Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídící techniky

Platforma pro prototypování algoritmů pro palubní odhadování pohybových stavů vozidel multisenzorickou fúzí

Matěj Kříž

Vedoucí: doc. Ing. Zdeněk Hurák, Ph.D.
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kříž** Jméno: **Matěj** Osobní číslo: **498936**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Platforma pro prototypování algoritmů pro palubní odhadování pohybových stavů vozidel multisenzorickou fúzí

Název bakalářské práce anglicky:

Platform for prototyping algorithms for onboard estimation of vehicle motion states by multisensor fusion

Pokyny pro vypracování:

1. Cílem projektu je vyvinout/sestavit hardwarovou platformu pro prototypování algoritmů pro odhadování pohybových stavů vozidel fúzí měření z více senzorů v reálném čase, a to měření poskytovaných moduly GNSS a IMU, ale i odometrických a další palubních měření (proudy motory tramvají) komunikovaných po palubní sběrnici. Dalším klíčovým vstupem pro algoritmy bude i digitální mapa terénu.
2. Při volbě vhodné výpočetní jednotky coby jádra celé platformy zvažujte mobilitu výsledné platformy, komerční dostupnost a zejména snadnost/rychlost vývoje algoritmů pro takovou platformu. Postačí výkonnější Raspberry Pi či BeagleBone? PC? dSpace MicroAutobox?
3. Požadavek na snadnost/rychlost vývoje vede na možnost implementovat algoritmy coby modely v Simulinku a generovat z nich kód pro cílovou platformu. Demonstrací funkčnosti platformy nechť je implementace jednoduchého algoritmu pro multisenzorickou fúzi.

Seznam doporučené literatury:

[1] J. Farrell, Aided Navigation: GPS with High Rate Sensors. McGraw-Hill, Inc., 2008.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Zdeněk Hurák, Ph.D. katedra řídicí techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **17.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

doc. Ing. Zdeněk Hurák, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Michael Šebek, DrSc. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání	Podpis studenta

Poděkování

Chtěl bych poděkovat zejména doc. Ing. Zdeňku Hurákovi, Ph.D. za odborné vedení a velkou ochotu během této práce i projektech, které jí předcházely. Dále bych chtěl poděkovat týmům *Škoda Digital* a *Herman elektronika*, za možnost čerpat z jejich zkušeností s tímto tématem. Velké poděkovaní patří také Loiovi Do za výraznou pomoc, zejména v počáteční fázi, při identifikaci možných problémů, které by tento projekt měl vyřešit.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a s použitím uvedené literatury a pramenů.
V Praze, 26. května, 2023

Abstrakt

Tato práce se zabývá vývojem jednodeskové platformy pro sběr dat ze senzorů pohybových stavů vozidel a pro prototypování algoritmů, které tato data fúzují. Tato platforma by měla zjednodušit záznam dat a standardizovat jejich formu pro zlepšení znovupoužitelnosti pro další projekty. Dále se zabývá dalšími podklady pro snadnější implementaci algoritmů, jako je práce s mapovými podklady a geografickými referenčními systémy.

Klíčová slova: senzorická fúze, odometrie, tramvaj, odhadování, záznam dat, ROS 2, IMU, GNSS, S–JTSK, embedded

Vedoucí: doc. Ing. Zdeněk Hurák, Ph.D.
Katedra řídící techniky
Karlovo náměstí 13–E
Praha 2

Abstract

This thesis describes development of embedded platform for recording data from vehicle motion–states–sensors and for development of algorithms using fusion of such data on board vehicles. This platform should simplify recording and standardize data format for reusability in future projects. The thesis also describes other improvements including algorithm–development simplifications concerning usage of maps and geographic reference systems.

Keywords: sensor fusion, odometry, tram, estimation, data logging, ROS 2, IMU, GNSS, S–JTSK, embedded

Title translation: Platform for prototyping algorithms for onboard estimation of vehicle motion states by multisensor fusion

Obsah

Obrázky

Kapitola 1
Úvod

1.1 Cíl

Cílem této práce je vytvoření platformy pro záznam a interakci se senzory pohybových stavů vozidel. Mezi hlavní cílová vozidla patří vozidla hromadné dopravy, zejména pak kolejová vozidla. Interakce se senzory by měla umožnit pokusy s algoritmy v reálném čase při pohybu vozidel. Záznam by měl být alespoň částečně autonomní pro fungování bez nutnosti fyzické přítomnosti u experimentu.

1.2 Motivace

Partneři z průmyslu jako např. *Škoda Digital* a *Herman elektronika* mají vlastní jednotky. Ty jsou však často nepřístupné z hlediska vnitřního fungování a parametrů. Dále není možné provádět testování a sběr dat v libovolném termínu a čase, jelikož jsme závislí na kapacitách a možnostech partnerů. Další výhodou vlastní platformy je standardizovaná struktura dat a možnost spouštět algoritmy v reálném čase přímo na jednotce.

Práce je součástí rozsáhlejšího projektu spadajícího pod Katedru řídící techniky. Jako hlavní práce probíhající současně s touto, bych uvedl práci Jakuba Kašpara, zabývající se přesnější lokalizací pomocí fúze jednotlivých senzorů, práci Radka Chládka, zabývající se odhadováním sklonu trati v daném bodu tratě a práci Matouše Vondráška, věnující se odhadu decelerace.

V předchozích experimentech a projektech bylo nutné znovu sbírat vlastní data. Tato práce by měla tuto činnost usnadnit tím, že standardizuje strukturu a způsob získávání dat. Jednotlivé senzory budou programovány blokově, pro lepší opětnou použitelnost i mimo zvolenou platformu.

1.3 Struktura práce

V kapitole 2 se věnuji popisu jednotlivých možností výběru platformy a konkrétnímu výběru pro toto užití. Kapitola obsahuje také informace o

jednotlivých platformách z hlediska vlastností jako konektivita a výpočetní výkon.

Kapitola 3 se zabývá popisem senzorů a implementace jejich komunikace. Věnuje se také komplikacím při jejich použití pro případné další projekty. Zahrnuje informace o senzorech GNSS, IMU a odometrie.

Kapitola 4 se zabývá možnostmi implementace vnitřní komunikace platformy a konečným výběrem z těchto možností pro tento projekt. Zároveň se věnuje popisu instalace a možným komplikacím při práci s těmito systémy.

V kapitole 5 se zabývám integrací celé struktury s prostředím *Matlab* pro implementaci algoritmů. Tato kapitola se také věnuje popisu prostředí *ROS Toolbox*.

Kapitola 6 se zabývá mapovými podklady včetně převodu ze systému *WGS84* na *S–JTSK*. Uvádí také popis zdroje dat pro výškových souřadnic pro Prahu a okamžité okolí.

V předposlední kapitole 7 je popsána struktura ukládání dat pro provedené a budoucí experimenty a měření.

Poslední kapitola 8 se zabývá testováním a popisem systému z pohledu uživatele. Zároveň obsahuje popisy provedených experimentů.

Kapitola 2

Platforma

2.1 Výběr platformy

2.1.1 Požadavky

Pro výběr platformy byly stanoveny základní požadavky:

Výkon

Výkonnostní charakteristika je náročná na kvantifikaci. Na trhu je, vzhledem k dominanci AI vývoje, mnoho platforem jejichž výkon je přizpůsoben tomuto úkolu. Jako příklad lze uvést řadu `NVIDIA Jetson`, s výkonnými GPU.

Pro tuto aplikaci je potřebný výkonný procesor pro rychlé výpočty modelu. Není proto třeba investovat více do platforem určených k AI. i přes vyšší pořizovací cenu jsou zde platformy uvedeny, pro lepší orientaci v jednotlivých možnostech.

Integrovatelnost s prostředím *Matlab* a *Simulink*

Jelikož většina níže zmíněných platforem funguje na operačním systému *Linux*, je jejich spolupráce s prostředím *Matlab* relativně přímočará. Pro procesorové architektury `ARM` a `ARM-64` je třeba skripty a modely pro *Matlab* a *Simulink* kompilovat do jazyka *C/C++*. Platformy `x86-64` jsou schopné spustit tyto skripty a modely v interpretovaném režimu, popřípadě je možné je na této platformě přímo vyvíjet.

Cena

Cenově vhodnou platformou se jeví ta, kde poškození nebo ztráta neohrozí možnost podnikat další pokusy. Jelikož je jedním z požadavků možnost zanechat platformu v prostředku hromadné dopravy pro sběr dat, není, vzhledem k možnosti poškození, možné využít platformy jako *dDPACE LabBox*. Cena by se měla pohybovat řádově v tisících až desetitisících korun.

Podobnost se známými platformami

Většina potenciálních uživatelů je seznámena s vývojem na platformách jako `Raspberry PI`. Přestože bude úkolem této práce, co nejvíce oddělit samotnou platformu od experimentů, není vyloučena možnost dalšího vývoje platformy jinými uživateli. Je proto výhodné, pokud budou předem obeznámeni ze základním fungováním hardwaru a operačního systému dané platformy.

Open–source

Pro lepší integrovatelnost platformy by bylo vhodné mít k dispozici otevřené zdroje pro software i hardware. Některé senzory toto nemohou splňovat, ale při možnosti výběru bude lepší zvolit open–source alternativu.

2.1.2 Přehled parametrů a dostupnosti

Výčet platforem s výrobci je uveden v tabulce 2.1. Podrobnější parametry jsou uvedeny v dodatku A. Na obrázku 2.1 je pouze zjednodušené zakreslení parametrů platforem.

výrobce	název	architektura
Raspberry PI F.	*Raspberry PI 4B 4GB* [9]	`ARM-64`
ODROID	*H3+* [17]	`x86-64`
ODROID	*M1 8GB* [18]	`ARM-64`
ODROID	*N2+ 4GB* [19]	`ARM-64`
NVIDIA	*Jetson AGX Orin 64GB D. K.* [22]	`ARM-64`
NVIDIA	*Jetson Nano D. K.* [22]	`ARM-64`
SeeedStudio	*Jetson AGX Orin 32GB H01 D. K.* [16]	`ARM-64`
BeagleBone	*AI-64* [4]	`ARM-64`
Solidrun	*HoneyComb LX2* [23]	`ARM-64`

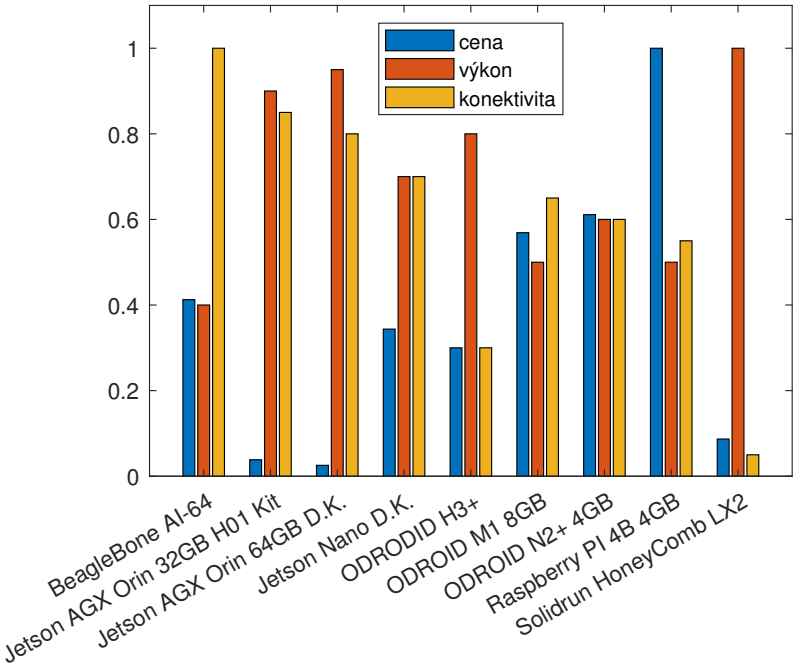
Tabulka 2.1: Výčet platforem

Všechny tyto platformy ve svém základním nastavení fungují na sytému *Linux*. Všechny jsou postaveny na architektuře `ARM-64`, s výjimkou `ODROID H3+`, který je postavený na procesoru `x86-64` od společnosti `Intel`.

2.1.3 Kritéria výběru

Pro danou aplikaci je vhodné, aby platforma měla následující vlastnosti:

1. co nejvyšší počet externích komunikačních rozhraní pro komunikaci s vnějším prostředím
2. rychlé a spolehlivé úložiště pro ukládání velkého množství dat (např. karta SD není považována za dlouhodobě spolehlivou)
3. rozhraní Ethernet pro připojení k tramvaji za využití protokolu TRDP
4. rozhraní pro připojení k PC pro snadnou konfiguraci (např. USB Device, debug UART, druhý Ethernet konektor)



Obrázek 2.1: Zjednodušené srovnání platforem. Výkonem je myšlen výkon CPU a konektivitou množství externích rozhraní (SPI, UART, I2C, GPIO, PWM). Vyšší hodnocení ceny znamená levnější produkt.

2.1.4 Rozhodnutí

Vzhledem k možnosti spustit nekompilovaný *Matlab/Simulink* skript jsem se rozhodnul pro zařízení `ODROID H3+` (na obr. 2.2), a to i přes absenci mnoha externích rozhraní a vyšší spotřebu, která znemožňuje napájení z menších akumulátorů. Připojení některých externích senzorů bude problematičtější, ale implementace bude, vzhledem k podobnosti s klasickým laptopem, jednodušší. Do budoucna by toto také mělo výrazně zjednodušit práci se zařízením.

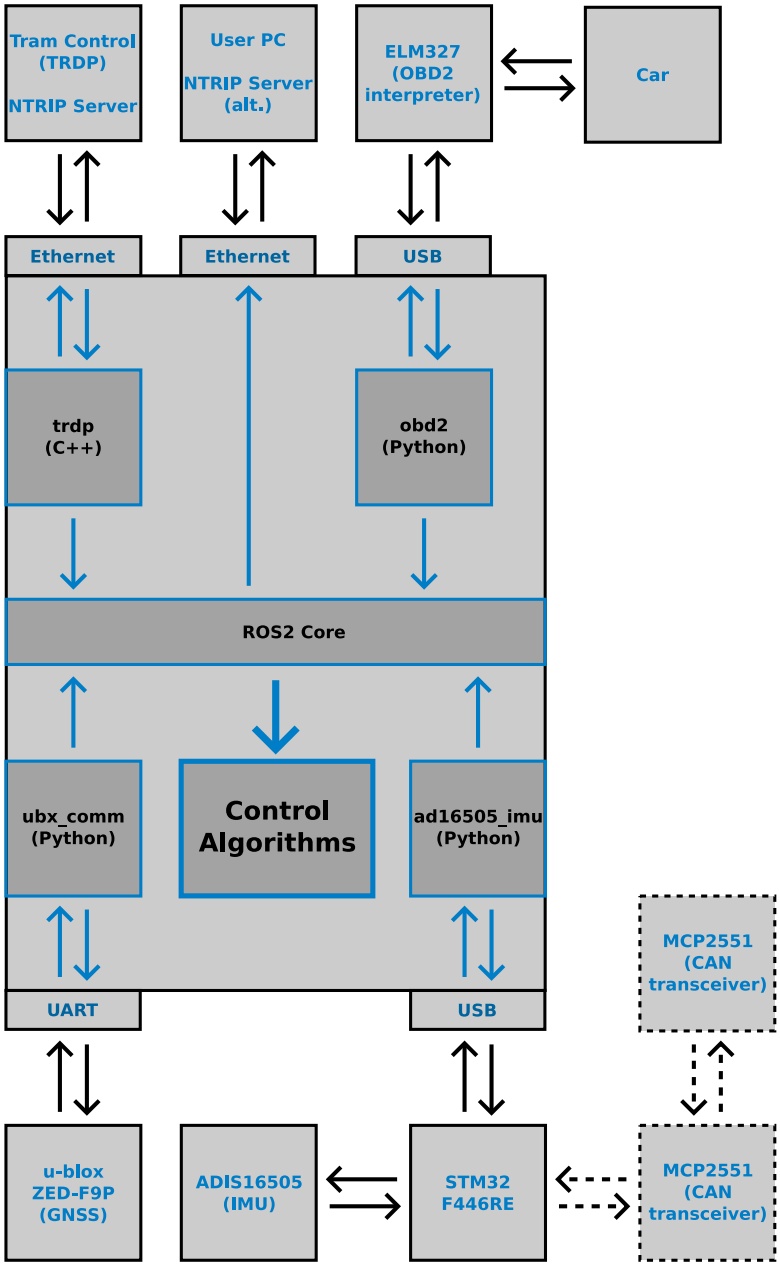
Při integraci *Matlab* bude možné nejen spouštět nekompilované skripty, ale také kompilovat přímo na zařízení vývojáře a na platformu odesílat přímo binární soubory.

Obrázek 2.2: Jednodeskový počítač *ODROID H3+* (Zdroj obrázku: [17])

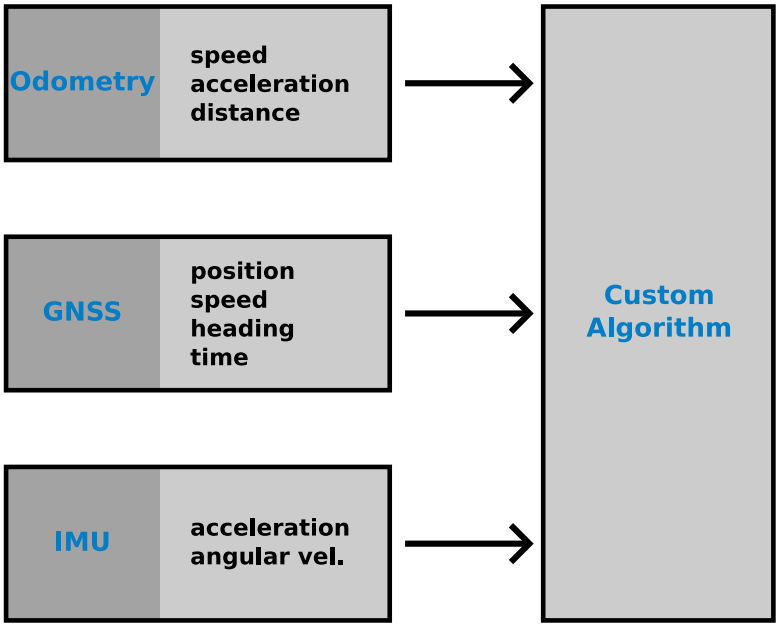
Kapitola 3

Senzory a komunikace

Nejdůležitějšími komponentami jsou jednotlivé senzory. Jejich primární rozložení je zobrazeno na obrázku 3.1. Pro zjednodušení je také uvedena datová struktura pro algoritmy na obrázku 3.2. Fotografie celé jednotky je na obrázku 3.3.



Obrázek 3.1: Rozložení senzorů a komunkace



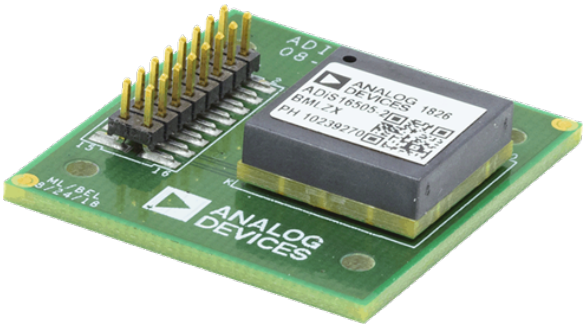
Obrázek 3.2: Datová topologie celého zařízení

Obrázek 3.3: Fotografie celé jednotky

3.1 Inerciální měřící jednotka

3.1.1 Požadavky

Pro práci nám byla předem dostupná IMU *ADIS16505–1* (na obrázku 3.4) od společnosti *Analog Devices*. Byla již použita v předchozích pokusech, ale pouze za pomoci breakout desky *EVAL-ADIS-FX3* od stejné společnosti. Tato deska je kompatibilní pouze s operačním systémem Windows, což by nám neumožnilo přejít pro platformu na Unixový systém, který je společný pro všechny uvažované platformy.



Obrázek 3.4: Senzor *ADIS16505–1* s výchozí deskou (Zdroj obrázku: [3])

Jednotka byla vyžívána již dříve při podobných pokusech. Dobé zkušenosti s jejím používáním a parametry odpovídající konvenčnímu pohybu povrchových dopravních prostředků (např. rozsahy a rozlišení), převážily pro vývoj vlastního připojení k počítači.

3.1.2 Přehled komunikace

Zvolená IMU umožňuje SPI komunikaci s parametry uvedenými v tabulce 3.1. Samotný proces komunikace probíhá následovně: IMU v režimu vlastního hodinového cyklu 2 kHz změní polaritu výstupního signálu `DATA OUT` ve chvíli kdy naměří nová data, na tento signál je navázáno přerušení mikroprocesoru (zvolen byl mikroprocesor *Nucleo STM32F446–RE*), který následovně vyčte požadovaná data. Další možností je dodávat IMU externí hodinový signál který může mít jinou frekvenci, případně pomocí pulsů na vstupním kanálu `SYNC` synchronizovat měření s aplikací.

Pro čtení dat z IMU je výhodné využívat tzv. `BURST MODE`, který umožňuje bez přestávek mezi čtením vyčíst až 32 B dat. Klasické čtení dat funguje na principu dotazu na jeden 16-bitový registr. V `BURST` tomto režimu se kontinuálně odešlou údaje z tříosého gyroskopu, akcelerometru, údaje o teplotě a časové údaje měření.

Jelikož samotný akcelerometr je třeba oddělit od vibrací platformy a zároveň by bylo vhodné implementovat ho tak, aby byl použitelný i pro další zařízení jako PC nebo jiné jednodeskové počítače, bylo jako rozhraní mezi počítačem a mikroprocesorem *STM32* zvoleno USB. Pro takovouto aplikaci plně postačí

SCLK	$< 2\,\mathrm{MHz}$
CPOL	high
CPHA	2. hrana
bit mode	16-bit

Tabulka 3.1: Popis SPI komunikace zařízení ADIS16505–1

název	popis	rozlišení	rozsah
`x_gyro`	osa gyroskopu x	$9.5367 \times 10^{-8}\,°/\mathrm{s}$	$-125 - 125\,°/\mathrm{s}$
`y_gyro`	osa gyroskopu y	$9.5367 \times 10^{-8}\,°/\mathrm{s}$	$-125 - 125\,°/\mathrm{s}$
`z_gyro`	osa gyroskopu z	$9.5367 \times 10^{-8}\,°/\mathrm{s}$	$-125 - 125\,°/\mathrm{s}$
`x_accl`	osa akcelerometru x	$4.6730 \times 10^{-6}\,\mathrm{m/s^2}$	$-78.3 - 78.3\,\mathrm{m/s^2}$
`y_accl`	osa akcelerometru y	$4.6730 \times 10^{-6}\,\mathrm{m/s^2}$	$-78.3 - 78.3\,\mathrm{m/s^2}$
`y_accl`	osa akcelerometru z	$4.6730 \times 10^{-6}\,\mathrm{m/s^2}$	$-78.3 - 78.3\,\mathrm{m/s^2}$
`temp`	teplota IMU	$0.1\,°\mathrm{C}$	$-40 - 105\,°\mathrm{C}$

Tabulka 3.2: Popis 32–bitového `BURST` režimu u *ADIS16505–1*

jednodušší protokol *USB–FS* (*Full Speed*). Mikroprocesor *STM32F4* poskytuje základní rozhraní pro *LL* (*Low–Level*) knihovny a také Middleware nadstavbu pro implementaci Virtuálních `COM` rozhraní pro USB.

Virtuální `COM` rozhraní skrze USB umožní jednodušší implementaci pro další použití bez nutnosti větších znalostí problematiky protokolu USB pro další uživatele. Dále také zachová základní velikost datových balíků 64 B, což umožní posílat veškeré zprávy získané ze zařízení IMU a také např. implementaci `CAN BUS` (popsáno v části 3.5).

V rámci jednoduchosti implementace byla zvolena náhrada tohoto USB rozhraní skrze *ST–LINK F103*, jelikož zařízení *Nucleo STM32F446–RE* hardwarově neodpovídá pro připojení USB zařízení. Implementace skrze USB je zahrnuta ve zdrojích, zároveň je ale uvedena implementace skrze UART připojení *ST–LINK F103*.

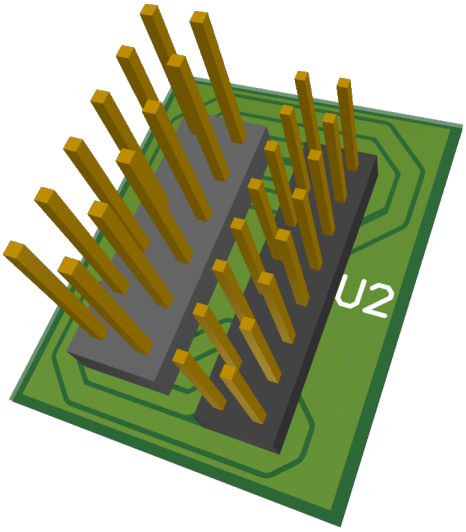
Pro vyčtení dat z IMU je použito externí přerušení, implementováno skrze *Low–Level* knihovny. Toto přerušení vyvolá čtení 32–bitového balíku obsahujícího data popsaná zapsaná v tabulce 3.2.

3.1.3 Komplikace

Pro implementaci SPI je poměrně běžné, že softwarové ovladače neumožňují komunikaci na 16–bitových slovech, což komplikuje integraci s prostředími jako *Python*. Také knihovny *HAL* pro *STM32* mají poměrně nepřívětivé rozhraní pro práci s 16–bitovými slovy. Bylo proto nutné pro spolehlivost využít *LL* knihovny a např. přeposílání skrze přerušení implementovat ručně.

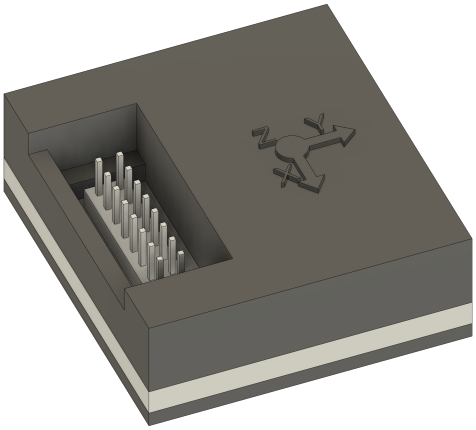
Aby nedošlo k poškození samotné IMU, nebylo možné ji oddělit od vývojové desky, která je připojena konektorem 2 x 8 pinů. Tento konektor má rozteč 2 mm, což neodpovídá častěji užívané rozteči 2.54 mm. Jelikož konektory těchto dvou verzí do sebe vzájemně nepasují, bylo třeba vytvořit převodník, tento převodník byl dočasně vytvořen připájením vodičů na kolíkové lišty, ale

pro budoucí použití bylo vytvořeno PCB (zobrazeno na obrázku 3.5).



Obrázek 3.5: Breakout deska pro IMU

Možnost mechanického poškození samotné IMU byla řešena obalem, vytištěným na 3D tiskárně. Tento obal je zobrazen na obrázku 3.6.



Obrázek 3.6: Obal pro IMU *ADIS16505*

3.1.4 Parametry implementace

Obecně má rozhraní zpoždění kolem 0.5 ms. Jedinou problémovou částí je spojení mezi mikroprocesorem a zařízením *ODROID H3+*, které je momentálně implementováno skrze *ST–LINK F103*. Dalším možným zrychlením by byla paralelizace odesílání dat skrze `UART` s přijímáním skrze `SPI`, jelikož se však v obou případech jedná o nepaketové připojení. To by ale nebylo kompatibilní s rozhraním USB, pro které je toto pouze zástupná metoda.

Propustnost skrze USB je srovnatelná s původní deskou *EVAL-ADIS-FX3*, připojení skrze UART vykazuje určité zpoždění. Experimentálně bylo zjištěno,

že pro stabilizaci připojení je vhodnější odesílat pouze každé 4. měření. Pro připojení skrze čisté *USB Full Speed* je praktickou frekvencí 1000 Hz, tudíž každé 2. měření, při plné frekvenci 2000 Hz jsou přítomny nezanedbatelné chyby. Jedním z řešení je přenášet skrze 64–bytové pakety 2 měření zároveň s předpokladem, že čas příjmu prvního z nich by se dopočítal jako

$$t_1 = t_2 - \frac{1}{2000} \ .$$

Pokud by se zvažovalo použití *USB High Speed*, bylo by nutné k zařízení připojit adekvátní fyzickou vrstvu, jelikož čipy *STM32* nejsou samy o sobě schopny vytvořit dostatečně rychlý signál pro tento typ USB. Výrazně by to však zrychlilo připojení k počítači, jelikož tato USB konfigurace zrychluje připojení na 480 Mbit/s a zvětšuje pakety na 255 bytů.

Implementace zároveň umožňuje testy bez integrace z vnitřním prostředím, jelikož blokový kód pro IMU je oddělen tak, aby byl použitelný sám o sobě.

3.1.5 Testování implementace

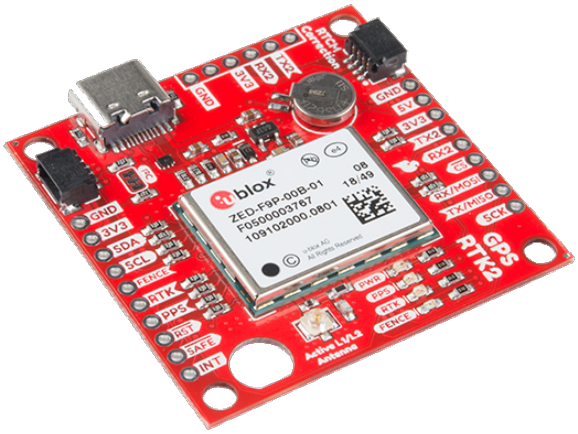
Nejdelší test, kdy systém stabilně běžel trval 4 hodiny a 36 minut. Tento test probíhal pouze v laboratorních podmínkách a nelze tedy přímo vyvozovat stabilitu ve vozidlech. Jediný problém by mohl nastat v operačním systému nebo hardwaru připojení. Hardware je dostatečně robustní a software je implementován tak aby se sám spustil při spuštění zařízení, a restartoval při chybě procesu, lze tudíž předpokládat, že delší měření by nemělo způsobit větší problémy. Data z měření jsou dostupná v dokumentaci přiložené k práci.

Nejdelší test ve vozidle trval 53 minut v automobilu. Po celou dobu experimentu byl systém stabilní.

3.2 GNSS přijímač

3.2.1 Požadavky

Pro pokusy byl předem zvolen dostupný a často užívaný modul *u–blox ZED–F9P*. Konkrétně ve verzi *00B–002*. Pro snadnější manipulaci byla použita vývojová deska *SparkFun GPS-RTK2 Board–ZED–F9P* (na obrázku 3.7). Pro danou aplikaci je třeba co nejvyšší frekvence aktualizací pozice. Tento modul umožňuje frekvenci až 40 Hz, za splnění požadavků dále zmíněných.



Obrázek 3.7: *u–blox ZED–F9P* s deskou *Sparkfun* (Zdroj obrázku: [1])

Další výhodou je schopnost přímé integrace diferenciální GNSS pomocí zpráv `RTCM` protokolu. Běžně by bylo nutné implementovat vlastní filtr pro využití diferenciální GNSS, jelikož však tato implementace není hlavním cílem, a jedná se pouze o volitelné zpřesnění pozice, využil jsem implementace zahrnuté přímo v modulu *ZED–F9P*.

3.2.2 Přehled komunikace

Zařízení *u–blox ZED–F9P* přichází s již implementovaným komunikačním rozhraním protokolu `ubx`. Existuje mnoho otevřených knihoven pro jeho využití. Alternativou je protokol `NMEA`, který je také podporován. Protokoly mají několik výrazných rozdílů, `NMEA` protokol využívá ke komunikaci zprávy založené na `ASCII`, tudíž je čitelný pro lidskou obsluhu. Oproti tomu `ubx` protokol je založený na binárních zprávách. Výhodou komunikace skrze binární rozhraní je vyšší přenosová rychlost.

Protokol `ubx` je také uzpůsoben vnitřním parametrům modulu. Podporuje syntetizované zprávy, kdy modul využije data z více GNSS sítí, přiřadí jim přesnost a podle nich vypočte přesnější polohu. Hlavní využívanou zprávou bude `UBX-NAV-PVT`, která poskytne veškeré měřitelné údaje o pozici a rychlosti, s několika dalšími informacemi uvedenými v tabulce 3.3.

Zatímco souřadnice pozice jsou vypočteny fúzí vícero konstelačních systémů, hodnota pozemní rychlosti dodávaná modulem se získává pomocí Dopplerova jevu a měření pochybu vůči satelitům.

Vzhledem k vyšší rychlosti protokolu `ubx` a jeho výborné podpoře i ve srovnání s obecně rozšířenějším protokolem `NMEA` (včetně překladačů do `NMEA` [11]) jsem zvolil protokol `ubx` jako základ pro získávání dat. Ukládání probíhá do `.csv` souborů, není tedy třeba mít `ASCII` kompatibilní protokol určený pro zobrazení dat.

K automatické konfiguraci modulu lze využít sériové rozhraní. Jelikož modul má definované čekací doby po konfigurační zprávě a chybovost sériového přenosu pro nižší frekvence rozhraní UART je prakticky nulová, lze zařízení konfigurovat čistě pomocí odeslání předem připraveného řetězce. Zároveň

název	popis
fixType	Typ fixace pozice (2D/3D/none)
lat	údaj zeměpisné šířky v systému *WGS84*
lon	údaj zeměpisné délky v systému *WGS84*
height	výška nad referenčním elipsoidem *WGS84*
hACC	odhad horizontální přesnosti
vAcc	odhad vertikální přesnosti
gSpeed	pozemní rychlost (2D)
headMot	směr pohybu

Tabulka 3.3: Důležité údaje z ubx zprávy UBX-NAV-PVT

modul po příkazu uložení zachovává data i po odpojení napájení. Konfigurace modulu není přímo implementována, jelikož záleží na možnostech a požadavcích uživatele. Doporučuje se vypnou zprávy NMEA a ponechat zapnutou pouze zprávu UBX-NAV-PVT.

Pro základní komunikaci byla využita knihovna *pyubx2* [5], jelikož má dlouhodobou podporu a je nejčastěji využívanou knihovnou. *u–blox* poskytuje vlastní knihovny pro komunikaci v jazyce *C*, ty jsou však určeny pro finální produkty, nikoliv pro prototypování.

3.2.3 Komplikace

Původně bylo zamýšleno pro komunikaci využít rozhraní I2C, to ale nepodporuje odesílání RTCM zpráv do modulu, bylo tedy nahrazeno rozhraním UART. Tato změna proběhla bez výrazných změn pro vnitřní fungovaní obou komponent.

3.2.4 Parametry implementace

Byly vytvořeny dvě implementace: jedna za použití jazyka *Python* a knihovny *pyubx2*, druhá pomocí jazyka *C++*. Implementace v jazyce *C++* byla rychlejší, ale nutnost kompilace a nepřenositelnost nebyly praktické pro toto konkrétní využití, aby bylo možné obhájit její využití. *Python* implementace je pomalejší, ale usnadňuje úpravy pro rychlejší vývoj, zároveň zahrnuje možnost dekódovat i jiné zprávy než UBX-NAV-PVT. Také zahrnuje dekódování NMEA a RTCM protokolů bez nutnosti další úpravy nebo studia dokumentace.

3.3 On-board diagnostics (OBD 2)

3.3.1 Požadavky

Pro snadnější vývoj a testování algoritmů bylo nutné zbavit se závislosti na hromadné dopravě, jelikož ne vždy je možné testovat přímo na lince. Jako nejlepší simulace byl zvolen osobní automobil a rozhraní OBD 2, jakožto dostupné a snadno připojitelné rozhraní.

příkaz	význam
`AT Z`	restartování
`AT SP n`	přepnutí protokolu (`n` – číslo protokolu)
`01 00`	vyčtení dostupných příkazů
`01 0D`	vyčtení otáček motoru
`01 0D`	vyčtení rychlosti
`01 11`	vyčtení pozice plynového pedálu

Tabulka 3.4: Příkazy pro zařízení *ELM327*

Tento senzor slouží pouze pro testování, není tedy nutné, aby splňoval náročné požadavky. Hlavním požadavkem je jeho co největší jednoduchost a spolehlivost. Implementace musí fungovat spolehlivě bez dalších nastavení. Jako volitelné měřené veličiny musí zahrnovat rychlost vozidla, stlačení plynového pedálu a otáčky motoru, se snadnou implementací dalších veličin.

Jako vhodný překladač byl zvolen *ELM327* od společnosti *ELM Eletronics*. Tento překladač překládá všechny `OBD 2` protokoly na unifikované rozhraní `ASCII` – `RS232` a je připojitelný skrze USB.

3.3.2 Přehled komunikace

Existuje mnoho veřejně dostupných implementací `OBD 2` pro jazyky jako *Python* nebo *C++*, většinou jsou tyto implantace příliš složité a obecné pro jednoduché využití jako je vyčítání rychlosti automobilu. Napsal jsem tedy vlastní jednodušší implementaci, která by měla být spolehlivější než většina veřejně dostupných za cenu značně omezených schopností.

Při implementaci jsem vycházel z dokumentace pro zařízení *ELM327* [6] a popisu `OBD 2` protokolu [30] a výčtu komunikačních kódů [29]. Implementace je vytvořena v jazyce *Python* pro snadnou úpravu, nižší rychlost v tomto případě není problémem, jelikož samotné rozhraní `OBD 2` není příliš rychlé a dosahuje maximální spolehlivé frekvence zpráv okolo 5 Hz, pro simulaci zpráv tramvaje nebo autobusu hromadné skrze `TRDP` (část 3.4) nebo rozhraní `CAN` (část 3.5) je nutné výsledky interpolovat.

Pro konfiguraci modulu se využívají `AT` příkazy, podobné jako u síťových prvků, příkazy se ukončují znakem `\r` (carriage return). Základní příkazy, využité v implementaci, jsou zapsány v tabulce 3.4.

3.3.3 Komplikace

Přestože se jedná o poměrně běžné a často používané rozhraní má některé části, které nejsou plně jasné. Asi největší nejasností je výběr konkrétního protokolu. Modul obsahuje systém automatického nařízení protokolu, ale tento systém funguje jen na novějších vozidlech, která jsou plně kompatibilní s `OBD 2`. Pro starší vozidla, u kterých je kompatibilita pouze částečná (podstatná část vozidel z roku 2003 a dříve [30]), je třeba otestovat všechny protokoly a když dojde k chybě celý modul restartovat.

3.3.4 Parametry implementace

Jak již bylo zmíněno výše, implementace má stabilní dosažitelnou frekvenci 5 Hz pro všechny zprávy. Zpoždění není měřitelné a žádná dokumentace tento údaj neobsahuje a z velké části je závislý na parametrech počítače samotného automobilu. Čistě z porovnání dat získaných z GNSS a `OBD 2` je možné usuzovat, že se pohybuje mezi 100 a 400 ms. Tato odchylka je značně nestabilní a není proto možné tento senzor použít pro rigorózní měření rychlosti vozidla.

Další limitací je rozlišení měření, kdy nejnižší měřená jednotka je 1 km/h, což není použitelné pro přesná měření, zároveň je vhodné usuzovat, že některá vozidla mohou mít tento údaj zkreslený z mnoha různých důvodů. Jelikož nebyl testován každý automobil nelze předpokládat, že se jedná o přesný odhad.

Směr pohybu je vzhledem k charakteru zprávy nemožné určit. Ve chvíli, kdy vozidlo zastaví by bylo nutné předpokládat dva systémy, jeden pro každý směr. Nepředpokládá se, že by se při pokusech couvalo, ale je vhodné tuto informaci při experimentech zohlednit a změnu směru pohybu zaznamenat, aby nezpůsobovala chyby.

Otáčky motorů jsou měřitelné v rozsahu 0 – 16383.75 RPM, s minimální jednotkou 0.25 RMP. Stlačení plynového pedálu je udáváno v rozsahu 0 – 1 s minimální jednotkou $\frac{1}{255}$.

Implementace umožňuje separátní užití senzoru bez vnitřní komunikace platformy skrze oddělenou implementaci v rámci zdrojových kódů.

3.3.5 Testování implementace

Implementace je stabilní a neměla by způsobovat problémy, přesto je vhodné při experimentech ve vozidle kontrolovat frekvenci příchozích zpráv. Čtení by se mělo při chybě procesu samo znovu nastartovat. Je ale možné, že restart při větší chybě bude, zejména u starších vozidel, vyžadovat restartování palubního počítače a celého motoru.

Implementace byla bez problémů kompletně testována na vozidlech *Škoda*, *Fiat* a *Seat*. Další vozidla, zejména pak novější modely by neměly způsobovat žádné problémy.

3.4 Train Real-time Data Protocol (TRDP)

`TRDP`, protokol využívaný v kolejových vozidlech, je vyvíjený a udržovaný open-source iniciativou *TCNOpen* [24]. Je využívaný prakticky ve všech moderních kolejových vozidlech. Pro nás je důležité zejména jeho využití v tramvajích *Škoda 15T*. *Škoda Trasnportation* je také pozorovatelem při iniciativě *TCNOpen* [24].

název	popis	jednotka	rozlišení
`speed_FDI`	interně vypočtená rychlost tramvaje	m/s	10^{-8}m/s
`teethCountAbsolute`	absolutní stav čítače na kolech	–	1
`speed_VCU`	rychlost poskytnutá jednotkou *VCU*	m/s	n/a
`teethCountRelative`	relativní stav čítače na kolech	–	1
`weight`	odhad váhy tramvaje	t	1 t
`force`	třecí síla zrychlení a brždění	N	n/a
`timestamp`	časová značka měření (od 1970)	s	$\frac{1}{65536}$ s

Tabulka 3.5: Údaje poskytované tramvají *Škoda 15T*

3.4.1 Popis protokolu

`TRDP` protokol je založený `UDP/IP` a `TCP/IP` zprávách. Zahrnuje dva základní typy zpráv: `Process Data` a `Message Data`. Pro naší implementaci je důležitý právě první zmíněný typ. Centrální počítač tramvají *Škoda 15T* lze propojit s naší platformou konektory `Ethernet`.

Iniciativa *TCNOpen* poskytuje základní rozhraní v jazyce *C* [25]. Toto rozhraní nazvané `TRDP Light` umožňuje základní komunikaci na úrovni `publisher` – `subscriber`. Zároveň je navázané na `VOS` – virtuální operační systém, který zaručuje kompatibilitu s vícero architekturami a operačními systémy. Pro naše potřeby jsme knihovnu zkompilovali pro operační systém Linux a architekturu `x86-64`. Jako další, pro nás důležitá možnost je kombinace Linux – `ARM-64` pro případné použití na jiných platformách jako např. *Raspberry PI 4B*. K tomuto účelu je nutné znovu zkompilovat knihovny pro danou platformu, celý postup je popsaný v dokumentaci knihovny.

Základní implementace nám byla poskytnuta týmem *Škoda Digital*, nutné úpravy zahrnují integraci do vnitřního komunikačního prostředí, vytvoření metod pro záznam a testování spolehlivosti. Jako základní jazyk implementace byl zvolen jazyk *C++*. Hlavními důvody pro tuto volbu jsou: jednoduchost komunikace s oficiální knihovnou projektu `TRDP` a absence nutnosti změn implementace, jelikož se nepředpokládá změna v protokolu ze strany týmu *Škoda Digital*.

3.4.2 Parametry implementace

Systém tramvaje odesílá údaje vypsané v tabulce 3.5, bohužel v průběhu testů nebyla ještě zprovozněna velká část měření na straně týmu *Škoda Digital*. Například rozlišení pro hodnoty `speed_VCU` a `force` nejsou známy.

Perioda datového přenosu je 20 ms, což odpovídá frekvenci 50 Hz. Tato frekvence je fixně daná vnitřním nastavením systému vozidla a není možné tuto frekvenci měnit.

3.4.3 Testování implementace

Testování probíhalo již na podzim roku 2022, a jelikož se od té doby v protokolu nic nezměnilo, lze předpokládat, že komunikace s tramvají bude probíhat bez problémů i při dalších pokusech.

Bohužel se v roce 2023 nepodařilo provést test při jízdě tramvají, jelikož tramvaje vybavené požadovanými přístroji nejsou běžně v provozu. Byly provedeny vnitřní testy připojení pomocí `localhost`. Tyto testy byly úspěšné a nebyly nalezeny žádné problémy.

3.5 Controller Area Network (CAN bus)

3.5.1 Požadavky

Pro komunikaci s dalšími vozidly hromadné dopravy byl vznesen požadavek na schopnost přijímat zprávy skrze rozhraní `CAN`. Jelikož nebyly vzneseny konkrétní požadavky na komunikaci, bylo nutné provést jakoukoliv implementaci co nejgeneričtěji.

3.5.2 Přehled komunikace

Předpokládá se, že komunikace bude probíhat skrze externí mikrokontroler *STM32*. Tyto mikrokontrolery a rozšířeně také desky *Nucleo* poskytují rozhraní `CAN` jakožto jeden ze základních připojení. Pro Připojení je však třeba převést hodnoty signálů na hodnoty snesitelné pro samotný mikrokontroler. K tomu slouží transceivery jako například deska *MCP2551* os společnosti *Microchip*. Tento konkrétní transceiver poskytuje pro nás dostatečnou rychlost 1 MB/s.

`CAN` rozhraní pro *STM32* umožňuje komunikaci s využitím přerušení, což značně urychluje odesílání dat do centrálního počítače. Při použití snížené frekvence IMU by mělo být možné zkombinovat přijímače IMU a `CAN` v jednom mikrokontroleru. Pokud by se použilo *USB–HS*, bylo by možné zkombinovat obě komunikace za plného výkonu.

3.6 Nevyužité senzory

V rámci vývoje bylo vytvořeno několik senzorů s implementacemi uloženými v repozitářích práce. Jako první senzor rychlosti byl použit magnetický senzor pro jízdní kolo. Oproti běžnému magnetickému tachometru měl 4 magnety pro vyšší přesnost odhadu rychlosti. Implementace byla provedena s využitím čítačů signálu.

Při předchozích pokusech byl testován GNSS senzor *ZED–F9R* a *ZED–F9K*. Tyto moduly by měly dle výrobce umožňovat fúzi senzorů přímo v modulu. Oba moduly integrují vlastní jednotku IMU a je možné do nich přivádět hodnoty odometrie z vnějších senzorů jak z podobě zpráv, tak jako analogové pulsy.

Bohužel se při podzimních experimentech ukázaly jako nepraktické pro aplikace na kolejových vozidlech. Jejich vnitřní algoritmy jsou vytvořeny pro pohyb menších vozidel. Nejpodobnější vozidlo v nabídce algoritmů je pro osobní automobil. Dalším problémem je neviditelnost vnitřních stavů modulu. Vnitřní algoritmy jsou vytvořené, aby podporovaly plug–and–play režim, tudíž neposkytují žádná další nastavení a neexistuje ani popis jejich vnitřního fungování.

Při pohybu v kolejových vozidlech tyto algoritmy předpokládaly chybu a zcela ignorovaly údaje z vnější odometrie. Zároveň procesor modulu není dostatečně rychlý a celý algoritmus je značně nestabilní, zejména při vyšších frekvencích příjmu dat.

Kapitola 4

Intraplatformní komunikace

Při sběru dat ze senzorů postačuje implementace separátních programů pro čtení a ukládání dat. Pokud tato data chceme v reálném čase přeposílat do algoritmu, je nutné mít k dispozici prostředí, které je schopné jednotlivé procesy propojit. Vzhledem k neznámé topologii konkrétního experimentu (např. nahrazení `TRDP` senzorem `ODB2`), není možné využít běžné metody multiprocesové komunikace, jako jsou sdílené proměnné a mutexové zámky.

Klasickou odpovědí na tento problém je *ROS – Robot Operating System* [20]. Je dostupný ve dvou implementacích jakožto *ROS* a *ROS 2*. Tyto implementace spolu mají společné pouze jméno, vnitřní systém je zcela odlišný. Další možností je *eCAL – enhanced Communication Abstraction Layer* [8] od inicitativy *Eclipse Foundation*.

4.1 Výběr rozhraní

4.1.1 Požadavky

Hlavním požadavkem je jednoduchost použití. Jelikož samotná data ze senzorů nemají extrémně vysokou frekvenci ani velikost (max. 2000 Hz / 64 B pro jednotku IMU), není třeba se příliš zaměřovat na rychlost

Dalším požadavkem je integrace s prosředím *Matlab* a *Simulink*, a to nejlépe formou oficiální podpory ze strany společnosti *MathWorks*. Nutností je také možnost distribuovatelnosti, aby byla zachována možnost algoritmus spustit na vlastním zařízení se sdílením zpráv skrze síťová rozhraní.

Nutností je také kompatibilita s oběma hlavními operačními systémy *Windows* a *Linux*, podpora *macOS* není považována za nutnou, je ale je vítaným zlepšením.

4.1.2 Přehled parametrů

ROS

ROS je middleware systém původně vydaný v roce 2007, je využívaný zejména v robotice a autonomním řízení. Jedná se o spolehlivou metodu přenosu dat

mezi procesy. Jedná se však o starší software a některé jeho části nejsou uzpůsobeny moderním požadavkům.

Systém podporuje Unixové operační systémy jako Linux, ale jeho podpora operačního systému Windows je pouze experimentální. Tato experimentální verze je plně funkční pro naše potřeby, je však praktičtější pracovat na systémech s plnou podporou.

Další slabinou *ROS* je nemožnost budoucí integrace do realtime algoritmů, samotný middleware nebyl plánován pro tato použití a jeho adaptace na vlastnosti, pro tyto aplikace požadované, je náročná a neefektivní.

Instalace prostředí *ROS* pro *Debian*–based systémy je přímočará a popsaná v dokumentaci. Co se týče instalace pro ostatní Linuxové platformy, je skvěle popsaná instalace pro *Arch*–based systémy. Instalace pro Windows má sice kvalitní dokumentaci, ale i přesto může být komplikovaná a nemusí se podařit, zejména s ohledem na verze jazyka *Python*, které nejsou vždy zpětně kompatibilní.

Integrace do prostředí *Matlab/Simulink* je jednoduchá a postačuje k ní oficiální toolbox. Problémy mohou nastat při verzování jazyka *Python*, je proto nutné dbát na přesný postup popsaný na stránkách *MathWorks*. Nejjednodušší metodou pro vyhnutí se komplikacím je zvolit čistou instalaci celého operačního systému.

ROS 2

ROS 2 je middleware systém z roku 2017, z vnějšího pohledu se jedná o kopii systému *ROS*, vnitřní systémy jsou zcela odlišné. Oba systémy jsou schopné interoperability, tato funkce ale zpožďuje přenos dat a komplikuje vývoj.

Jelikož první *LTS* verze byla vydána až v roce 2020, jedná se o poměrně nový systém, který má stále v pokročilých aspektech problémy se stabilitou. Pro základní aplikace využívané pro naše požadavky je ale systém plně stabilní a počítá se s podporou do budoucna.

ROS 2 narozdíl od svého předchůdce zaručuje podporu pro realtimové aplikace, byl to hlavní důvod pro tvorbu kompletně nového prostředí. Kompatibilita tohoto systému je podrobně popsána v tabulce 4.1.

Při instalaci `ROS 2` pro integraci do prostředí *Matlab* je třeba vzít v úvahu odlišné požadavky obou aplikací na verzi jazyka *Python*. Zatímco *ROS 2* ve verzi *Foxy* požaduje *Python 3.8.x*, *ROS Toolbox* pro Matlab vyžaduje verzi *Python 3.9.x*. Tyto dvě verze nejsou vzájemně kompatibilní. Pro instalaci na *Windows 10* bude nutné nainstalovat obě verze. Pro instalaci na *Ubuntu* bude nutné zvolit verzi *Ubuntu 20.04*, která nativně obsahuje *Python 3.8.x* a pro potřeby toolboxu doinstalovat verzi *Python 3.9.x*. *MathWorks* není schopen zaručit kompatibilitu s *Ubuntu 22.04*. Pro *Matlab 2023a* je ale možné využít nativní *Python 3.8.x* bez nutnosti další instalace.

Pro instalaci verze *Humble* nelze počítat s podporou *MathWorks*. Jelikož verzi *Foxy* končí podpora v květnu 2023, nebude mít *ROS Toolbox* minimálně do verze *Matlab 2023b* podporu pro aktivní verzi *ROS 2*. To by nemělo způsobovat větší problémy, jedná se však o komplikaci pro instalaci a spolehlivost vývoje.

operační systém	verze *Foxy*	verze *Humble*
Ubuntu 20.04 – `x84-64`	ano	komp. ze zdroje
Ubuntu 20.04 – `ARM-64`	ano	komp. ze zdroje
Ubuntu 20.04 – `ARM32`	komp. ze zdroje	komp. ze zdroje
Ubuntu 22.04 – `x84-64`	ne	ano
Ubuntu 22.04 – `ARM-64`	ne	ano
Ubuntu 22.04 – `ARM-32`	ne	komp. ze zdroje
Debian 10/11 – všechny arch.	komp. ze zdroje	komp. ze zdroje
RHEL 8 – `x86-64`	komp. ze zdroje	ano
Windows 10 – `x86-64`	ano	ano
macOS Mojave – `x86-64`	ano	komp. ze zdroje

Tabulka 4.1: Kompatibilita verzí *ROS 2* s operačními systémy a platformami.

eCAL

eCAL od iniciativy *Eclipse Foundation* je middleware podobný *ROS 2*. Jedná se o brokerless systém, který umožňuje komunikaci mezi procesy na úrovni binárních zpráv. Jelikož nezáleží na formátu binární zprávy, lze použít jakýkoliv systém pro serializaci. Jako doporučený se uvádí open–source systém *Protobuf* od společnosti *Google*.

Tento systém byl použit při experimentech z podzimu 2022. Jedná se o vysoce spolehlivý a rychlý systém, který kromě lokálního sdílení pomocí sdílené paměti umožňuje také síťové sdílení zpráv. Ve srovnání s *ROS 2* se jedná o odlehčený a rychlejší systém vhodný pro odesílání velkého objemu dat. Velkou výhodou je také možnost navázání zpráv na *ROS 2* a komunikace s tímto systémem. Dalším plusem je také možnost odesílání více typů zpráv v jednom kanálu bez vzájemného rušení.

Největší slabinou tohoto systému je slabá podpora. Pro *MATLAB* je k dispozici pouze neoficiální knihovna, která funguje velmi dobře a je jednodušší na integraci než *ROS Toolbox*, ale postrádá jednoduchost rozhraní plnohodnotného toolboxu. Zároveň je *eCAL* vytvářen relativně malým týmem a nelze zaručit podporu do budoucna. Již zmíněné propojení s *ROS 2* je momentálně funkční, ale momentálně bez plnohodnotného vývojářského týmu.

Knihovna pro *MATLAB* umožňuje komunikaci pouze ve formátu `std::`, tudíž dokáže interpretovat např. hodnoty `std::String`, nedokáže však pracovat s komplikovanějšími strukturami, je proto nutné manuálně vytvořit deserializátor. Alternativou je využít neoficiální knihovny pro *Protobuf* [7]. Přesto se však jedná o komplikaci.

Instalace je výrazně jednodušší než v případě *ROS*/*ROS 2* jelikož *Eclipse* poskytuje instalační balíky pro *Windows* a *Ubuntu 20.04*/*22.04*. Testy proběhly na všech zmíněných platformách (pro *Windows* ve verzích *10* a *11*) pro jazyky *Python* a *C++*. Co se týče verzí jazyka *Python* jsou podporovány všechny verze od *Python 3.6.x* po *Python 3.11.x* a nové verze jsou rychle doplňovány.

4.1.3 Rozhodnutí

Vzhledem k tomu, že *ROS 2* téměř přímým vylepšením *ROS*, lze druhou možnost téměř okamžitě vyřadit, jelikož nemusíme řešit zpětnou kompatibilitu. Při srovnání systémů *eCAL* a *ROS 2* vyplývá, že pro dlouhodobě využívané systémy by byl vhodnější *eCAL*. Vývoj na této platformě je jednodušší, a zároveň má lepší parametry rychlosti a jednodušší integraci do prostředí. Velká výhoda *ROS 2* je obecné povědomí a lepší dokumentace. Je vyšší pravděpodobnost že uživatel bude obeznámen s fungováním systému *ROS 2*, který je využívaný v průmyslu i ve vzdělávacích institucích.

Podpora ze strany *MathWorks* je zde také určujícím parametrem, jelikož *eCAL* má prakticky nulovou oficiální podporu. Podpora pro *ROS 2* je přinejlepším komplikovaná, ale existuje možnost že se bude s časem zlepšovat. *eCAL* má sice podporu od mnoha společností účastnících se iniciativy *Eclipse Automotive* pod kterou *eCAL* spadá, mnohé z těchto firem ale přesto využívají *ROS*/*ROS 2*.

Z těchto důvodů jsem se rozhodl pro *ROS 2*, pro zjednodušení instalace byl vytvořen skript, který provede kompletní instalaci pro *Ubuntu 20.04*. Pro *Windows 10*/*11* je v dodatku B.

Kapitola 5

Integrace s prostředím Matlab a Simulink

Vzhledem k nejčastějšímu způsobu práce s řídícími algoritmy na škole bylo jako cíl integrace zvoleno prostředí *Matlab*. Jelikož máme k dispozici prostředí *ROS 2*, je možné přeposílat data v reálném čase.

5.1 Integrace s prostředím ROS 2

Jako primární způsob byl určený oficiální *ROS Toolbox*, který podporuje jak *ROS*, tak *ROS 2*. Oficiálně je komunikace podporována pro interpretovaný i kompilovaný program. Při práci v prostředí *Simulink*, je komunikace abstrahovaná na `Subscribe` a `Publish` bloky, které mají jako vstup/výstup `bus` signálů z každé zprávy jako na obr. 5.1. Nastavení bloku `Subscribe` je zobrazeno na obr. 5.2.

Před samotnou instalací je nutné nainstalovat námi využívané zprávy do prostředí *Matlab*. K tomu lze využít příkazy

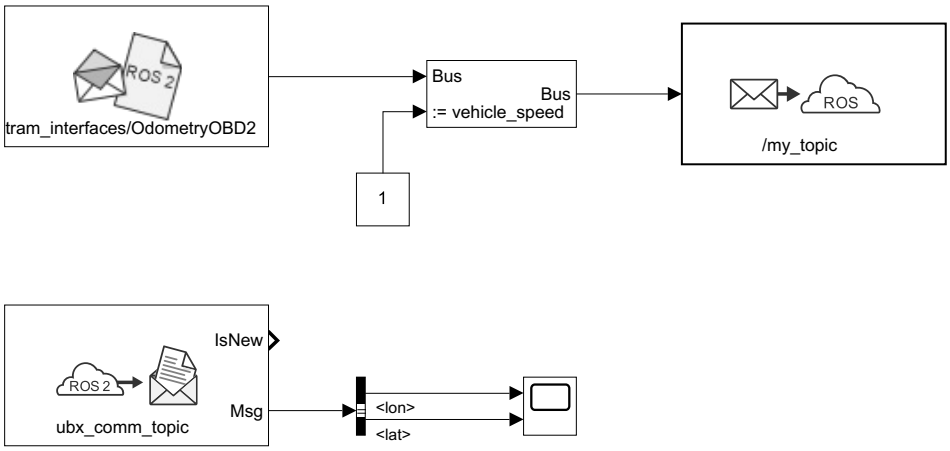
```
rosinit();
ros2genmsg("path_to_folder");
```

Ty v základním adresáři (např. `C:\Users\usr\Documents\MATLAB` pro *Windows*) vytvoří soubory pro interpretaci a pozdější kompilaci zpráv. Pro tento projekt jsou samotné zprávy vytvořeny v repozitáři programu.

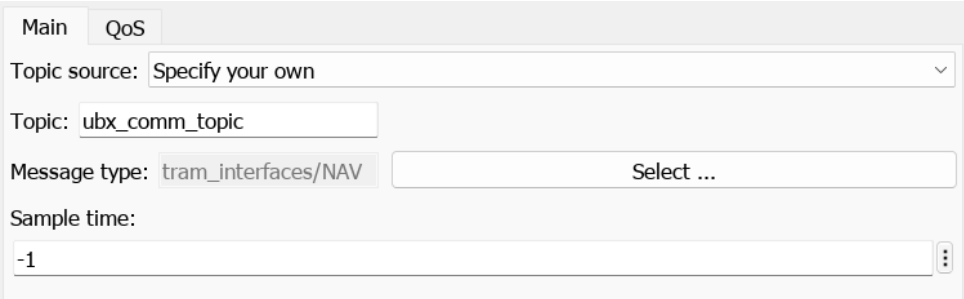
Tento postup vyžaduje kompletní instalaci prostředí *ROS 2* jak je popsáno v kapitole 4. Pro *Windows* je k dispozici instalační balík popsaný v dodatku B. Tato instalace je postačující pro kompletní práci v prostředí *Matlab*, které značně zjednodušuje interakci s tímto prostředím. Zejména pro *Windows* se tak uživatel může vyhnout mnoha chybám. Na *UNIX*–based systémech je tato práce výrazně jednodušší.

5.2 Kompilace pro externí zařízení

Oproti dřívějším verzím *Matlab* doplnil funkční kompilační target pro zařízení se systémy `Linux`. Tato verze vypadá plně funkční, včetně multiprocessingu a



Obrázek 5.1: *ROS 2* bloky v prostředí *Simulink*. Horní část zobrazuje odeslání zprávy, kdy se přiřazuje hodnota proměnné, spodní část zobrazuje přijetí bloku a vybrání hodnot pro zobrazení.



Obrázek 5.2: Parametry `Subscribe` bloku v prostředí *Simulink*.

připojení konektorů USB, což byly hlavní chyby v předchozích verzích, které se řešily custom targety jako je např. `ert_linux` vytvořený přímo na ČVUT.

Jedním z možných vysvětlení může být, že *ROS Toolbox* využívá vlastní target, který nevychází z klasického `grt.tlc` (Generic Real–Time Target), ale podobně jako `ert_linux` z `ert.tlc`, z balíku *Embedded Coder*. Jako možnost se tedy naskytuje využít právě tento target, i přesto, že nakonec nemusí využívat *ROS*.

Kapitola 6

Práce s mapovými podklady

Jako součást práce bylo zvolení vhodné interakce s mapovými podklady. Pro základ by bylo vhodné využít podklady z pražského Geoportálu a databáze *Open Street Maps*. Práci s *OSM* zastane *Mapping Toolbox*, větším problémem je převod do českého geodetického systému *S–JTSK*. V něm je uložena většina podrobnějších dat pro Českou republiku. *Mapping Toolbox* poskytuje základní převod, implementace však není kompletní a chybí jí konečná dotransformace vycházející z původní nepřesnosti zavedení tohoto systému.

6.1 Převod mezi systémy *WGS84* a *S-JTSK*

Pro převod jsem vytvořil implementaci pro *Matlab*, tato implementace je zároveň propojena s výškovou mapou `https://gitlab.fel.cvut.cz/krizmat3/matlab_prague_height_map`. Tento převod je vhodný i pro další projekty, jelikož je nezávislý na našich požadavcích.

6.1.1 Postup převodu

Postup převodu je komplikovaný a jelikož není k dohledání mnoho zdrojů pro jeho implementaci uvedu konkrétní postup zde. Jedná se o Křovákovo zobrazení neboli kuželové zobrazení.

Ve výpočtech se pracuje se základními konstantami [32] [28]:

$$
\begin{aligned}
a_{\text{wgs84}} &= 6378137.0\,\text{m}\ , \\
f^{-1}_{\text{wgs84}} &= 298.257223563\ , \\
a_{\text{bessel}} &= 6377397.15508\,\text{m}\ , \\
f^{-1}_{\text{bessel}} &= 299.152812853\ ,
\end{aligned}
$$

$$
\begin{aligned}
e_{\text{wgs84}} &= \sqrt{1-(1-f_{wgs84})^2}\ , \\
e_{\text{bessel}} &= \sqrt{1-(1-f_{bessel})^2}\ ,
\end{aligned}
$$

kde a je hlavní poloosa elipsoidu, f zploštění a e excentricita, `wgs84` a `bessel` určují, zda se jedná o elipsoid definovaný pro systém *WGS84* nebo se jedná o Besselův elipsoid z roku 1841. Ten se využívá při definici *S–JTSK*.

Vycházet budeme z hodnot

ϕ_{wgs84}	zeměpisná šířka v systému v radiánech *WGS84* (latitude),
θ_{wgs84}	zeměpisná délka v systému v radiánech *WGS84* (longitude),
A_{wgs84}	výška nad referenčním elipsoidem v metrech *WGS84* (altitude).

Tyto souřadnice potřebujeme nejprve převést na pravoúhlé ECEF (Earth–Centered Earth–Fixed), abychom mohli provést posun středu elipsoidů a rotaci souřadných systémů.

Pro další výpočet potřebujeme poloměr křivosti ve směru šířky

$$\kappa_{\phi_{\text{wgs84}}} = \frac{a_{\text{wgs84}}}{\sqrt{1 - e_{\text{wgs84}} \sin(\phi_{\text{wgs84}})^2}}$$

Polohu pak stanovíme jako

$$\vec{r_{\text{wgs84}}} = \begin{pmatrix} (\kappa_{\phi_{\text{wgs84}}} + A) \cos(\phi_{\text{wgs84}}) \cos(\theta_{\text{wgs84}}) \\ (\kappa_{\phi_{\text{wgs84}}} + A) \cos(\phi_{\text{wgs84}}) \sin(\theta_{\text{wgs84}}) \\ ((1 - e^2_{\text{wgs84}})\, \kappa_{\phi_{\text{wgs84}}} + A) \sin(\phi_{\text{wgs84}}) \end{pmatrix} .$$

Dále je třeba provést posun středu a rotaci. Posun se provede o vektor [12]

$$\vec{r_0} = \begin{pmatrix} -533.230 \\ -75.375 \\ -452.045 \end{pmatrix} .$$

Jako další konstanty budeme potřebovat pootočení $\vec{\omega}$ a měřítko transformace m.

$$m = -8.750 \times 10^{-6}$$

$$\begin{aligned} \omega_x &= 2.6733 \times 10^{-6}\,\text{rad} , \\ \omega_y &= 1.1980 \times 10^{-6}\,\text{rad} , \\ \omega_z &= 2.9646 \times 10^{-6}\,\text{rad} , \end{aligned}$$

Pro rotaci můžeme matici R aproximovat jako

$$R = \begin{bmatrix} 1 & \omega_z & -\omega_y \\ -\omega_z & 1 & \omega_x \\ \omega_y & -\omega_x & 1 \end{bmatrix} .$$

V implementaci je provedena kompletní a neaproximovaná rotace.

Výsledná transformace má tvar

$$\vec{r_{\text{sjtsk}}} = \vec{r_{\text{wgs84}}} + (1 + m)\, R \times \vec{r_0} .$$

Pro převod na souřadnice ϕ_{sjtsk}, θ_{sjtsk} a A_{sjtsk} je nutné využít iteraci.

Zvolíme hodnotu

$$t_0 = \frac{r_{\mathrm{sjtsk}\,z}}{(1 - e^2_{\mathrm{bessel}})\,p} \ ,$$

kde

$$p = \sqrt{r^2_{\mathrm{sjtsk}\,x} + r^2_{\mathrm{sjtsk}\,y}} \ .$$

Poté iterace probíhá jako

$$t_i = \frac{r_{\mathrm{sjtsk}\,z}}{p - \frac{a_{\mathrm{bessel}}\,e^2_{\mathrm{bessel}}}{\sqrt{1+(1-e^2_{\mathrm{bessel}})t^2_{i-1}}}} \ .$$

Pro přesnou iteraci se doporučuje kontrolovat změny hodnoty t, pro naší aplikaci však s jistotou postačuje 12–20 iterací.

Z tohoto výsledku přesně určíme geodetické souřadnice jako

$$\phi_{\mathrm{sjtsk}} = \arctan(t) \ ,$$

$$\theta_{\mathrm{sjtsk}} = 2\,\arctan\left(\frac{r_{\mathrm{sjtsk}\,y}}{r_{\mathrm{sjtsk}\,x} + p}\right) \ ,$$

$$A_{\mathrm{sjtsk}} = \sqrt{1+t^2}\left(p - \frac{a_{\mathrm{bessel}}}{\sqrt{1+(1-e^2_{\mathrm{bessel}})\,t^2}}\right) \ .$$

Závěrečná část převádí tyto souřadnice na rovinné souřadnice Y, X a H.

Pro výpočty je nutná zvolená střední zeměpisná šířka a jí odpovídající kulová šířka

$$\theta_0 = 49.5^\circ = 0.863938\,\mathrm{rad}\,.$$

Z ní jsou vypočteny konstanty

$$\alpha = \sqrt{1 + \frac{e^2_{\mathrm{bessel}}}{1 - e^2_{\mathrm{bessel}}}\,\cos(\theta_0)} \approxeq 1.000597498371542 \ ,$$

$$U_0 = \arcsin\left(\frac{\sin(\theta_0)}{\alpha}\right) \approxeq 0.863239102658489\,\mathrm{rad} \ ,$$

$$k = \tan\left(\frac{U_0}{2} + \frac{\pi}{4}\right)^{-1}\left[\tan\left(\frac{\theta_0}{2} + \frac{\pi}{4}\right)\left(\frac{1 + e_{\mathrm{bessel}}\,\sin(\theta_0)}{1 - e_{\mathrm{bessel}}\,\sin(\theta_0)}\right)^{\frac{e_{\mathrm{bessel}}}{2}}\right]^{\alpha} \approxeq 1.003419163966575 \ ,$$

kde U_0 je kulová šířka.

Pro další část výpočtu je třeba znát sférickou šířku U a délku V

$$U = 2\,\arctan\left[k\left(\tan\left(\frac{\theta_{\mathrm{sjtsk}}}{2} + \frac{\pi}{4}\right)\left(\frac{1 - e_{\mathrm{bessel}}\,\sin(\theta_{\mathrm{sjtsk}})}{1 + e_{\mathrm{bessel}}\,\sin(\theta_{\mathrm{sjtsk}})}\right)^{\frac{e_{\mathrm{bessel}}}{2}}\right)^{\alpha}\right] - \frac{\pi}{2} \ ,$$

$$V = \alpha\left(\phi + \frac{53^\circ}{3}\right) \ , \frac{53^\circ}{3} \approxeq 0.308341501185665\,\mathrm{rad}\,.$$

Nyní je třeba převést na kartografické souřadnice S a D pomocí Gaussovy koule s pólem na souřadnicích

$$\psi_Q = 48.25^\circ \approxeq 0.842121364087264\,\mathrm{rad}\ ,$$
$$\theta_Q = 42.5^\circ \approxeq 0.741764932097590\,\mathrm{rad}\ ,$$

kde sférická šířka je s posunutím o $U_{Q0} = 11.5^\circ \approxeq 0.200712863979348\,\mathrm{rad}$ je

$$U_Q = 2\,\arctan\left[k\left(\tan\left(\frac{\theta_Q}{2}+\frac{\pi}{4}\right)\left(\frac{1-e_{\mathrm{bessel}}\,\sin\left(\theta_Q\right)}{1+e_{\mathrm{bessel}}\,\sin\left(\theta_Q\right)}\right)^{\frac{e_{\mathrm{bessel}}}{2}}\right)^{\alpha}\right] - \frac{\pi}{2} + U_{Q0} \approxeq 1.042168563797517\,\mathrm{rad}\ .$$

Délkový rozdíl mezi šířkou Q a šířkou transformovaného bodu je

$$\Delta V = V_Q - V = \alpha\,\theta_Q - V\ .$$

Konformní tečný kužel má základní šířku

$$S_0 = 78.5^\circ \approxeq 1.370083462815549\,\mathrm{rad}\ .$$

K redukci zkreslení se zmenšuje poloměr s poměrem 0.9999. K výpočtu potřebujeme následující konstanty

$$n = \sin\left(S_0\right) = \approxeq 0.979924704620830\ ,$$

$$\rho_0 = 0.999\,a_{\mathrm{bessel}}\,\frac{\sqrt{1-e_{\mathrm{bessel}}^2}}{\tan\left(S_0\right)\left(1-e_{\mathrm{bessel}}^2\,\sin\left(\theta_0\right)^2\right)} \approxeq 1298039.004638987\,\mathrm{m}\ .$$

Polární souřadnice pak vypočteme jako

$$\rho = \rho_0\left(\frac{\tan\left(\frac{S_0}{2}+\frac{\pi}{4}\right)}{\tan\left(\frac{S}{2}+\frac{\pi}{4}\right)}\right)^n\ ,$$
$$\varepsilon = n\,D\ .$$

Z nich už snadno dostaneme výsledné rovinné pravoúhlé souřadnice

$$Y = \rho\,\sin\left(\varepsilon\right)\ ,$$
$$X = \rho\,\cos\left(\varepsilon\right)\ ,$$
$$H = A_{\mathrm{sjtsk}}\ .$$

6.1.2 Chyby převodu

Pro větší přiblížení kartografickému systému se využívá konečná dotransformace s deseti konstantami [13]

$$\begin{array}{cc} a_1 & 0.2946529277 \times 10^{-1} \\ a_2 & 0.2515965696 \times 10^{-1} \\ a_3 & 0.1193845912 \times 10^{-6} \\ a_4 & -0.4668270147 \times 10^{-6} \\ a_5 & 0.9233980362 \times 10^{-11} \\ a_6 & 0.1523735715 \times 10^{-11} \\ a_7 & 0.1696780024 \times 10^{-17} \\ a_8 & 0.4408314235 \times 10^{-17} \\ a_9 & -0.8331083518 \times 10^{-23} \\ a_{10} & -0.3689471323 \times 10^{-23} \end{array}$$

pro dotransformaci vycházíme z posunutých bodů

$$Y_{\mathrm{red}} = Y - 654000.0 \ ,$$

$$X_{\mathrm{red}} = X - 1089000.0 \ .$$

z nichž získáme

$$\begin{aligned} \Delta Y = a_2 + X_{\mathrm{red}}\, a_4 + Y_{\mathrm{red}}\, a_3 + a_9 \left({X_{\mathrm{red}}}^4 - 6\, {X_{\mathrm{red}}}^2\, {Y_{\mathrm{red}}}^2 + {Y_{\mathrm{red}}}^4 \right) + \\ a_6 \left({X_{\mathrm{red}}}^2 + {Y_{\mathrm{red}}}^2 \right) + X_{\mathrm{red}}\, a_8 \left({X_{\mathrm{red}}}^2 - 3\, {Y_{\mathrm{red}}}^2 \right) + 2\, X_{\mathrm{red}}\, Y_{\mathrm{red}}\, a_5 + \\ Y_{\mathrm{red}}\, a_7 \left(3\, {X_{\mathrm{red}}}^2 - {Y_{\mathrm{red}}}^2 \right) - 4\, X_{\mathrm{red}}\, Y_{\mathrm{red}}\, a_{10} \left({X_{\mathrm{red}}}^2 - {Y_{\mathrm{red}}}^2 \right) \ , \end{aligned}$$

$$\begin{aligned} \Delta X = a_1 + X_{\mathrm{red}}\, a_3 - Y_{\mathrm{red}}\, a_4 - a_8 \left(3\, {X_{\mathrm{red}}}^2 - {Y_{\mathrm{red}}}^2 \right) + \\ a_{10} \left({X_{\mathrm{red}}}^4 - 6\, {X_{\mathrm{red}}}^2\, {Y_{\mathrm{red}}}^2 + {Y_{\mathrm{red}}}^4 \right) + a_5 \left({X_{\mathrm{red}}}^2 + {Y_{\mathrm{red}}}^2 \right) + \\ X_{\mathrm{red}}\, a_7 \left({X_{\mathrm{red}}}^2 - 3\, {Y_{\mathrm{red}}}^2 \right) - 2\, X_{\mathrm{red}}\, Y_{\mathrm{red}}\, a_6 + 4\, X_{\mathrm{red}}\, Y_{\mathrm{red}}\, a_9 \left({X_{\mathrm{red}}}^2 - {Y_{\mathrm{red}}}^2 \right) \ . \end{aligned}$$

Ty poté použijeme pro data s korekcí

$$Y_{\mathrm{kor}} = Y + \Delta Y \ ,$$

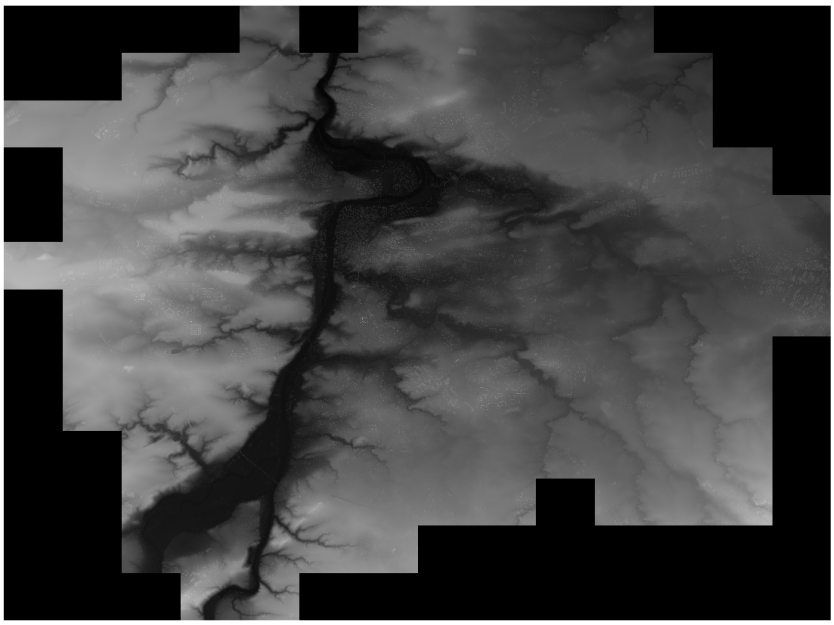
$$X_{\mathrm{kor}} = X + \Delta X \ .$$

Tato korekce poskytuje zpřesnění s podkladovými materiály řádově v desítkách cm. Pro Prahu se jedná o zpřesnění v rozsahu 15 cm – 25 cm.

Převod má bohužel stále odchylku zejména v souřadnici Y, toto pravděpodobně lze přičíst vzájemnému posunu souřadných systémů bylo by tedy vhodné aktualizovat převodní konstanty *WGS84* a evropských souřadných systémů.

6.2 Zdroje mapových podkladů

Pro primární odhad k pozdější práci z výškovými daty byla využita databáze geoportálu hl. města Prahy [10]. Pro Prahu a blízké okolí poskytuje výškovou mapu s rozlišením 1 m a deklarovanou přesností 0.25 m. Mapa je zobrazena na obr. 6.1. Vzhledem k velikosti byl přepracován soubor z formátu `.tif` na formát `.jpg`. Způsob čtení a práce se souborem je uveden ve veřejném repozitáři `https://gitlab.fel.cvut.cz/krizmat3/matlab_prague_height_map`. Velikost dat byla redukována z 1800 MB na 600 MB. Součástí repozitáře je také převod ze souřadnic *WGS84*.



Obrázek 6.1: Výšková mapa Prahy [10]

Kapitola 7

Struktura dat

V rámci standardizace dat bylo úkolem také vytvořit strukturu pro ukládání dat z experimentu do repozitářů. Struktura má kombinovat dobrou čitelnost pro stroje i pro člověka a využívat běžné formáty, aby nebylo nutné doplňovat kompatibilitu pro další jazyky.

Pro samotná data se naskytovaly dvě možnosti souborů, `.csv` a `.h5` (`HDF`). `HDF` jsou založené na binárním ukládání, zatímco `.csv` jsou textové soubory. Jelikož nesbíráme žádná příliš rozsáhlá data, není třeba řešit velikost souborů. Objemy dat v souborech `.csv` se pohybují v desítkách MB za hodinu. `.csv` soubory nabízejí jednodušší práci a je možné skrze ně přímo kontrolovat průběh pouze s textovým editorem. Tudíž jsem zvolil `.csv` jako primární formát pro ukládání. Pokud by bylo nutné data komprimovat, je možné je převést na jiné formáty. Případným řešením může být také zabalení na adresáře `.zip` nebo `.7z`, které mají schopnost zmenšit tento typ dat až o 95 %.

Pro informace o experimentu, která jsou strojově čitelná byl zvolen formát `.toml`, který dobře kombinuje strojovou a lidskou čitelnost.

Samotná struktura adresáře má formát

```
2023-01-01_00
  | info.toml
  | sensor_data
  | | sensors.toml
  | | s00
  | | | parsed.csv
  | | | raw.rawdata
  | | s01
  | | | parsed.csv
  | | | raw.rawdata
  | compiled_data
  | | compilations.toml
  | | c00
  | | | data.csv
  | | | data.mat
  | analysis
  | | analysis.toml
```

```
| | a00
| | | results...
```

Název adresáře určuje rok (`yyyy`), měsíc (`mm`), den (`dd`) a číslo experimentu (`nn`) jako `/yyyy-mm-dd_nn/`. Kompilace znamená základní zpracování dat jako je např. časová synchronizace. Analýza poté ukládá konečný výsledek (např. ve formátu `.pdf`).

V tomto případě by soubory vypadaly takto

`2023-01-01_00/info.toml`

```
experiment_name = "name"

participants = [
  [ "Jmeno Prijmeni", "prijjme@fel.cvut.cz"],
]
description ="""Descriprion of experiment."""
```

`2023-01-01_00/sensor_data/sensors.toml`

```
[sensorname1]
  clock = 0
  type = "sensor_type1"
  description = """popis senzoru"""
[sensorname1]
  clock = 0
  type = "sensor_type2"
  description = """popis senzoru"""
```

Kde hodnota `clock` odlišuje, zda byly senzory připojeny ke stejnému přístroji/zdroji času.

`2023-01-01_00/sensor_data/s00/parsed.csv`

```
timestamp_pc,speed,alt,
s,m/s,m,
1672527600.00001,0.69,4.20,
...
```

V `.csv` souboru první řádek určuje pro soubor unikátní název veličiny a druhý řádek určuje jeho jednotku pro usnadnění pozdější práce se souborem. Soubory `.rawdata` nejsou nijak standardizovány, ale je doporučeno do `parsed.csv` převést veškeré hodnoty, nebo doplnit skript `convert.x` pro převod dat z `raw.rawdata` na `parsed.csv` a to v libovolném jazyce (např. *Matlab*, *Julia*, *Python*).

`2023-01-01_00/compiled_data/compilations.toml`

```
[compilationname1]
  date = "2023-01-01"
  authors = [
    ["Jmeno Prijmeni", "prijjme@fel.cvut.cz"]
  ]
  descriprion = """popis"""
```

Data v souboru `data.csv` by měla kopírovat strukturu naměřených dat, soubory `data.mat` jsou definováný klasickou skrukturou prostředí *Matlab*.

`2023-01-01_00/analysis/analysis.toml`

```
[analysisname1]
  date = "2023-01-01"
  authors = [
    ["Jmeno Prijmeni", "prijjme@fel.cvut.cz"]
  ]
  descriprion = """popis"""
```

Vnitřní struktura podadresáře `a00` již není definována.

Pro přečtení těchto dat v softwaru Matlab lze použít funkci

```
>> readtable("file_name.csv");
```

která tato data importuje jako tabulku, kde jsou data uložena jako sloupcové vektory indexovatelné pomocí jejich tagů.

Kapitola 8

Testování platformy

8.1 Popis použití

Platforma je vytvořena pro režim plug–and–play, tudíž není nutná žádná dodatečná instalace. Pro primární instalaci je při stažení kompletního repozitáře připraven skript pro čistou instalaci z nového operačního systému.

Předpokladem pro instalaci je pouze *Ubuntu 20.04*. Po stažení repozitáře se v repozitáři nachází spustitelný skript `./install/install_all.sh`, který nainstaluje všechny nutné balíky a vytvoří složky pro ukládání dat.

Dalším krokem je kompilace balíků *ROS 2*. Pro tento krok je třeba se přesunout do `./tram_ros2_workspace` kde se pomocí příkazu

```
$ rosinit
```

(alias vytvořený instalačním skriptem) inicializuje *ROS 2* a pomocí

```
$ colcon build
```

zkompilují skripty.

V tuto chvíli je všechno připraveno k inicializaci. Pro tu je třeba zkopírovat inicializační skripty pro spuštění v režimu `Systemd`, který spouští jednotlivé procesy při spuštění zařízení. Pro toto je ve složce `./install/auto_launch/` připraven skript `copy_all.sh`, který zkopíruje všechny inicializační skripty a přiřadí patřičná oprávnění.

Po spuštění jednotlivých procesů je třeba tyto procesty přiřadit do rozhraní `Systemd`. Toto se dělá příkazem `systemctl`. Aby nebylo nutné si dané příkazy pamatovat jsou opět k dispozici skripty v podložkách složky `./install/auto_launch/` pro jednotlivé komponenty. Skript `enable.sh` například přiřadí daný proces pro spuštění při startu (bez nutnosti přihlášení uživatele). `disable.sh` tento proces odstraní ze spouštěcí sekvence. Pro vypnutí všech procesů je k dispozici skript `disable_all.sh`. Skript `start.sh` spustí proces okamžitě bez nutnosti restartu.

Skripty jsou dostatečně robustní, aby se v případě chyby samy restartovaly. Není tedy nutné nijak zasahovat do zařízení v průběhu pokusu.

Parametry spuštění jsou k dispozici v podsložce `~/tram_data/config/`, `.toml` soubory obsahují konfiguraci všech parametrů. Měnitelné parametry jsou např. nastavení portu nebo názvy sítí *ROS*.

Výsledný záznam dat se nachází v podsložce `~/tram_data/log/`, kde jsou ve složkách organizovány záznamy podle času jejich zahájení. Pokud není senzor připojen k zahájení měření dojde po jeho připojení.

8.2 Připojení z vnějšího zařízení

Pro připojení z vnějšího zařízení pro sledování měření nebo pro spuštění algoritmu z nebo na jiném zařízení je nutné připojit obě zařízení skrze Ethernetový kabel. Pro jednoduchost zde popíšu proces, který je vhodný pro *Ubuntu* popřípadě *WSL Ubuntu*.

Pro instalaci *WSL* pro *Windows* je vhodné použít *WSL 1*, jehož instalace je popsána v dodatku B.

Další instalace je totožná pro *Ubuntu* i *WSL*. Lze použít instalaci popsanou na stránkách `https://docs.ros.org/en/foxy/` nebo skript využitý pro instalaci pro platformu popsaný výše.

Připojení k jednotné *ROS* síti by mělo být jednoduché a mělo by postačovat propojení obou zařízení kabelem. Zejména na *Windows* se vyskytují problémy, kdy je třeba na zařízení *Windows* spustit sdílení sítě s cílovou platformou a u té poté provést restart.

Pro sledování sítě je postup totožný jako pro instalaci skriptů. Místo inicializace do spouštěcí sekvence však po příkazech

```
$ rosinit
$ colcon build
```

musí spustit nové okno terminálu a použít příkazy

```
$ rosinit
$ . install/local_setup.bash
$ ros2 run network_display acc
```

Příkaz `acc` lze nahradit příkazy `nav`, `obd2` a `trdp` pro sledování živých dat ze akcelerometru, gnss, `OBD 2` a `TRDP` respektive.

8.3 Příklad postupu experimentu

K platformě připojíme všechny senzory. Pokud chceme připojit externí zařízení, je třeba ho připojit skrze ethernetový kabel před startem. Připojení ethernetového kabelu pro příjem `TRDP` je také nutné provést před startem.

Spustíme platformu startovacím tlačítkem na straně a čekáme 30 sekund, aby došlo ke spuštění systému. Pokud jsme v automobilu je možné nastartovat

motor. Pokud by byl motor nastartován před spuštěním, systém je také funkční.

Příjem dat můžeme ověřit skrze skripty pro sledování sítě zmíněné výše. Pro rychlou kontrolu je možné ověřit přímo senzory. Modul GNSS bliká při periodickém příjmu dat, deska pro IMU má podobnou funkci, senzor `OBD 2` má spuštěné zelené indikační světlo.

Pro vypnutí platformy se doporučuje vypnutí stejným tlačítkem jako při startu. Aby nedošlo k chybě, je nutné před dalším pokusem vypojit napájení. Zařízení *ODROID H3+* uvádí v dokumentaci nepravdivé informace a napájení na USB po vypnutí není nulové.

8.4 Práce se zařízením

Uvnitř je předinstalováno *Ubuntu 20.04* a *Matlab 2023a*. Grafické rozhraní je primárně vypnuté aby zbytečně neodebíralo výpočetní výkon. Pro jeho spuštění je třeba použít aliasový příkaz

```
$ gui_start
```

8.5 Spuštění algoritmu

Program lze nastavit pro autonomní spuštění podobně jako skripty pomocí systému `Systemd`. Pro nastavení tohoto postupu lze využít soubory původní instalace a pouze změnit názvy skriptů. Předpokládá se však živé spouštění. Je proto připraveno SSH při připojení ethernetového kabelu.

Po připojení SSH je možnost spustit zkompilovaný program, nebo použít *Matlab*. Pro spuštění *Matlab*u skrze terminál je připraven alias

```
$ matlab-terminal
```

Jeho spuštění vypadá takto

```
aa4cc@aa4cc-ODROID-H3:~$ matlab-terminal
MATLAB is selecting SOFTWARE OPENGL rendering.

            < M A T L A B (R) >
  Copyright 1984-2023 The MathWorks, Inc.
  R2023a (9.14.0.2206163) 64-bit (glnxa64)
            February 22, 2023

To get started, type doc.
For product information, visit www.mathworks.com.

>>
```

Pro spuštění *Simulink* modelu je třeba překopírovat soubor do zařízení a použít *Matlab* příkaz

```
>> sim('model', sim_time);
```

kde `model` je název modelu uložený v aktuální složce a `sim_time` je doba simulace.

Kompilace je lehce náročnější. Je třeba mít nainstalovaný *ROS 2*, *ROS Toolbox* a zkompilované *ROS 2* zprávy pro *Matlab*, jak bylo popsáno v kapitole 5.

V simulinkovém okně v záložce `Apps` se nachází položka `Robot Operating System (ROS)`, po jejím zvolení je nutné v dialogovém okně zvolit položku `ROS 2`. Nově se objeví záložka `ROS`, ve které je možnost přidat zařízení skrze SSH. Pokud práce probíhá přímo na zařízení, je možné zvolit `localhost`.

Skrze položku `Hardware Settings` je nutné změnit `Device Type` na `x86-64 (Linux 64)`, aby kompilace proběhla korektně. Solver je automaticky nastaven na `Fixed-step`, jelikož pro kompilované aplikace nelze využít jinou možnost. Nyní je možné vygenerovat kód a přímo z okna *Simulink* ho také spustit a řídit skrze dialogové okno `Control Panel`.

Pokud zůstanou zařízení připojená, je možné sledovat průběh přímo v zařízení `Scope`. Další možností je pak využít blok `To File` pro záznam hodnot.

Poměrně dobré zkušenosti mám s generováním kódu a ruční kompilací. Pokud se kód vygeneruje a poté se celý vygenerovaný adresář překopíruje do zařízení je možné použít klasické kompilační příkazy přímo v adresáři

```
$ rosinit
$ colcon build
$ ros2 run model
```

Je možné narazit na komplikace při ruční i automatické kompilaci, že *Matlab* správně nepřekopíruje data *ROS 2* zpráv. Poté je nutné ručně zkompilovat tyto zprávy jako je popsáno v kapitole 5 přímo na cílovém zařízení a vygenerovanou složku překopírovat do kompilačního adresáře `/ros2_ws/`.

8.6 Popis experimentů

Pro prezentaci schopností platformy jsem zvolil jízdu automobilem, která využívá senzory odometrie (`OBD 2`), inerciální měřící jednotky a GNSS. Topologie experimentu je zobrazena na obrázku 8.1. Jako příklad jsou uvedeny průběhy rychlostí ze senzorů `OBD 2` a GNSS na obrázku 8.3.

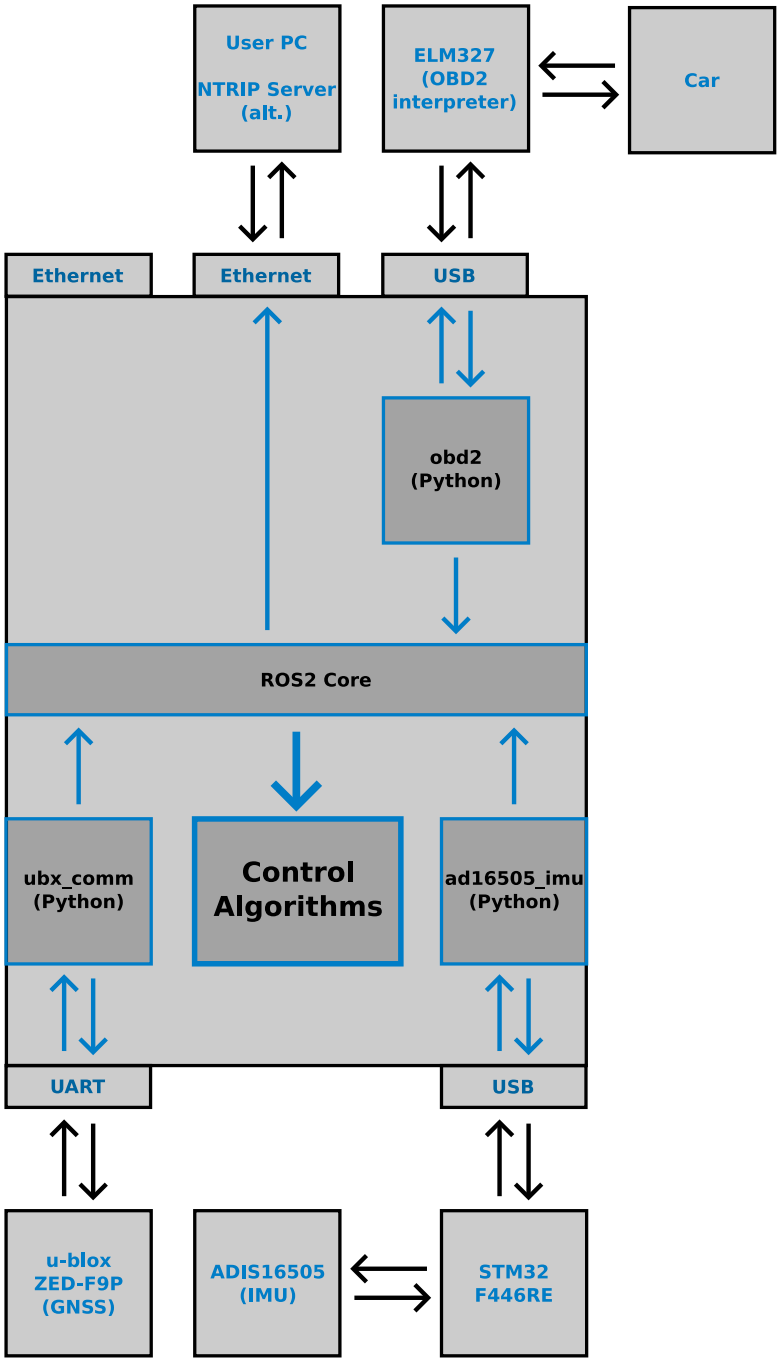
Z obrázku je patrné, že senzor `OBD 2` je přesný zejména v určení binárního stavu stacionární/pohybující se. Při rozjezdech se projevují odchylky od rychlosti naměřené pomocí GNSS. Hlavním zdrojem této odchylky je zpoždění. Toto zpoždění se po rozjedu sníží.

Zároveň je na obrázku patrná vyšší chyba kvantování z druhého zdroje, kde je kvantování dáno rozlišením senzoru $1\,\mathrm{km/s} \approxeq 0.277\,\mathrm{m/s}$.

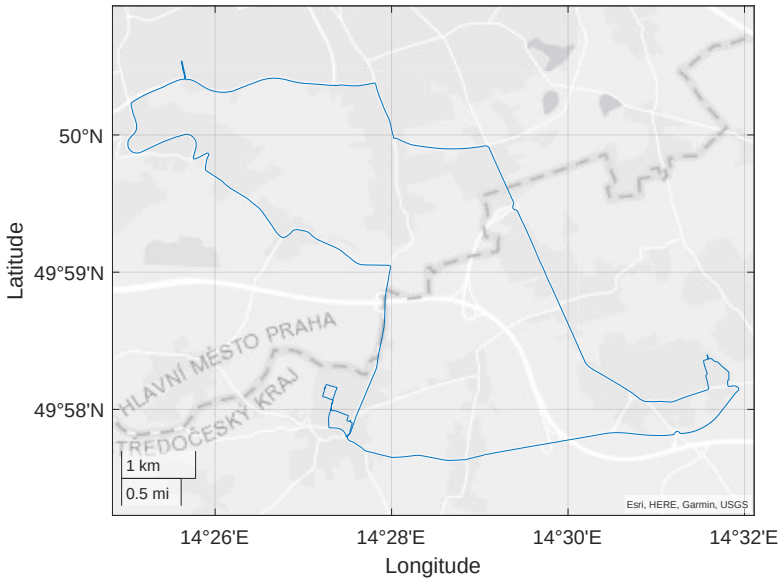
Na obrázku 8.4 jsou zobrazeny získané výškové hodnoty získané z GNSS a mapových podkladů. Je na něm dobře patrná odchylka a také omezený

rozsah dostupnosti mapových podkladů, jelikož experiment probíhal na okraji Prahy.

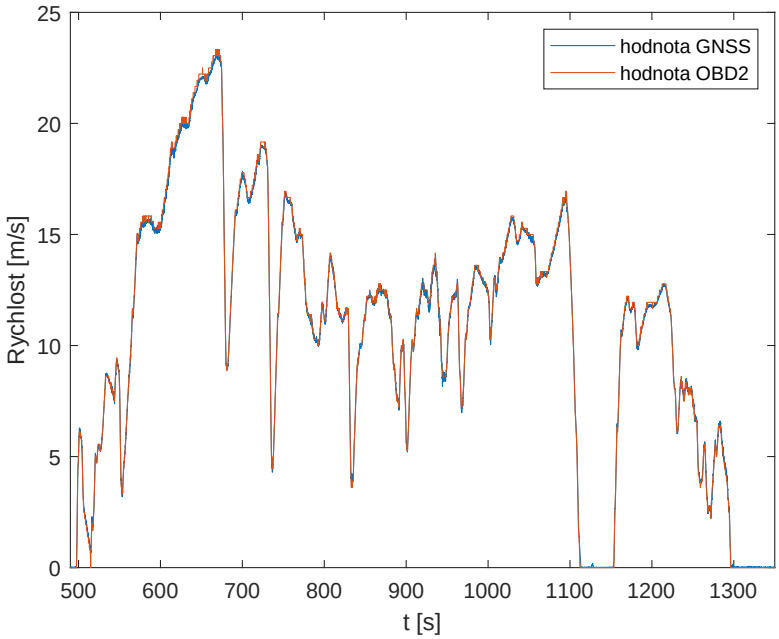
Mapa projeté trasy je na obrázku 8.2, ukázka dat z akcelerometrů je na obrázku 8.5.



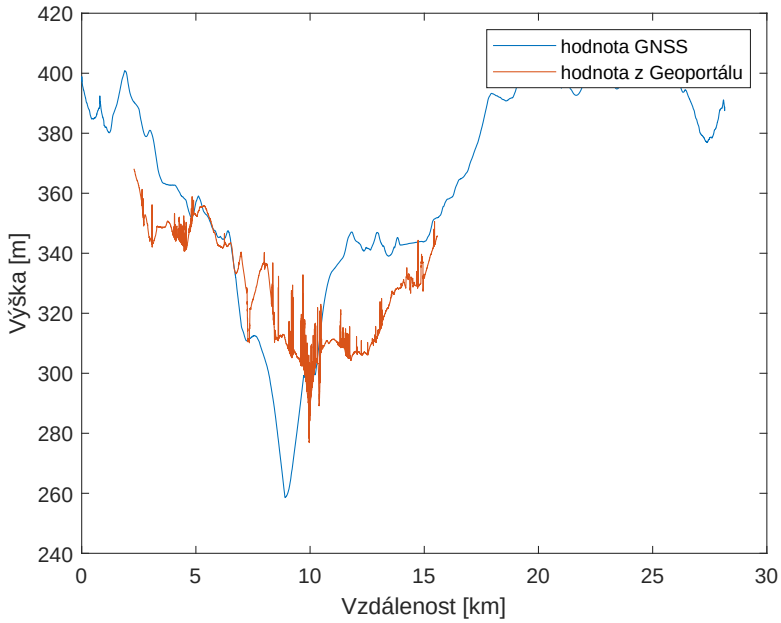
Obrázek 8.1: Topologie při experimentu



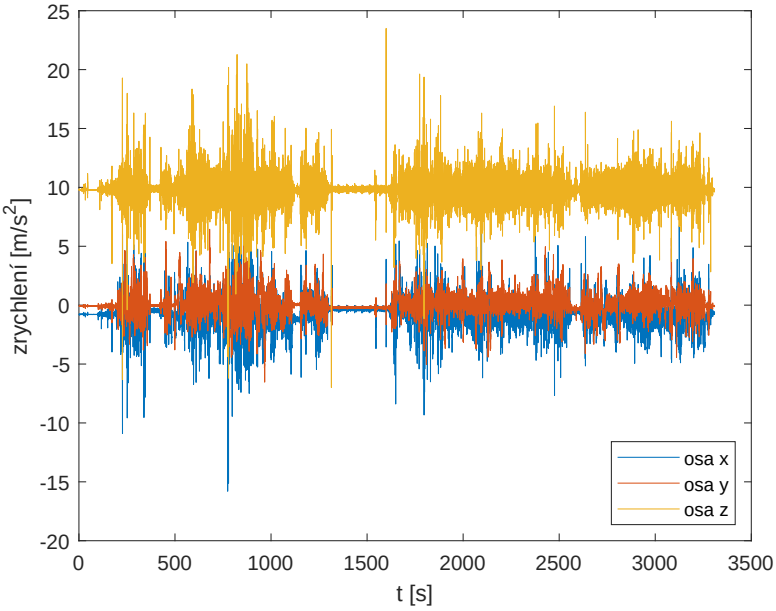
Obrázek 8.2: Projetá trasa při experimentu



Obrázek 8.3: Příklad průběhů rychlosti dle GNSS a `OBD 2`, jedná se o výběr z dat, aby byly patrné rozdíly v hodnotách



Obrázek 8.4: Průběhy výškových údajů získaných z GNSS a mapových podkladů z kapitoly 6.



Obrázek 8.5: Měření akcelerometrů z IMU, na ose z je dobře patrná hodnota $9.8\,\mathrm{m/s^2}$

Kapitola 9

Závěr

9.1 Úspěšně dokončné části

V průběhu práce se podařilo vytvořit spolehlivý způsob sběru dat z palubní odometrie a dalších senzorů. Bylo dosaženo praktičtějšího a spolehlivějšího propojení s prostředím *Matlab* a *Simulink*. Problémy s touto integrací byly vyřešeny a popsány jak v této práci, tak v přiložené dokumentaci.

V rámci práce došlo k výběru platformy ODROID H3+, zejména kvůli podpoře architektury `x86-64`. Dále se na tuto platformu navázaly senzory GNSS, odometrie a IMU. Podařilo se také vytvořit a standardizovat prostředí *ROS 2* pro záznam a sdílení naměřených dat v reálném čase.

Pro budoucí projekty by mělo být dosaženo značného snížení doby od začátku projektu po první získaná data. Jelikož se pro vývoj využívaly běžné a open–source zdroje a prostředky, měla by pro tento projekt být zaručena podpora do budoucna.

Systematický systém při tvorbě systému by měl zaručovat možnost použití komponent a softwarových bloků pro jiné projekty i na jiných platformách. Například IMU s vlastní interpretační deskou by měla být použitelná pro jakékoliv rozhraní USB.

Naměřená data jsou uložena tak aby umožňovala snadné použití a systém měření by měl zaručovat udržení tohoto standardu.

Navržené skripty a programy usnadňují instalaci a zjednodušují práci se senzory tak, aby nebyla nutná vnitřní znalost fungování konkrétních senzorů.

9.2 Možnosti pro zlepšení

Vzhledem ke zkušenostem získaným během tohoto projektu bych zpětně změnil výběr platformy. Jedním z původních cílů byla možnost na této platformě používat kompilované a nekompilované skripty napsané pro *Matlab* a *Simulink*. Tento cíl byl dle mého názoru zbytečně ambiciózní, jelikož software *ROS 2* se ukázal jako více než schopný přenášet data v reálném čase skrze Ethernet. Tento přenos by umožnoval spouštění náročnějších algoritmů na externím zařízení.

Tato možnost by umožňovala užití jednodušší platformy na architektuře

`ARM-64`, která by zaručovala lepší konektivitu a nižší spotřebu energie pro snadnější napájení.

Dalším zlepšením by bylo převedení celého projektu na real–time platformu, aby se dosáhlo nejen předpokládané latence, ale garance těchto latencí.

Příloha A

Údaje o platformách

Podrobnější tabulky jsou k dispozici v souboru `platforms.ods` v databázi práce.

		Raspberry PI 4B 4GB	*ODROID H3+*	*ODROID M1 8GB*	*ODROID N2+ 4GB*
architektura	`ARM64`	`x86-64`	`ARM64`	`ARM64`	`ARM64`
cena	CZK	1 700	5 500	2 900	2 700
# jader	-	4	4	4	4 (+2)
max. frekv. CPU	MHz	2147	3300	2000	2400
max. RAM	GB	4	64	8	4
max. úložiště	GB	n/a	n/a	n/a	n/a
UART	-	1	1	2	1
SPI	-	1	0	1	1
I2C	-	1	2	2	2
CAN	-	0	0	0	0
max. GPIO	-	28	0 (+ 2)	25	25
PWM	-	4	0	0	0
Ethernet	-	1	2	1	1
USB	-	4	4 (+ 1)	4	5

		NVIDIA Jetson AGX Orin 64GB Developer Kit	*NVIDIA Jetson Nano Developer Kit*	*SeeedStudio Jetson AGX Orin 32GB H01 Kit*	*BeagleBone AI–64*	*Solidrun HoneyComb LX2*
architektura	`ARM64`	`ARM64`	`ARM64`	`ARM64`	`ARM64`	`ARM64`
cena	CZK	65000	4800	43000	4000	20000
# jader	-	12	6	8	2	16
max. frekv. CPU	MHz	2200	1500	2200	2000	2000
max. RAM	GB	64	8	32	4	64
max. úložiště	GB	64	n/a	64	16	16
UART	-	1	1	1	4 (+ 1 TX)	0
SPI	-	1	1	2	2	0
I2C	-	2	2	2	3	0
CAN	-	2	0	3	2	0
max. GPIO	-	26	22	31	65	0
PWM	-	2	0	2	6	0
Ethernet	-	1	1	2	1	5
USB	-	7	4	7	2	6

Příloha B

Instalace *ROS 2* na *Windows 10/11*

B.1 Instalace přímo na *Windows*

Microsoft poměrně bez povšimnutí vydal vlastní podporu pro *ROS 2* jako součást iniciativy *Microsoft IoT*. V rámci testů jsem provedl instalace obou verzí *ROS 2 Foxy* a *ROS 2 Humble*, obě fungují spolehlivě s prostředím *Matlab*. Testy byly provedeny na obou systémech *Windows 10* a *Windows 11*. S nástroji kompatibility by neměl být problém s využitím novějšího systému i v případě že není explicitně podporován.

Instalace je kompletně popsána na stránce `https://ms-iot.github.io/ROSOnWindows/GettingStarted/SetupRos2.html`.

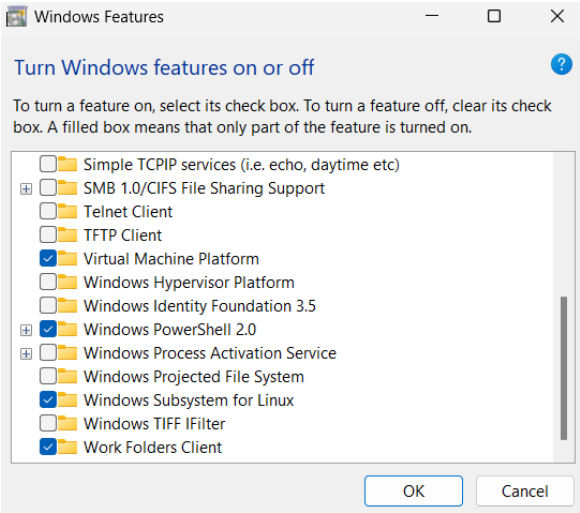
Pro instalaci je nejprve nutné mít připravené *Microsoft Visual Studio 2019*, je důležité zvolit právě tuto verzi, jelikož je nutná zpětná kompatibilita. Jako balík potřebný k instalaci je nutné zvolit *Desktop development with C++*. Instalaci je možné provést např. z adresy `https://visualstudio.microsoft.com/cs/vs/older-downloads/`.

Pokud chcete instalovat jinou verzi než *Foxy* na konci postupu vyměnte příkaz pro instalaci z `ros-foxy-desktop` na `ros-humble-desktop`, nebo jinou, Vámi zvolenou verzi.

B.2 Instalace na *WSL*

Další možností instalace na *WSL – Windows Subsystem for Linux*. Je dná se o nejjednodušší metodu pro jednoduchou instalaci a testování s téměř plnohodnotým prostředím. Při použití *WSL 2* je možné nainstalovat i plnohodnotné prostředí pro *MATLAB* a *Simulink*. Pro tuto verzi je ale problematické se připojit na síť, aniž by se musela provádět náročná konfigurace.

Pro jednodušší konfiguraci je vhodné využít *WSL 1*, který je sice výkonově slabší, ale je lépe integrovaný do *Windows*. Jelikož sdílí síťové prostředky s mateřským operačním systémem, není nutné komplikovaně konfigurovat propojení. Tato volba je jednodušší pro vývoj. *WSL* je pro některé verze *Windows* předinstalováno. Popřípadě je možné ho nainstalovat v okně Windows Features jako na obr. B.1.



Obrázek B.1: Instalace WSL

Samotný operační systém je doporučeno stáhnout z *Microsoft Store*. Tato instalace poskytuje nejjednodušší průběh a setkal jsem se s nejmenším množstvím chyb. Instalace *ROS 2* je poměrně přímočará, případně je dostupný skript popsaný v kapitole 8.

Pro propojení s externím zařízením USB je možné využít například `USBIPD-WIN` [27], čímž vytvoříme virtuální sériový port propojený s vnějším zařízením.

Příloha C

Literatura

[1] Amazon.ca. Sparkfun gps-rtk2 board - zed-f9p. https://www.amazon.ca/SparkFun-GPS-RTK2-Board-ZED-F9P-Qwiic/dp/B07NBPNWNZl, 2023. [Online; přístup 2023-05-10].

[2] Analog Devices. *Precision, Miniature MEMS IMU ADIS16505*, 4 2020. Rev. C.

[3] Analog Devices. Adis16505 datasheet and product info. https://www.analog.com/en/products/adis16505.html, 2023. [Online; přístup 2023-05-10].

[4] BeagleBoard.org. BeagleBone AI-64 Documentation. https://docs.beagleboard.org/latest/boards/beaglebone/ai-64/. [Online; přístup 2023-04-22].

[5] SEMU Consulting. pyubx2. https://github.com/semuconsulting/pyubx2, 2023.

[6] ELM Electronics. *ELM327, OBD to RS232 Interpreter*, 7 2016.

[7] FarSounder, Inc. protobuf-matlab. https://github.com/farsounder/protobuf-matlab, 2012.

[8] Eclipse Foundation. eCAL. https://eclipse-ecal.github.io/ecal/. [Online; přístup 2023-04-22].

[9] Raspberry Pi Foundation. Raspberry PI 4 model B specifications. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/. [Online; přístup 2023-04-22].

[10] Geoportál Praha. Digitální model povrchu - 1m. https://www.geoportalpraha.cz/cs/data/otevrena-data/1164EC41-5008-4824-B15B-3D033FED9DD1, 2022.

[11] Petr Holub. ubx2nmea. https://github.com/holubp/ubx2nmea, 2018.

[12] Zdeněk Hrdina. Transformace souřadnic ze systému wgs-84 do systému s-jtsk. https://www.ibot.cas.cz/personal/wild/data/WGS_JTSK.pdf, 1997.

[13] Pešek, Ivan Kostelecký, Jan, Kostelecký, Jakub. Metodika převodu mezi ETRF2000 a S-JTSK varianta 2. https://www.cuzk.cz/Zememerictvi/Geodeticke-zaklady-na-uzemi-CR/GNSS/Nova-realizace-systemu-ETRS89-v-CR/Metodika-prevodu-ETRF2000-vs-S-JTSK-var2(101208).aspx, 2010.

[14] Mathworks. ROS Toolbox Documentation. https://www.mathworks.com/help/ros/. [Online; přístup 2023-04-28].

[15] Michal Sojka. On generating Linux applications from Simulink. https://rtime.ciirc.cvut.cz/~sojka/blog/on-generating-linux-applications-from-simulink/.

[16] NVIDIA. Nvidia Jetson Orin. https://www.seeedstudio.com/AGX-Orin-32GB-H01-Kit-p-5569.html. [Online; přístup 2023-04-22].

[17] ODROID. ODROID-H3+ Specifiacations. https://wiki.odroid.com/odroid-h3/. [Online; přístup 2023-04-22].

[18] ODROID. ODROID-M1 Specifiacations. https://wiki.odroid.com/odroid-m1/. [Online; přístup 2023-04-22].

[19] ODROID. ODROID-N2+ Specifiacations. https://wiki.odroid.com/odroid-n2/. [Online; přístup 2023-04-22].

[20] ROS. ROS. https://www.ros.org/. [Online; přístup 2023-04-21].

[21] ROS. ROS2 Platform Suppport. https://www.ros.org/reps/rep-2000.html. [Online; přístup 2023-04-26].

[22] SeeedStudio. Jetson Orin. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/. [Online; přístup 2023-04-22].

[23] SolidRun. SolidRun HoneyComb LX2 Start Guide. https://solidrun.atlassian.net/wiki/spaces/developer/pages/197494288/HoneyComb+LX2+ClearFog+CX+LX2+Quick+Start+Guide. [Online; přístup 2023-04-22].

[24] TCNOpen. TCNOpen. https://www.tcnopen.eu/. [Online; přístup 2023-04-26].

[25] TCNOpen. TCNOpen TRDP. https://sourceforge.net/projects/tcnopen/. [Online; přístup 2023-04-26].

[26] u-blox. *u-blox 8 / u-blox M8, Receiver description*, 1 2023. Rev. R28.

[27] Frans van Dorsselaer. usbipd-win. https://github.com/dorssel/usbipd-win, 2023.

[28] Wikipedia. Bessel ellipsoid — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Bessel_ellipsoid, 2023. [Online; přístup 2023-05-10].

[29] Wikipedia. OBD-II PIDs — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/OBD-II_PIDs, 2023. [Online; přístup 2023-04-11].

[30] Wikipedia. On-board diagnostics — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/On-board_diagnostics, 2023. [Online; přístup 2023-04-11].

[31] Wikipedia. USB — Wikipedia, the free encyclopedia. http://cs.wikipedia.org/w/index.php?title=USB&oldid=22341265, 2023. [Online; přístup 2023-05-01].

[32] Wikipedia. World Geodetic System — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/World_Geodetic_System, 2023. [Online; přístup 2023-05-10].