. Možnost nastavení zobrazování dat z jednotlivých os zde ale zůstává. Stejně tak u GPS modulu není žádné nastavení os, místo toho je zde volba, zda chceme číst a zobrazovat informace o pozici nebo čase, příp. oboje. Každý ze senzorů obsahuje i nastavení svého rozsahu. Změna tohoto nastavení je automaticky provedena také v samotném hardwaru, nutností je ale funkční komunikace mezi klientem a serverem. Umístění telemetrie se na voze může měnit, proto je blok pro akcelerometr i gyroskop doplněn nastavením osy směřující ve směru jízdy. V případě špatného nastavení těchto os by při zobrazení docházelo k zobrazování naměřených dat ve špatné ose. Jak již bylo zmíněno dříve, modul gyroskopu obsahuje i vestavěný teploměr, proto i v nastavení je možnost pro povolování čtení teploty z tohoto senzoru.



Obr. č. 8.5 Stránka pro změnu nastavení inerciální senzorů

Stejně jako na předchozích stránkách s nastavením tak i na této stránce dochází k načtení hodnot nastavení automaticky při zobrazení stránky. V tomto případě je ale stránka navíc doplněna tlačítkem pro přečtení aktuálních dat ze serveru, tedy i ze samotného hardwaru. Tento přístup jsem zvolil z důvodu, že se jedná o citlivé nastavení a uživatel musí mít možnost ověřit si, zda nastavení uložené v klientské aplikaci opravdu odpovídá nastavení serverové aplikace a hardwaru. K získání těchto hodnot ze serveru nedochází automaticky, ale až po stisknutí zmiňovaného tlačítka. Důvodem k tomuto řešení bylo zrychlení odezvy aplikace při nechtěném překlikutí při výběru stránky s nastavením, kdy aplikace nemusí navazovat spojení se serverem a zjišťovat hodnoty všech nastavení, ale pouze zobrazí nastavení uložené v klientské aplikaci.

8.6.5 Nastavení – DataView

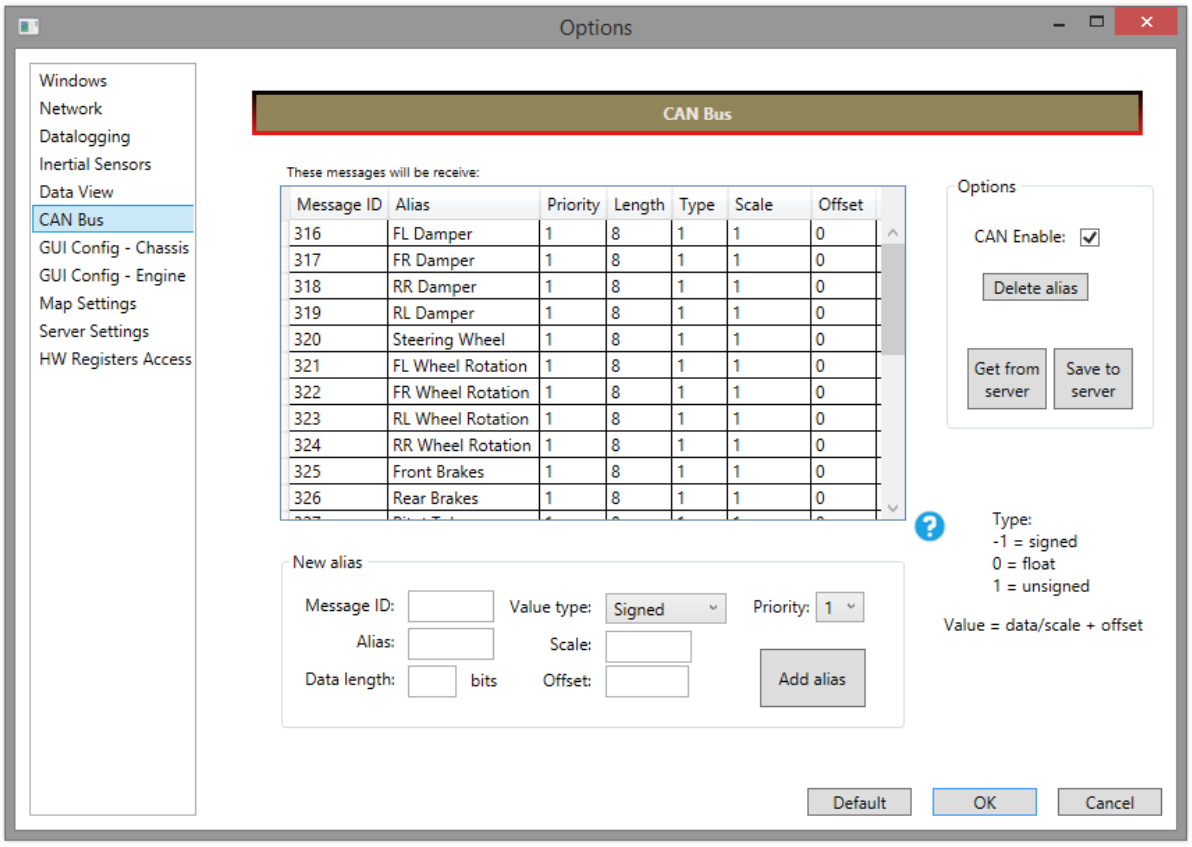
Tato stránka obsahuje nastavení pro zobrazení dat v tabulce `DataView` popsané v kapitole 8.5. Můžeme zde nastavit rychlost obnovování dat v ms, která je omezena nejnižší hodnotou 100 ms, tzn. 10 změn na sekundu. Nižší hodnota by již pouze nadměrně zatěžovala procesor a výsledná data by při takto rychlých změnách stejně nebyla čitelná. Dalším nastavením je zde počet zobrazovaných desetinných míst. Jedná se opravdu pouze o nastavení zobrazování v okně `DataView` to znamená, že i při nastavení zobrazování celých čísel (0 desetinných míst) budou data v okně zobrazujícím stav senzorů motoru i šasi zobrazena stále stejně s předem daným počtem desetinných míst. Poslední možností nastavení je způsob zobrazování CAN zpráv. Můžeme je zobrazovat buď pomocí jejich

identifikátoru, nebo pomocí jejich názvu. Případně zde můžeme zobrazování CAN zpráv úplně zakázat. V případě zobrazování CAN zpráv pomocí jejich identifikátoru je hodnota ID zobrazována v dekadické soustavě.

8.6.6 Nastavení – CAN Bus

Tato stránka slouží k nastavení přijímaných CAN zpráv klienta i serveru. Hlavním prvkem stránky je, jak je zobrazeno na obr. č. 8.6, tabulka zobrazující všechny CAN zprávy, které budou serverem vyžadovány a následně přijímány. Tyto zprávy jsou uloženy v lokální kolekci `Globals.canAliases`, do které jsou uloženy ihned po spuštění programu načtením z XML souboru CAN-aliases.xml umístěného ve složce programu. Po ukončení práce na této stránce je aktuální kolekce do tohoto XML souboru opět uložena. Obě tyto operace, stejně tak jako další metody a třídy pro práce s CAN zprávami, jsou obsaženy v třídě `CANClass`.

První sloupec tabulky zobrazuje ID zprávy v dekadické podobě. Jak již bylo popsáno v kapitole 7.1, vyvinutá telemetrie pracuje jak se standardním rámcem CAN zpráv, tak i s rozšířeným, proto ID zprávy může nabývat až 29bitových hodnot. Druhý sloupec zobrazuje přidělený alias zprávy a je důležitý pro propojení zprávy s konkrétním ID s jejím významem. Podle těchto aliasů dochází k zobrazování dat v grafickém rozhraní. Zobrazení všech aliasů CAN zpráv, se kterými program pracuje a zobrazuje je ve speciální podobě (například natočení kol apod.), je možné pomocí ikony otazníku umístěné v pravém dolním rohu tabulky. Po kliknutí na tuto ikonu je zobrazeno nové okno `RecommendedAliases`, ve kterém jsou všechny tyto aliasy zobrazeny. Toto okno čerpá data do tabulky ze statické kolekce `recommendedAliases`. Tato kolekce je určena pouze pro čtení a je definovaná v třídě `Globals`.



Obr. č. 8.6 Stránka pro změnu nastavení sběrnice CAN

Pokud se vrátíme zpět k popisu nastavení CAN zpráv, tak kromě ID zprávy a jejího aliasu tabulka obsahuje také prioritu zprávy, která se může pohybovat od 1 do 3. Práce s prioritami je popsána v kapitole 7.4.5. Další sloupec udává délku zprávy v bitech. Většinou je délka zprávy 8 bitů tedy 1 Byte, ale může se objevit i zpráva s délkou více Bytů, nebo dokonce zpráva s délkou necelých Bytů, například pouze 4 bity a podobně. Se všemi těmito stavy aplikace počítá a délku je možné nastavit v bitech až do hodnoty 64 bitů, což je podle definice protokolu CAN maximální možná velikost dat odesílaných v jedné CAN zprávě. Data obsažená v CAN zprávě mohou být různého typu. Pro takový případ slouží sloupec s označení „Type“, ve kterém je zobrazeno číslo vyjadřující typ zprávy, jak je zobrazeno v tabulce 8.1.

Zástupné číslo	Typ dat
-1	Číslo se znaménkem
0	Číslo s plovoucí desetinnou čárkou
1	Číslo bez znaménka

Tabulka 8.1 Typy dat přenášených zpráv

Poslední dva sloupce obsahují měřítko a offset dat. Tyto dvě hodnoty nám umožňují převést přijatá data do správného rozsahu. Podrobnější popis zpracování dat různých typů a práce s měřítkem a offsetem je popsána v kapitole 8.1.

Zobrazenou tabulku s CAN zprávami můžeme přímo editovat případně po označení konkrétního řádku tabulky a stisknutí tlačítka „Delete alias“ tento řádek nenávratně smazat.

Pro odesílání nastavených zpráv na server slouží tlačítko na pravé straně „Save to server“, které pro každou prioritu vytvoří zprávu se všemi identifikátory CAN zpráv konkrétní priority a odešle na server. Ve výsledku se tedy provede odeslání 3 zpráv. Až na kontinuální zobrazování dat senzorů ze serveru probíhá veškerá komunikace přes TCP protokol. Proto i v tomto případě probíhá odeslání zpráv přes TCP protokol a je vyžadováno potvrzení od serveru o příjmu všech zpráv. Nemůže tedy dojít k situaci, kdy uživatel neví, zda došlo k úspěšnému příjmu nastavení serverem, protože ve všech případech je o výsledku provádění nastavení informován vyskakovacím oknem.

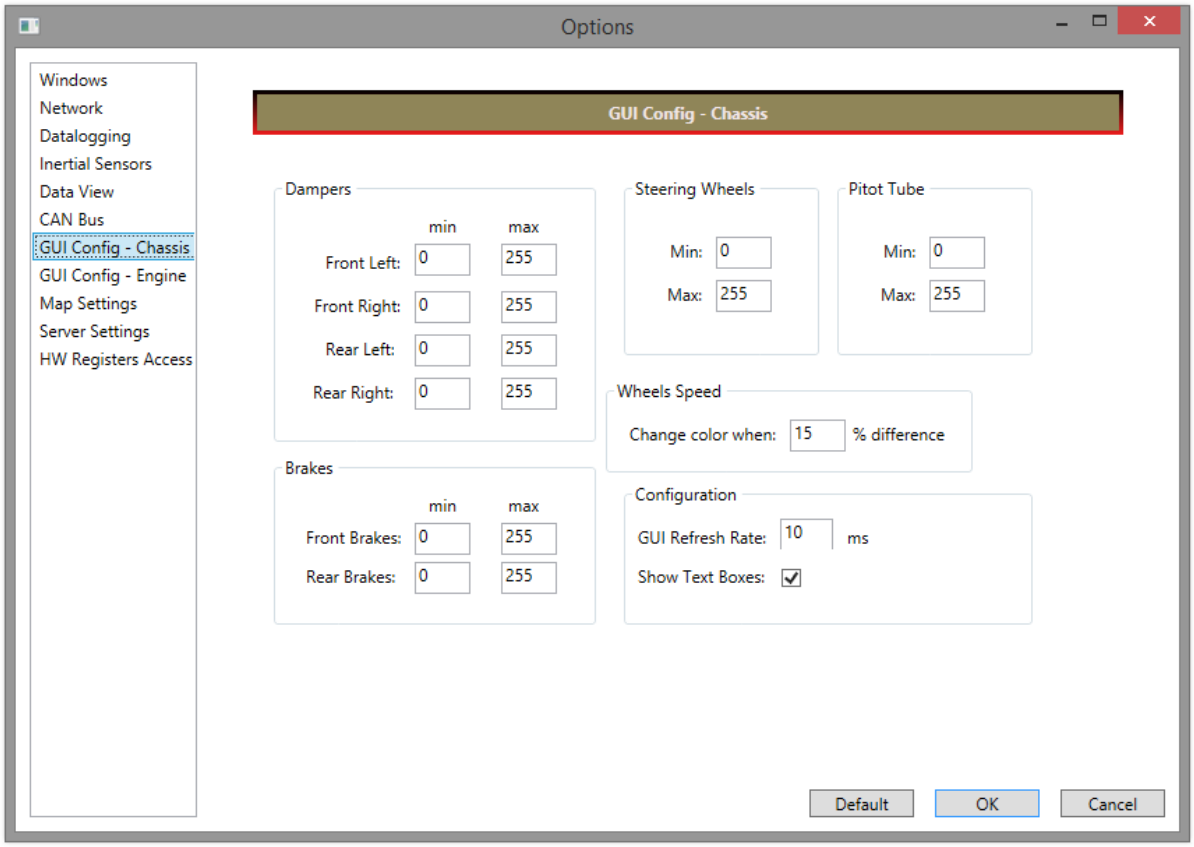
Příjem nastavení CAN zpráv ze serveru je poněkud složitější. Na serveru jsou uloženy pouze identifikátory zpráv a jejich priority. Žádné další informace na serveru nejsou, proto při příjmu dat ze serveru dojde v tabulce k zobrazení hodnot na serveru, ale bez doplňujících informací jako jsou například délka dat, alias a podobně. Proto, aby nedošlo k přepsání nastavení v klientské aplikaci, je při příjmu dat vytvořená nová kolekce dat, která je zobrazena v tabulce. Lokálně uložená kolekce dat není změněna až do doby, kdy změnu lokální kolekce za vzdálenou kolekci uživatel potvrdí. K tomuto potvrzení je vyzván po ukončení stránky s tímto nastavením.

Pro vkládání nových dat do tabulky slouží blok umístěný ve spodní části stránky. Po vyplnění všech údajů týkajících se přidávané CAN zprávy jsou nejprve všechny tyto hodnoty zkontrolovány, zda neobsahují chybu. Následně je provedena kontrola, zda tabulka již neobsahuje zprávu se stejným identifikátorem, jako je identifikátor nové zprávy, protože zprávy s duplicitním ID nejsou povoleny. Po obou těchto kontrolách dojde k přidání zprávy do aktuální kolekce zobrazené v tabulce, kdy se může jednat o kolekci získanou ze serveru i o lokální kolekci.

8.6.7 Nastavení – GUI Config – Chassis

Další stránka se týká nastavení zobrazování dat v okně TopView, kdy jsou získaná data zobrazována na šasi formule. Toto okno je popsáno v kapitole 8.2. Všechna nastavení jsou opět rozdělena do logických bloků, jak je zobrazeno na obr. č. 8.7. Tyto bloky nastavují tlumiče, volant, Pitotovu trubici, brzdy, rychlost otáčení kol a poslední blok obsahuje obecná nastavení. Mezi tyto nastavení patří nastavení rychlosti aktualizace dat v grafickém rozhraní a také zatržítko umožňující povolení nebo zakázání zobrazení vedle grafických ukazatelů také číselných hodnot. Předchozí bloky obsahují jednotlivá data a u nich nastavení minimální a maximální hodnoty. Tímto dosáhneme zobrazování dat v celém svém rozsahu. Příkladem mohou být například potenciometrické snímače zmáčknutí tlumiče. Samotné snímače mají rozsah například 0 až 100, ale tlumič se může pohybovat pouze v rozsahu například 15 až 80. Proto se do příslušných políček v této nastavovací stránce zadají tyto hodnoty a v grafickém rozhraní pak bude pohyb tlumiče zobrazen ve svém plném rozsahu, kdy minimální hodnotě bude odpovídat hodnota 15 a maximální hodnotě hodnota 80. Zbytek stupnice bude

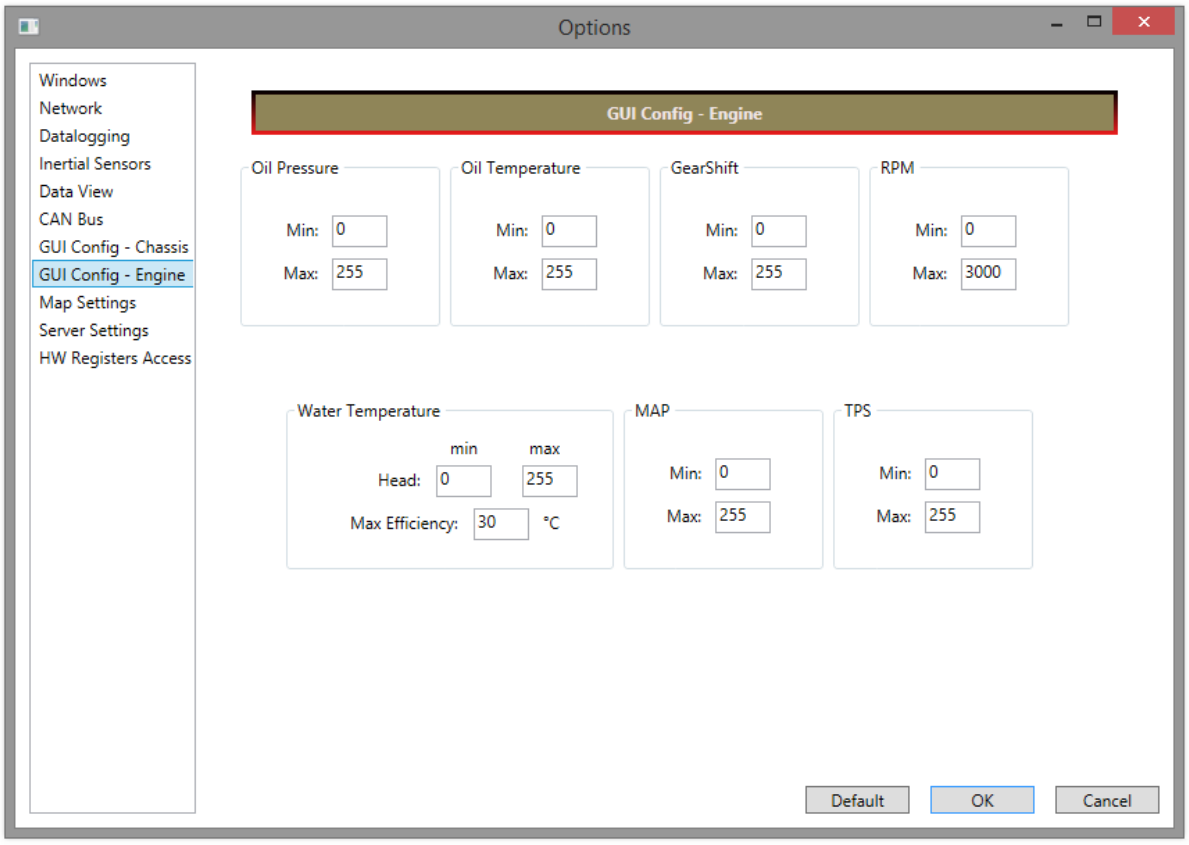
přepočítán na tento rozsah. Blok pro rychlost otáčení kol obsahuje nastavení definující hodnotu rozdílu otáčení jednoho kola oproti ostatním, při kterém dojde v grafickém rozhraní ke změně barvy ukazatele prokluzu.



Obr. č. 8.7 Stránka pro změnu nastavením GUI pro šasi

8.6.8 Nastavení – GUI Config – Engine

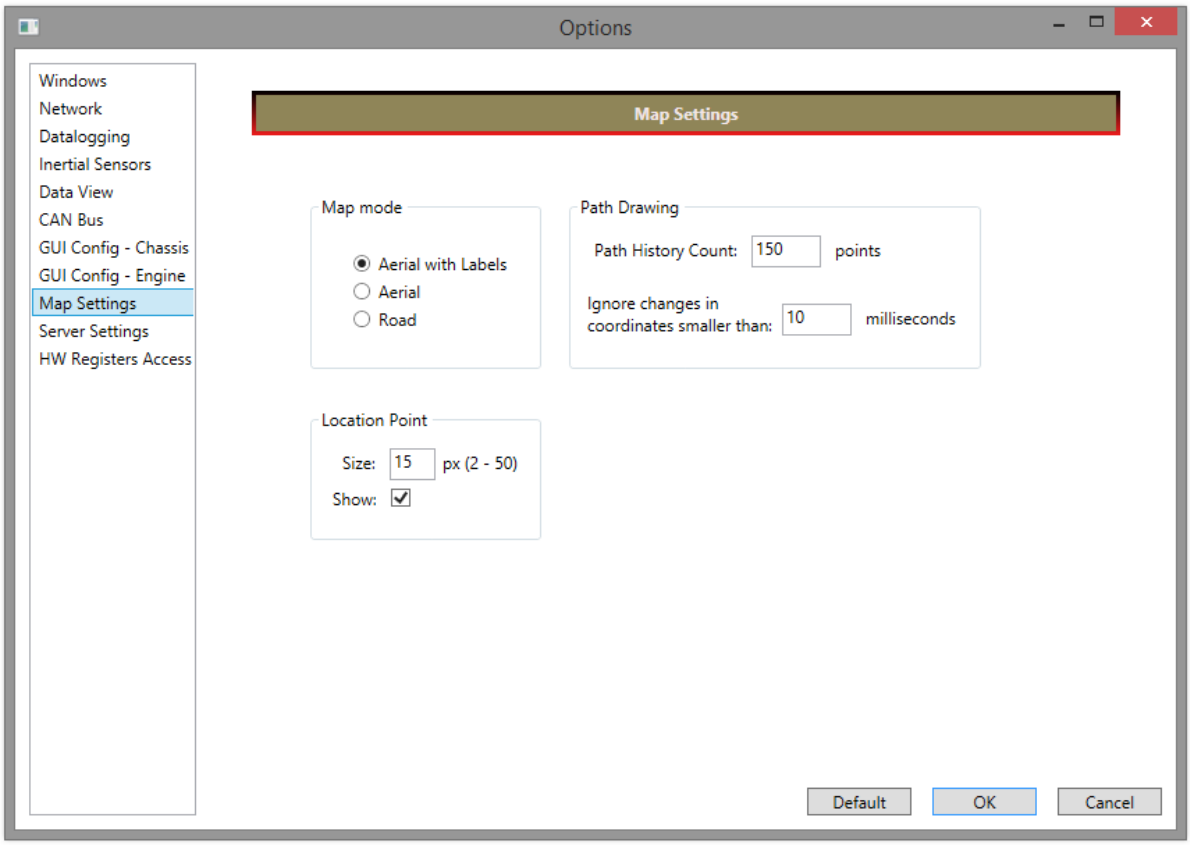
Tato stránka je velmi podobná předchozí stránce pro nastavení zobrazování dat na šasi formule s tím rozdílem, že tato stránka slouží pro nastavení zobrazování dat na motoru formule v okně `EngineView` popsaného v kapitole 8.3. Opět jsou nastavení rozdělena do logických bloků. Tentokrát jsou to bloky pro nastavení tlaku oleje, teploty oleje, zařazené rychlosti, otáček, teploty chladící kapaliny, MAP senzoru a natočení škrtící klapky. Všechny tyto bloky obsahují políčka pro nastavení minimálních a maximálních hodnot. Blok pro zobrazování teploty chladící kapaliny navíc obsahuje možnost nastavení maximální efektivity chlazení chladiče ve stupních Celsia. Jedná se vlastně o maximální rozsah ukazatele efektivity chlazení. Tato stránka je zobrazena na obr. č. 8.8.



Obr. č. 8.8 Stránka pro změnu nastavení GUI pro motor

8.6.9 Nastavení – Map Settings

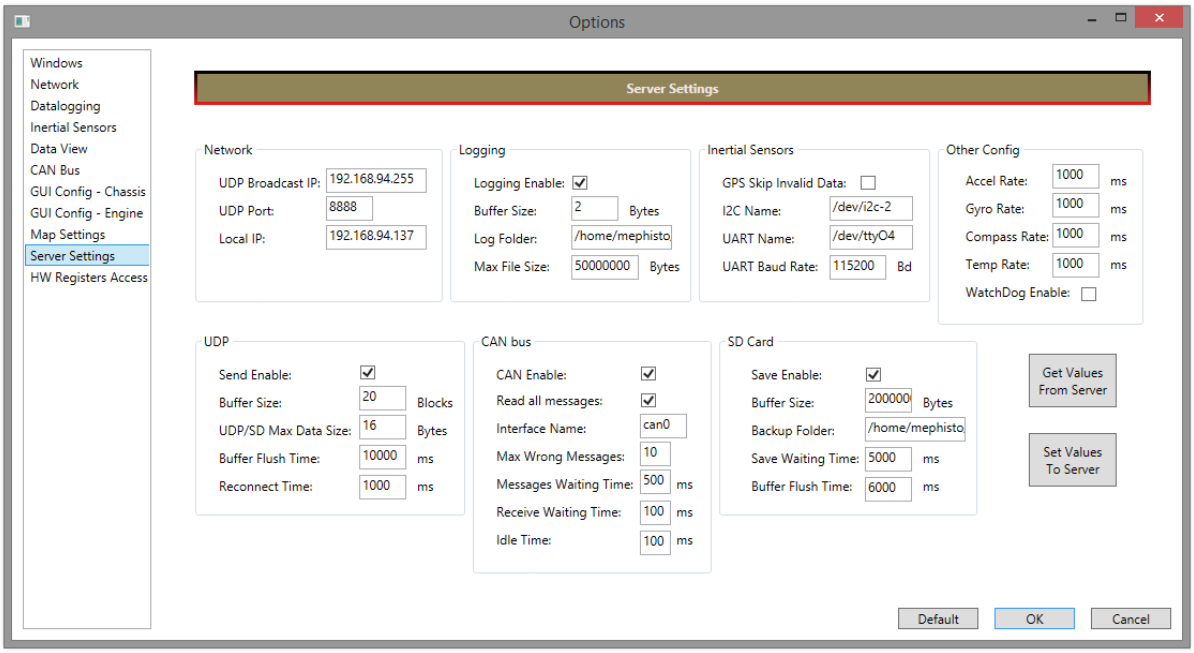
Tato stránka zobrazená na obr. č. 8.9 nastavuje vzhled mapy a vykreslování trasy. První blok obsahuje nastavení mapových podkladů. Může se jednat o letecký pohled s popiskami nebo bez, nebo obecnou mapu. Ve všech případech jsou použity mapové podklady Bing, jak je rozebráno v kapitole 8.4. Další blok se týká vykreslování trasy. Zde lze nastavit velikost bufferu pro ukládání historie bodů. Tyto body jsou následně spojovány a vykreslovány jako projetá trasa. Aby nedocházelo k ukládání bodů, které jsou u sebe příliš blízko, je zde možnost i nastavit minimální vzdálenost nového bodu od předchozího uloženého. Proto například když formule stojí na místě je uložen pouze jeden bod s polohou. Pokud se formule začne pohybovat, začnou se body ukládat s nastavenou četností. Tato hodnota se zadává v milisekundách zeměpisné šířky a délky a doporučená hodnota je 10 ms. Zadání této hodnoty přímo v metrech není možné, protože převod zeměpisné šířky nebo délky na metry se liší podle aktuální polohy na zeměkouli, a proto není lineární. V posledním bloku je uživateli umožněno nastavit zobrazení červeného bodu indikující aktuální polohu na mapě a jeho rozměru. Tento rozměr je omezen na hodnoty od 2 do 50 pixelů, což je vhodná velikost bodu. V případě zadání hodnoty mimo rozsah je nastavena nejbližší povolená hodnota.



Obr. č. 8.9 Stránka pro změnu nastavení zobrazení map

8.6.10 Nastavení – Server Settings

Tato stránka slouží k nastavení samotné aplikace běžící na serveru. Změna nastavení na této stránce může způsobit nefunkčnost celé aplikace, proto je tato stránka určena pouze zkušeným uživatelům. Před samotným zobrazením stránky je uživateli zobrazeno varovné okno upozorňující na tuto skutečnost. Nastavení na této stránce nejsou uložena lokálně, proto po potvrzení varovné zprávy jsou ze serveru všechna tato nastavení načtena a následně zobrazena viz obr. č. 8.10. Uložení nové konfigurace na server příp. její opětovné načtení se provádí tlačítky v pravém dolním rohu stránky.



Obr. č. 8.10 Stránka pro změnu nastavení serveru

Veškerá nastavení serverové aplikace jsou přístupná z této stránky. Je zde možné nastavit parametry síťového přenosu jako je broadcastová IP adresa, lokální IP adresa a komunikační port. S bezdrátovým přenosem souvisí i blok „UDP“. V tomto bloku je k dispozici povolení samotného UDP přenosu pro odesílání dat ze senzorů i maximální velikost jedné zaznamenané zprávy. Toto nastavení je zde důvodu případného budoucího rozšíření aplikace o další zprávy, které by mohly zabírat více než 16 Bytů, nicméně v současné konfiguraci musí toto číslo být 16 Bytů. Odesílání dat přes protokol UDP neprobíhá po jednotlivých zprávách, ale po bloku zpráv, jak bylo vysvětleno v kapitole 7.4.7. Velikost odesílaného bloku stejně tak jako nejdelší interval mezi jednotlivými odesláními je zde možné také nastavit. Pokud se serverové aplikaci nepodaří navázat spojení s žádným klientem, nebo navazování spojení selže, je za čas definovaný v posledním políčku bloku „UDP“ zkoušeno znova navázat spojení.

Blok „Logging“, jak již sám jeho název napovídá, obsahuje nastavení týkající se logování dat. Je nutné podotknout, že na rozdíl od klientské aplikace je zde pod pojmem logování myšleno logování stavu aplikace, ze kterého je v případě chyby možné vyčíst příčinu problému. Více o logování stavu serverové aplikace je popsáno v kapitole 7.4.8. V tomto bloku je možné povolit nebo zakázat logování, nastavit umístění logovací souboru, jeho maximální velikost a v neposlední řadě i velikost bufferu, který se bude do souboru zapisovat.

Blok „Inertial Sensors“ obsahuje nastavení komunikačních sběrnic s inerciálními senzory. Kromě zaškrtávacího okénka definujícího zda se mají údaje o pozici z GPS modulu ignorovat, pokud není potvrzena jejich správnost, nedoporučuji tyto nastavení měnit bez patřičné změny systému serveru.

Naproti tomu nastavení v bloku „Other Config“ je možné měnit dle libosti. Tento blok obsahuje nastavení frekvence odesílání dat z jednotlivých inerciálních senzorů. Navíc je v tomto bloku

možnost povolení nebo zakázání hlídání funkčnosti aplikace tzv. Watchdogem. Princip Watchdogu je podrobně popsán v kapitole 7.4.2.

V bloku CAN Bus můžeme nastavit, zda chceme dostávat CAN zprávy, příp. jestli všechny nebo pouze vyžádané. Kromě nastavení jména rozhraní, které opět nedoporučuji bez změny HW měnit, je zde možné měnit parametry samotného získávání CAN zpráv. Nastavení „Max Wrong Messages" značí maximální počet přijatých chybných odpovědí, než je dotazování na konkrétní zprávu vzdáno. Podobně nastavení „Messages Waiting Time" značí maximální dobu po jakou čekat na správnou odpověď. Velmi podobné nastavení „Receive Waiting Time" nastavuje maximální dobu, po jakou čekat na jakoukoliv odpověď. Tato doba je menší než doba „Messages Waiting Time" a v případě, že CAN sběrnice neodpovídá po tuto dobu, není nutné čekat celý čas vyhrazený pro příjem zpráv, ale příjem může být ukončen. Posledním nastavením tohoto bloku je „Idle Time", což je vlastně frekvence odesílání správných odpovědí na klienta.

Posledním blokem této stránky je blok SD Card. Zde je možné nastavit ukládání získaných dat ze senzorů do souboru pro jejich následné stažení a použití. Jak již bylo rozebráno v kapitole 7.4.4 je toto průběžné ukládání dat vhodné pro situace, kdy dojde ke ztrátě spojení mezi serverem a klientem a data proto nejsou klientem zaznamenávána. Opět je zde možné toto ukládání povolit či zakázat i nastavit cestu k výslednému souboru. Pozornost je ale nutné věnovat nastavení velikosti ukládajícího bufferu. Vzhledem k ukládání na paměť typu Flash, která má omezený počet zápisů je nutné tuto hodnotu volit s ohledem na tento fakt. Já jsem zvolil hodnotu bufferu 200 000 B (necelé 2 kB). Výpočet jsem prováděl podle rovnic č. 8.5 a č. 8.6 získaných z [41].

$$lifetime = 2000000 * \frac{C - Cs - Block}{Fs * Fr} \; [s]$$

Rovnice 8.5 Obecná rovnice pro výpočet životnosti paměti typu Flash

Kde: C – kapacita paměti (BeagleBone Black má paměť 2 GB)
Cs – kapacita využívaná statickými soubory - systémem (~ 1 GB)
Block – velikost 1 bloku (pro konkrétní paměť použitou v BBB je to 256 KB)
Fs – velikost zapisovaného bloku (zvolená hodnota 200 000 B)
Fr – frekvence zápisu (liší se v závislosti na typu zpráv, ale předpokládaný bitrate je 960 B/s tzn. frekvence zápisu bloků o velikosti 200 000 B je $\frac{200000}{960} = 208{,}33\ s = 3{,}47\ min$, tedy zhruba 18 zápisu za hodinu.

Po dosazení do vzorce

$$lifetime = 2000000 * \frac{2048 - 1024 - 0{,}25}{0{,}1907 * 18} \cong 68092\ let$$

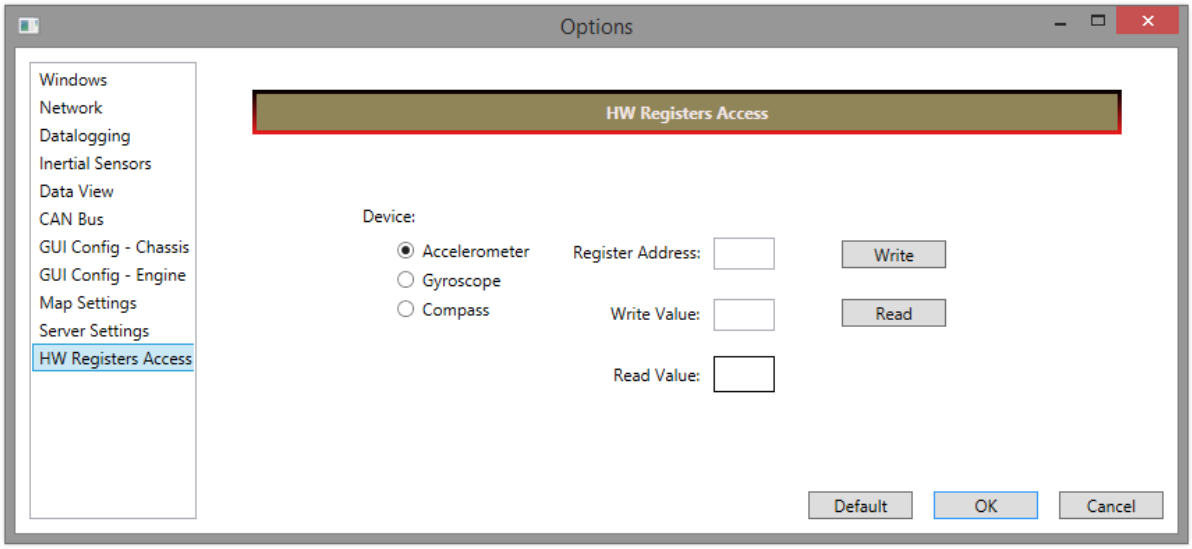
Rovnice 8.6 Výpočet životnosti vestavěné paměti typu Flash

Z výsledku je vidět, že pro toto malé množství ukládaných dat a relativně velkou kapacitu paměti je zvolená hodnota bufferu správná a nezkracuje nám výrazným způsobem životnost paměti.

V tomto bloku je také možné nastavit „Buffer Flush Time“, tedy maximální dobu po které je buffer uložen do souboru bez ohledu na jeho zaplnění.

8.6.11 Nastavení – HW Registers Access

Poslední stránka je zobrazena na obr. č. 8.11 a umožňuje přímý přístup do registrů inerciálních senzorů. Zápisem dat do určitých registrů může dojít k poškození samotného senzoru, proto je tato stránka určena pouze zkušeným uživatelům a stejně jako v předchozí stránce je před jejím zobrazením zobrazena varovná zpráva upozorňující na tyto rizika.



Obr. č. 8.11 Stránka pro přístup k registrům inerciálních senzorů

Po vybrání konkrétního senzoru a zadání adresy registru je po stisknutí tlačítka „Read“ přečtena a zobrazena hodnota z příslušného registru. Můžeme i zapisovat libovolné hodnoty do registrů vyplněním políčka „Write Value“ a stiskem tlačítka „Write“. Nutno podotknout, že ve všech políčkách, stejně tak jako v celé aplikaci pracuji s čísly v dekadické soustavě, nikoliv v hexadecimální jak by možná uživatel mohl očekávat. Práci s čísly v dekadické soustavě jsem zvolil z toho důvodu, že mi přijde vhodnější a lépe se mi s takovými daty pracuje. Význam hodnot jednotlivých registrů je dostupný v příslušných technických dokumentacích [33][34].

8.7 Soubory nastavení

Pro ukládání aplikačních a uživatelských nastavení jazyk C# používá sady Settings. V těchto sadách jsou uloženy kromě jména a hodnoty proměnné také její typ a rozsah („Scope“). Rozsah dělí proměnné na aplikační a uživatelské. Rozdíl mezi těmito dvěma typy je ten, že aplikační proměnné jsou určeny pouze pro čtení a jejich hodnota se dá měnit pouze při návrhu. Naproti tomu uživatelské proměnné mohou být libovolně měněny při běhu aplikace. Po ukončení a opětovném spuštění aplikace jsou tyto proměnné opět přístupné. To je docíleno ukládáním těchto nastavení do konfiguračního souboru `user.config` umístěného v systémové složce každého uživatele. V popisované aplikaci jsem jednotlivá nastavení rozdělil do 3 sad. Hlavní sadou je sada `Settings.settings`, která obsahuje všechna nastavení aplikace, které je možné v programu nastavit. Pro přehlednost ale tato sada neobsahuje nastavení týkající se zobrazování dat v grafickém rozhraní, které jsou uloženy v sadě `UIConfig.settings`. Mezi tyto nastavení patří nastavení ze stránek GUI Config – Engine (kapitola 8.6.8) a GUI Config – Chassis (kapitola 8.6.7). Poslední sadou je sada `DefaultValues.settings`. Tato sada jako jediná neobsahuje uživatelské nastavení ale aplikační a obsahuje pevně definované nastavení. Tyto nastavení jsou používány jako zdroj výchozích hodnot. Při stisknutí tlačítka „Default“ po zobrazení okna `OptionsPage` s nastaveními jsou data z této sady zobrazeny v okně s nastavením a uživatel následně může nastavení aktualizované výchozími hodnotami uložit.

8.8 Metriky

Tato kapitola obsahuje metriky klientské aplikace, které jsou zobrazeny v tabulce 8.2 a 8.3. Tabulka 8.4 pak obsahuje součet metrik serverové i klientské aplikace včetně všech vytvořených knihoven.

	Název souboru	**Řádků**	**Metod**	**Řádků XAML soub.**
Properties	AssemblyInfo.cs	56	0	
	DefaultValues.settings		49 pol.	
	Settings.setings		54 pol.	
	UIConfig.settings		32 pol.	
MainPages	DataViewPage.xaml.cs	54	3	21
	EngineView.xaml.cs	215	3	177
	MapPage.xaml.cs	140	5	29
	TopView.xaml.cs	330	3	343

Tabulka 8.2 Metriky klientské aplikace

	Název souboru	Řádků	Metod	Řádků XAML soub.
OptionsPages	CAN BusPage.xaml.cs	295	12	73
	Data ViewPage.xaml.cs	99	7	31
	DataloggingPage.xaml.cs	88	6	26
	GUI Config - EnginePage.xaml.cs	192	17	77
	GUI Config - ChassisPage.xaml.cs	230	21	81
	HW Registers AccessPage.xaml.cs	181	6	28
	CheckWiFiConnectionWindow.xaml.cs	103	4	11
	Inertial SensorPage.xaml.cs	945	59	121
	Map SettingsPage.xaml.cs	114	9	43
	NetworkPage.xaml.cs	195	9	32
	PleaseWait.xaml.cs	28	1	10
	RecommendedAliases.xaml.cs	34	2	17
	Server SettingsPage.xaml.cs	552	8	133
	WindowsPage.xaml.cs	51	3	21
	About.xaml.cs	33	2	13
	App.xaml.cs	18	0	63
	MainWindow.xaml.cs	179	16	117
	OptionsPage.xaml.cs	573	6	27
Třídy	CANClass.cs	185	14	
	DataViewClass.cs	349	9	
	Delegates.cs	754	28	
	Functions.cs	283	8	
	Globals.cs	181	0	
	LoggingClass.cs	659	6	
	TransitionClass.cs	219	11	
	WiFiFunctions.cs	259	4	

Tabulka 8.3 Metriky klientské aplikace

	Řádků	Metod
Knihovny	918	29
Serverová aplikace	2 879	78
Klientská aplikace v třídách	8 042	417
Klientská aplikace v XAML souborech	1 494	-
CELKEM	**13 333**	**524**

Tabulka 8.4 Souhrnné metriky veškerého napsaného kódu

9 Závěr

V rámci této diplomové práce bylo vytvořeno telemetrické zařízení, které umožňuje bezdrátové stahování dat z inerciálních senzorů a CAN sběrnice formule Dragon IV. Nejprve jsem se seznámil se studentskou soutěží Formula Student a závodním monopostem Dragon IV vyvíjeném skupinou TU Brno Racing právě pro tuto soutěž. Dále bylo nezbytné, abych si prostudoval problematiku použití telemetrie v závodních monopostech, kde jsem vycházel především ze závodů Formule 1. Před samotným vývojem telemetrie jsem musel dále nastudovat problematiku měření fyzikálních veličin v automobilu, používaných senzorech, komunikaci mezi nimi i komunikaci se sběrnicí CAN. Telemetrie využívá vlastní inerciální senzory a GPS modul, proto jsem se musel zjistit možnosti použití těchto senzorů a jejich funkci. Po získání teoretických znalostí v této problematice jsem přistoupil k samotnému návrhu hardwaru telemetrie, zvolení vhodných komponent a napsání obslužného programu jak pro hardware na straně serveru, tak i pro řídící počítač na straně klienta. Vývoj hardwaru jsem prováděl zároveň s vývojem softwaru, protože jejich vzájemná komunikace je velmi důležitá pro správnou funkčnost celého zařízení.

Podařilo se mi vyvinout funkční telemetrické zařízení včetně hardwaru a řídícího programu. Po úspěšném otestování hardwaru jsem nechal vyrobit metodou 3D tisku krabičku přesně pro navržený hardware, kterou jsem společně se zařízením umístil na formuli Dragon IV a provedl další testování v reálném prostředí. Veškeré testy byly úspěšné a po doladění chyb pracuje zařízení podle představ. Jedná se o kompletní telemetrický systém, který je nachystán pro koncové zákazníky a šlo by jej komerčně prodávat. Vyvinuté zařízení bude umístěno na formuli Dragon IV a použito při závodech v Maďarsku na okruhu Györ, německém Hockenheimringu a také v posledním závodě letošní sezóny v Hradci Králové.

Vývoj hardwaru tohoto zařízení bude pokračovat dále a to rozšířením zařízení o další moduly, které budou umožňovat použití i v jiných aplikacích. Konkrétně bych chtěl zařízení rozšířit o možnost použití jako sledovacího zařízení při závodech plachetnic. K zařízení by byl doplněn modul obsahující SIM kartu, který by pomocí technologie GSM odesílal data na webový server, kde by docházelo k jejich ukládání i zobrazování. Společně s touto změnou hardwaru by prošel dalším vývojem i software, který by byl rozšířen o webovou službu umožňující sledovat uživatelům aktuální data na internetu. Tím by uživatelé internetu mohli online sledovat pozici jednotlivých plachetnic v závodě opět včetně dat z jejich senzorů, neboť moderní plachetnice využívají také CAN sběrnici.

Literatura

[1] INSTITUTION OF MECHANICAL ENGINEERS. *Formula Student* [online]. London, 2013 [cit. 2013-12-15]. Dostupné z: http://www.formulastudent.com

[2] SAE INTERNATIONAL. *2014 Formula SAE© Rules* [online]. 2013 [cit. 15.12.2013]. Dostupné z: http://students.sae.org/cds/formulaseries/rules/2014_fsae_rules.pdf

[3] Formula Student Germany: Concept. FORMULA STUDENT GERMANY. *Formula Student Germany* [online]. 2013 [cit. 2013-12-18]. Dostupné z: http://www.formulastudent.de/fsg/about/concept/

[4] TU BRNO RACING. *TU Brno Racing - novinky* [online]. 2010, 2013 [cit. 2013-12-16]. Dostupné z: http://www.tubrnoracing.cz

[5] ALLEN, Gale. MINNESOTA STATE UNIVERSITY. *An Introduction to Telemetry* [online]. 2013, 11 s [cit. 2013-12-16]. Dostupné z: http://mavdisk.mnsu.edu/alleng/communications/DataRadio/p_telemetry.pdf

[6] Telemetry. *Formula 1 Dictionary* [online]. 2013 [cit. 2013-12-16]. Dostupné z: http://www.formula1-dictionary.net/telemetry.html

[7] Pokud by volant zaměstnával jezdce moc, pomoci může telemetrie - články. PASTOREK, Jan. *F1Sports.cz - podrobné zpravodajství ze světa F1* [online]. 2011, 21.2.2011 [cit. 2013-12-16]. Dostupné z: http://f1sport.autorevue.cz/pokud-by-volant-zamestnaval-jezdce-moc-pomoci-muze-telemetrie-26116

[8] ATLAS - Software. MCLAREN ELECTRONIC SYSTEMS. *McLaren Electronic* [online]. 2013 [cit. 2013-12-16]. Dostupné z: http://www.mclarenelectronics.com/Products/Product/ATLAS

[9] BEEBY, Stephen. *MEMS mechanical sensors*. Boston: Artech House, 2004, 269 s. ISBN 15-805-3536-4.

[10] TOYOTA MOTOR SALES, USA, Inc. *Temperature Sensors* [online]. 2013, 7 s [cit. 2013-12-16]. Dostupné z: http://www.autoshop101.com/forms/h32.pdf

[11] ZABLER, Erich. *Snímače v motorových vozidlech*. 1. české vyd. Praha: Robert Bosch odbytová s.r.o. - Automobilová technika [distributor], 2003, 148 s. Technické vzdělávání. ISBN 80-903-1325-6.

[12] NIRA Dynamics - Newsletter. NIRA DYNAMICS. *NIRA Dynamics - Enhancing the safety of tomorrow's vehicle* [online]. 2011, 19.10.2011 [cit. 2013-12-16]. Dostupné z: http://www.niradynamics.se/scripts/newsletter.php?id=55

[13] TOYOTA MOTOR SALES, USA, Inc. *Pressure Sensors* [online]. 2013, 9 s [cit. 2013-12-16]. Dostupné z: http://www.autoshop101.com/forms/h35.pdf

[14] VŠB-TU OSTRAVA. *Čidla, snímače, ovládací prvky* [online]. 26.11.2013 [cit. 2013-12-16]. Dostupné z: http://fei1.vsb.cz/kat430/data/ae/Cidla_snimace_ovladaci%20prvky.pdf

[15] TOYOTA MOTOR SALES, USA, Inc. *Position Sensors* [online]. 2013, 8 s [cit. 2013-12-16]. Dostupné z: http://www.autoshop101.com/forms/h33.pdf

[16] TOYOTA MOTOR SALES, USA, Inc. *Air Flow Sensors* [online]. 2013, 9 s [cit. 2013-12-16]. Dostupné z: http://www.autoshop101.com/forms/h34.pdf

[17] Oil Debris Sensors. GILL SENSORS. *Gill Sensors - Liquid Level - Position - Speed - Flow - Condition* [online]. 2013 [cit. 2013-12-16]. Dostupné z: http://www.gillsensors.com/content/oil_debris_sensor.html

[18] TOYOTA MOTOR SALES, USA, Inc. *Position / Speed Sensors* [online]. 2013, 8 s [cit. 2013-12-16]. Dostupné z: http://www.autoshop101.com/forms/h36.pdf

[19] GROVES, Paul D. *Principles of GNSS, inertial, and multisensor integrated navigation systems*. Boston: Artech House, 2008, 518 s. GNSS technology and applications series. ISBN 978-1-58053-255-6.

[20] GEEN, John a David KRAKAUER. New iMEMS® Angular-Rate-Sensing Gyroscope. *ADI - Analog Dialogue* [online]. 2003, vol. 37, č. 1 [cit. 2013-12-16]. Dostupné z: http://www.analog.com/library/analogdialogue/archives/37-03/gyro.pdf

[21] POLÁK, Karel. Sběrnice CAN. *Elektrorevue*. 2003, roč. 5, č. 3. DOI: 1213-1539. Dostupné z: http://www.elektrorevue.cz/clanky/03021/index.html

[22] ROBERT BOSCH GMBH. *CAN Specification: Version 2.0*. Stuttgart, 1991, 73 s. Dostupné z: http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf

[23] BARTOŠÍK, Petr. Patnáct let sběrnice CAN. *Automa*. 2001, č. 11. DOI: 1210-9592. Dostupné z: http://www.odbornecasopisy.cz/index.php?id_document=33717

[24] COLEY, Gerald a Robert DAY. BEAGLEBOARD.ORG. *BeagleBone Black: System Reference Manual* [online]. 2013, 121 s. [cit. 21.4.2014]. Dostupné z: https://github.com/CircuitCo/BeagleBone-Black/blob/master/BBB_SRM.pdf?raw=true

[25] BEAGLEBOARD.ORG. *BeagleBone.org - BeagleBone Black* [online]. 2014 [cit. 2014-04-20]. Dostupné z: http://beagleboard.org/Products/BeagleBone%20Black

[26] ELINUX. *Beagleboard: BeagleBone CANBus* [online]. 2013 [cit. 2014-04-20]. Dostupné z: http://elinux.org/Beagleboard:BeagleBone_CANBus

[27] RF SOLUTIONS. *GPS-610F* [online]. Červenec 2009, 16 s [cit. 2014-04-20]. Dostupné z: www.rfsolutions.co.uk/acatalog/DS-GPS610F-2.pdf

[28] GPS (Global Positioning System) - Mobile terms glossary. GSMARENA. *GSM Arena* [online]. 2014 [cit. 2014-04-20]. Dostupné z: http://www.gsmarena.com/glossary.php3?term=gps

[29] SKYTRAQ TECHNOLOGY, Inc. *Application Note AN0003: Binary Messages Of SkyTraq Venus 6 GPS Receiver*. 4.2.2010, 53 s.

[30] NATIONAL MARINE ELECTRONICS ASSOCIATION. *NMEA0183 Standard: v 4.10* [online]. 2014 [cit. 2014-04-20]. Dostupné z: http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp

[31] UBLOX. *ANN-MS: active GPS Antenna* [online]. 2011, 14 s [cit. 2014-04-20]. Dostupné z: http://www.u-blox.com/images/downloads/Product_Docs/ANN_Data_Sheet%28GPS-X-02021%29.pdf

[32] ST MICROELECTRONICS. *L78SxxC: 2A positive voltage regulators* [online]. květen 2012 [cit. 2014-04-20]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00000449.pdf

[33] ST MICROELECTRONICS. *LSM303DLHC: Ultra-compact high-performance eCompass module: 3D accelerometer and 3D magnetometer* [online]. Listopad 2013 [cit. 2014-04-20]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00027543.pdf

[34] ST MICROELECTRONICS. *L3G4200D: MEMS motion sensor: ultra-stable three-axis digital output gyroscope* [online]. Prosinec 2010 [cit. 2014-04-20]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00265057.pdf

[35] ZÁHLAVA, Vít. *Návrh a konstrukce desek plošných spojů: principy a pravidla praktického návrhu*. 1. vyd. Praha: BEN - technická literatura, 2010, 123 s. ISBN 978-80-7300-266-4.

[36] ČSN IEC 50(161). *Mezinárodní elektrotechnický slovník. Kapitola 161: Elektromagnetická kompatibilita*.

[37] Proudová zatížitelnost vodiče. PRAGOBOARD S.R.O. *PragoBoard s.r.o.: Vícevrstvé, dvou a jednovrstvé plošné spoje* [online]. 2014 [cit. 2014-04-21]. Dostupné z: http://www.pragoboard.cz/proudova_zatizitelnost

[38] IEEE Std. 610.12-1990. *Slovník IEEE Standardů SW inženýrství*. 2012.

[39] CHISNALL, David. 10 Things I Hate About (U)NIX: Everything Is a File (Unless It Isn't). In: *InformIT: The Trusted Technology Source for IT Pros and Developers* [online]. 2005 [cit. 2014-04-22]. Dostupné z: http://www.informit.com/articles/article.aspx?p=424451

[40] Download Bing Maps Windows Prsentation Foundation (WPF) Control, Version 1.0 from Official Microsoft Download Center. *Microsoft Download Center* [online]. 2014 [cit. 2014-04-24]. Dostupné z: http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=27165

[41] STEC®, Inc. *STEC Small Cards Wear Leveling And Lifetime Calculator: Application Note AN-0702* [online]. 5. leden 2008 [cit. 2014-04-24]. Dostupné z: www.stec-inc.com/downloads/AN-0702_STEC_SMALL_CARDS_WEAR_LEVELING_LIFETIME_CALCULATOR.pdf

[42] SOCIETY, Sponsor Microprocessor Standards Committee of the IEEE Computer. *IEEE standard for floating-point arithmetic*. New York: Institute of Electrical and Electronics Engineers, 2008. ISBN 978-073-8157-535.

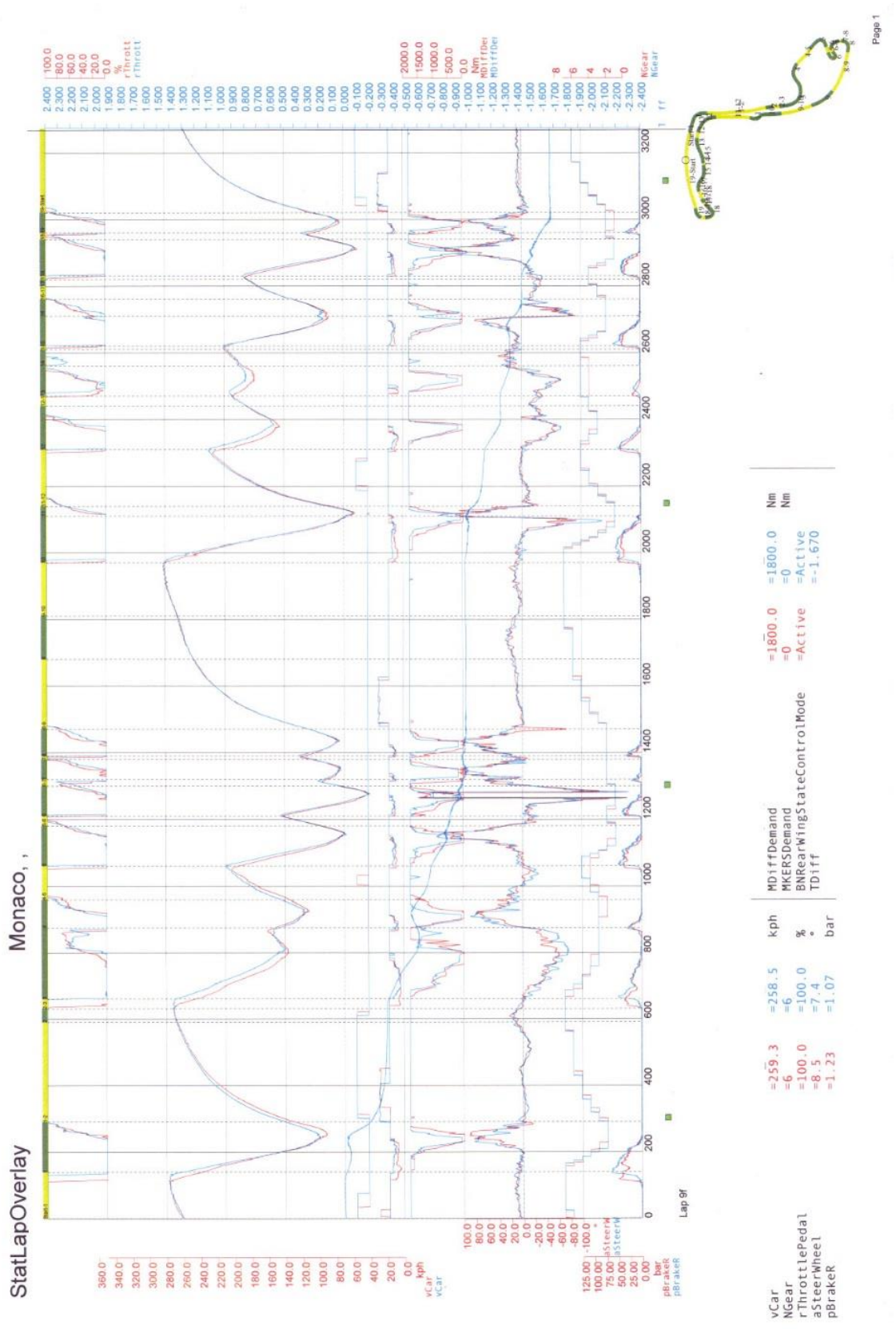
Seznam použitých zkratek

ABS – Anti-lock Brake System – automobilový bezpečnostní systém zabraňující zablokování kol
AP – Access Point – přístupový bod
Bitrate – počet přenesených bitů za jednu sekundu
Broadcast – zpráva, kterou v počítačové síti přijmou všechna připojená síťová rozhraní
CAN – Controller Area Network – komunikační sběrnice využívaná v automobilovém průmyslu
CLI – Common Language Infrastructure – mezikód, do kterého je převáděn zdrojový kód napsaný v jazyce C#
Code-behind – zdrojový kód běžící na pozadí grafické stránky při použití WPF
CRC – Cyclic Redundancy Check – hašovací funkce určená pro detekci chyb během přenosu zpráv
dps – degree per second – přesnost gyroskopu udávaná ve stupních za sekundu
DTO – Device Tree Overlay – mechanismus nastavování hardwaru
DTS – Device Tree Source – konfigurační soubory pro DTO
ECT – Engine Coolant Temperature – senzor pro měření teploty chladicí kapaliny
EGR – Exhaust Gas Recirculation – senzor teploty výfukových plynů a systém jejich recirkulace
EMC – Electromagnetic Compatibility – elektromagnetická kompatibilita
ESP – Electronic Stability Program – automobilový stabilizační systém
GPS – Global Positioning Systém – americký polohovací systém
HW – Hardware
IAT – Intake Air Temperature – senzor pro měření teploty nasávaného vzduchu
ID – Identifikátor
LED – Light-Emitting Diode – svítivá dioda
MAF – Mass Air Flow – senzor měřící průtok vzduchu
MEMS – Micro-Electro-Mechanical Systems – typ senzoru, který obsahuje jak elektronickou, tak mechanickou část
MMCX – Micro-Miniature Coaxial – typ konektoru
NTC – Negative Temperature Coefficient – typ teplotního senzoru
PTC – Positive Temperature Coefficient – typ teplotního senzoru
SMA – SubMiniature version A – typ konektoru
SMD – Surface Mount Device – součástka určená pro povrchovou montáž
SW – Software
TCP – Transmission Control Protocol – typ komunikačního protokolu v počítačových sítích
UDP – User Datagram Protocol – typ komunikačního protokolu v počítačových sítích
USB – Universal Seriál Bus – typ univerzálního rozhraní pro připojování periférií k PC
UTC – Coordinated Universal Time – systém udávající přesný čas podle atomových hodin
WPF – Windows Presentation Foundation – technologie pro .NET Framework oddělující vzhled aplikace od jejího chování
XAML – Extensible Application Markup Language – značkovací jazyk pro popis grafického rozhraní platformy .NET

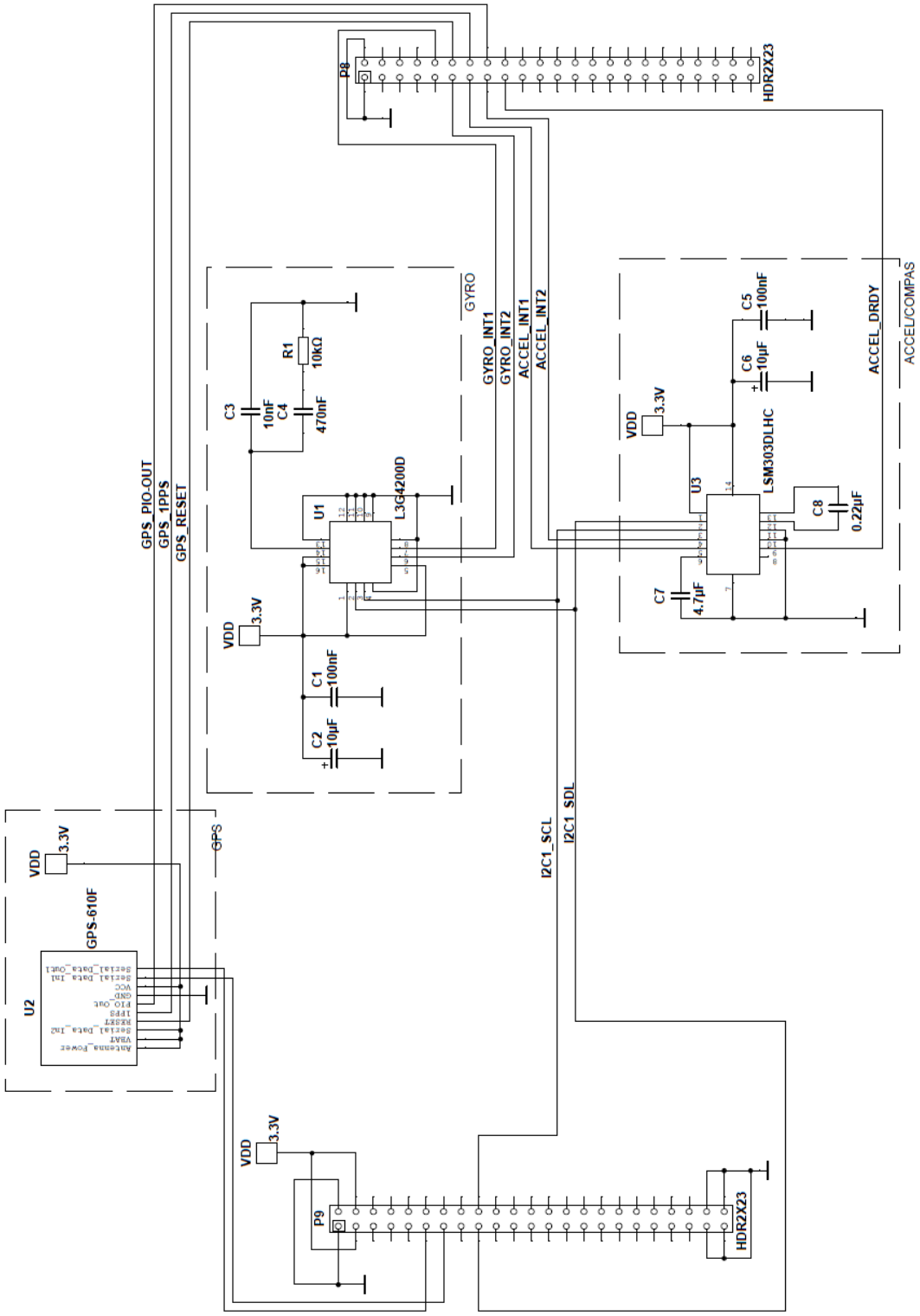
Seznam příloh

Přílohy

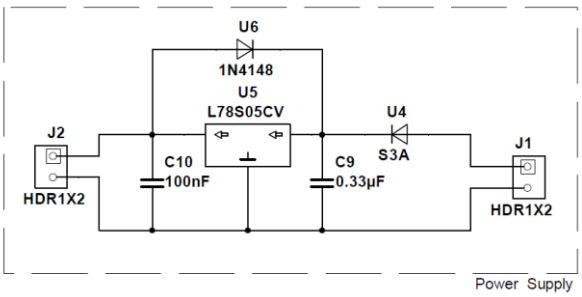
Příloha č. 1 Výstup z programu ATLAS [6]



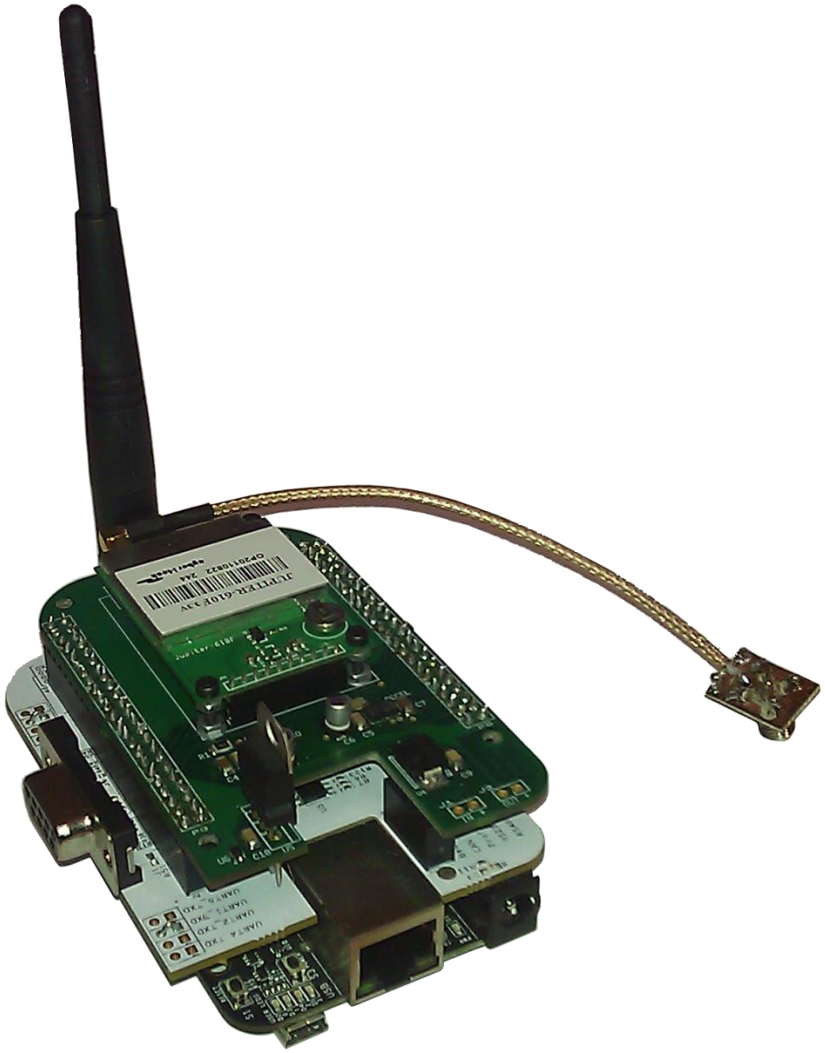
Příloha č. 2a Schéma zapojení rozšiřujícího modulu – komunikační blok



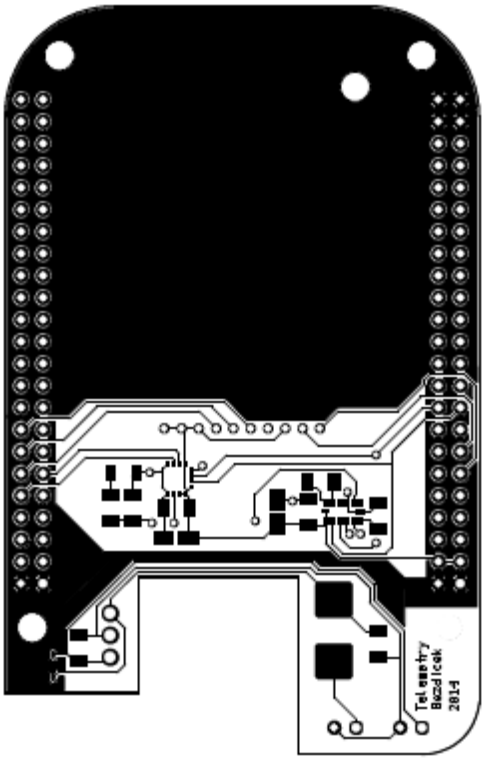
Příloha č. 2b Schéma zapojení rozšiřujícího modulu – napájecí blok



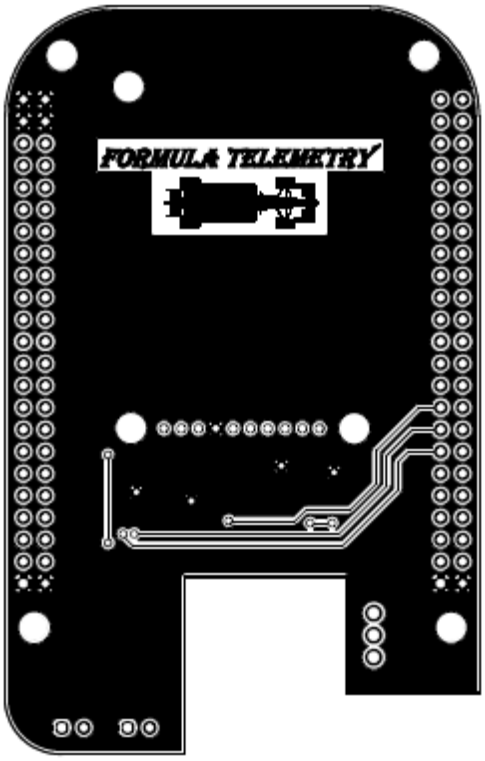
Příloha č. 3 Výsledné zařízení včetně rozšiřujícího modulu



Pohled ze strany TOP



Pohled ze strany BOTTOM



Příloha č. 5 Krabička pro telemetrii